Package ‘lfl’

February 11, 2020

Type Package
Title Linguistic Fuzzy Logic
Version 2.0
Date 2020-02-11
Author Michal Burda
Maintainer Michal Burda <michal.burda@osu.cz>
Description Various algorithms related to linguistic fuzzy logic: mining for linguistic fuzzy association rules, composition of fuzzy relations, performing perception-based logical deduction (PbLD), and forecasting time-series using fuzzy rule-based ensemble (FRBE).
License GPL-3
Suggests testthat, doMC, knitr, rmarkdown
Depends R (>= 3.4.0)
Imports Rcpp (>= 0.12.12), foreach, forecast (>= 5.5), plyr, tseries, e1071, zoo, utils
LinkingTo Rcpp
NeedsCompilation yes
SystemRequirements C++11
Encoding UTF-8
RoxygenNote 7.0.2
Repository CRAN
Date/Publication 2020-02-11 12:50:05 UTC

R topics documented:

aggregateConsequents .................................................. 3
algebra ................................................................. 5
antecedents ............................................................ 8
as.data.frame.farules .................................................. 9
as.data.frame.fsets .................................................... 10
R topics documented:

c.farules ......................................................... 11
cbind.fsets .................................................... 12
compose .......................................................... 13
consequents ...................................................... 15
ctx ................................................................. 16
defaultHedgeParams ........................................ 20
defuzz ............................................................. 21
evalfrbe ........................................................ 22
farules ............................................................ 23
fcut ................................................................. 24
fire ................................................................. 27
frbe ................................................................. 29
fsets ............................................................... 31
hedge ............................................................... 33
horizon ........................................................... 35
is.farules ........................................................ 36
is.frbe ............................................................ 37
is.fsets ........................................................... 38
is.specific ........................................................ 38
lcut ................................................................. 40
lcut3 .............................................................. 44
lf1 ................................................................. 46
lingexpr .......................................................... 47
mase ............................................................... 49
minmax ........................................................... 50
mult ................................................................. 51
pbld ................................................................. 52
perceive .......................................................... 54
plot.fsets ....................................................... 55
print.farules .................................................... 56
print.frbe ....................................................... 57
print.fsets ....................................................... 58
rbcoverage ...................................................... 59
reduce ............................................................ 60
rmse ............................................................... 62
searchrules ...................................................... 62
slices ............................................................. 65
smape ............................................................ 66
sobocienski ..................................................... 67
triangle .......................................................... 69
triangular ......................................................... 70

Index ......................................................... 72
aggregateConsequents

Aggregation of fired consequents into a resulting fuzzy set

Description

Take a character vector of consequent names, a numeric vector representing the degree of consequents’ firing and a matrix that models fuzzy sets corresponding to the consequent names, and perform an aggregation of the consequents into a resulting fuzzy set.

Usage

```r
aggregateConsequents(
  conseq,
  degrees,
  partition,
  firing = lukas.residuum,
  aggreg = pgoedel.tnorm
)
```

Arguments

- **conseq**: A character vector of consequents. Each value in the vector must correspond to a name of some column of the partition matrix. The length of this vector must be the same as of the degrees argument.
- **degrees**: A numeric vector of membership degrees at which the corresponding consequents (see the conseq argument) are fired.
- **partition**: A matrix of membership degrees that describes the meaning of the consequents in vector conseq: each column of the matrix corresponds to a fuzzy set that models a single consequent (of a name given by column names of the matrix), each row corresponds to a single crisp value (which is not important for this function), hence each cell corresponds to a membership degree in which the crisp value is a member of a fuzzy set modelling the consequent. Each consequent in conseq must correspond to some column of this matrix. Such matrix may be created e.g. by using `fcut()` or `lcut()` functions.
- **firing**: A two-argument function used to compute the resulting truth value of the consequent. Function is evaluated for each consequent in conseq, with corresponding degrees value as the first argument and corresponding truth-value of the consequent (from partition) as the second argument. In default, the Lukasiewicz residuum (lukas.residuum()) is evaluated that way.
- **aggreg**: An aggregation function to be used to combine fuzzy sets resulting from firing the consequents with the firing function. The function should accept multiple numeric vectors of membership degrees as its arguments. In default, the Goedel t-norm (pgoedel.tnorm()) is evaluated.
aggregateConsequents

Details

This function is typically used within an inference mechanism, after a set of firing rules is determined and membership degrees of their antecedents are computed, to combine the consequents of the firing rules into a resulting fuzzy set. The result of this function is then typically defuzzified (see `defuzz()`) to obtain a crisp result of the inference.

Function assumes a set of rules with antecedents firing at degrees given in degrees and with consequents in conseq. The meaning of the consequents is modeled with fuzzy sets whose membership degree values are captured in the partition matrix.

With default values of firing and aggreg parameters, the function computes a fuzzy set that results from a conjunction (Goedel minimum t-norm) of all provided implicative (Lukasiewicz residuum) rules.

In detail, the function first computes the fuzzy set of each fired consequent by calling `part[i] <- firing(degrees[i], partition[, conseq[i]])` for each i-th consequent and the results are aggregated using the aggreg parameter: `aggreg(part[1], part[2], ...)`. In order to aggregate consequents in a Mamdani-Assilian’s fashion, set firing to `pgoedel.tnorm()` and aggreg to `pgoedel.tconorm()`.

Value

A vector of membership degrees of fuzzy set elements that correspond to rows in the partition matrix. If empty vector of consequents is provided, vector of 1’s is returned. The length of the resulting vector equals to the number of rows of the partition matrix.

Author(s)

Michal Burda

See Also

`fire()`, `perceive()`, `defuzz()`, `fcut()`, `lcut()`

Examples

```r
# create a partition matrix
partition <- matrix(c(0:10/10, 10:0/10, rep(0, 5),
                     rep(0, 5), 0:10/10, 10:0/10,
                     0:12/12, 1, 12:0/12),
                     byrow=FALSE,
                     ncol=3)
colnames(partition) <- c('a', 'b', 'c')

# the result of aggregation is equal to:
# pmin(1, partition[, 1] + (1 - 0.5), partition[, 2] + (1 - 0.8))
aggregateConsequents(c('a', 'b'), c(0.5, 0.8), partition)
```
Description

Compute triangular norms (t-norms), triangular conorms (t-conorms), residua, bi-residua, and negations.

Usage

\[
\text{algebra(name, stdneg = FALSE, \ldots)}
\]

\[
\text{is.algebra(a)}
\]

\[
\text{goedel.tnorm(\ldots)}
\]

\[
\text{lukas.tnorm(\ldots)}
\]

\[
\text{goguen.tnorm(\ldots)}
\]

\[
\text{pgoedel.tnorm(\ldots)}
\]

\[
\text{plukas.tnorm(\ldots)}
\]

\[
\text{pgoguen.tnorm(\ldots)}
\]

\[
\text{goedel.tconorm(\ldots)}
\]

\[
\text{lukas.tconorm(\ldots)}
\]

\[
\text{goguen.tconorm(\ldots)}
\]

\[
\text{pgoedel.tconorm(\ldots)}
\]

\[
\text{plukas.tconorm(\ldots)}
\]

\[
\text{pgoguen.tconorm(\ldots)}
\]

\[
\text{goedel.residuum(x, y)}
\]

\[
\text{lukas.residuum(x, y)}
\]

\[
\text{goguen.residuum(x, y)}
\]

\[
\text{goedel.biresiduum(x, y)}
\]

\[
\text{lukas.biresiduum(x, y)}
\]
goguen.biresiduum(x, y)

invol.neg(x)

strict.neg(x)

Arguments

name
   The name of the algebra to be created. Must be one of: "goedel", "lukasiewicz", "goguen" (or an unambiguous abbreviation).

stdneg
   TRUE if to force the use of a "standard" negation (i.e. involutive negation). Otherwise, the appropriate negation is used in the algebra (e.g. strict negation in Goedel and Goguen algebra and involutive negation in Lukasiewicz algebra).

... For t-norms and t-conorms, these arguments are numeric vectors of values to compute t-norms or t-conorms from. Values outside the [0, 1] interval cause an error. NA values are also permitted.

For the algebra() function, these arguments are passed to the factory functions that create the algebra. (Currently unused.)

a
   An object to be checked if it is a valid algebra (i.e. a list returned by the algebra function).

x
   Numeric vector of values to compute a residuum or bi-residuum from. Values outside the [0, 1] interval cause an error. NA values are also permitted.

y
   Numeric vector of values to compute a residuum or bi-residuum from. Values outside the [0, 1] interval cause an error. NA values are also permitted.

Details

goedel.tnorm, lukas.tnorm, and goguen.tnorm compute the Goedel, Lukasiewicz, and Goguen triangular norm (t-norm) from all values in the arguments. If the arguments are vectors they are combined together firstly so that a numeric vector of length 1 is returned.

pgoedel.tnorm, plukas.tnorm, and pgoguen.tnorm compute the same t-norms, but in an element-wise manner. I.e. the values with indices 1 of all arguments are used to compute the t-norm, then the second values (while recycling the vectors if they do not have the same size) so that the result is a vector of values.

goedel.tconorm, lukas.tconorm, goguen.tconorm, are similar to the previously mentioned functions, except that they compute triangular conorms (t-conorms). pgoedel.tconorm, plukas.tconorm, and pgoguen.tconorm are their element-wise alternatives.

goedel.residuum, lukas.residuum, and goguen.residuum compute residua (i.e. implications) and goedel.biresiduum, lukas.biresiduum, and goguen.biresiduum compute bi-residua. Residua and bi-residua are computed in an element-wise manner, for each corresponding pair of values in x and y arguments.

invol.neg and strict.neg compute the involutive and strict negation, respectively.

Let a, b be values from the interval [0, 1]. The realized functions can be defined as follows:

- Goedel t-norm: \( \min\{a, b\} \);
• Goguen t-norm: $ab$;
• Łukasiewicz t-norm: $\max\{0, a + b - 1\}$;
• Goedel t-conorm: $\max\{a, b\}$;
• Goguen t-conorm: $a + b - ab$;
• Łukasiewicz t-conorm: $\min\{1, a + b\}$;
• Goedel residuum (standard Goedel implication): $1$ if $a \leq b$ and $b$ otherwise;
• Goguen residuum (implication): $1$ if $a \leq b$ and $b/a$ otherwise;
• Łukasiewicz residuum (standard Łukasiewicz implication): $1$ if $a \leq b$ and $1 - a + b$ otherwise;
• Involutive negation: $1 - x$;
• Strict negation: $1$ if $x = 0$ and $0$ otherwise.

Bi-residuum $B$ is derived from t-norm $T$ and residuum $R$ as follows:

$$B(a, b) = T(R(a, b), R(b, a)).$$

The arguments have to be numbers from the interval $[0, 1]$. Values outside that range cause an error. NaN values are treated as NAs.

If some argument is NA or NaN, the result is NA.

`algebra` returns a named list of functions that together form Goedel, Goguen, or Łukasiewicz algebra:

- "goedel": strict negation and Goedel t-norm, t-conorm, residuum, and bi-residuum;
- "goguen": strict negation and Goguen t-norm, t-conorm, residuum, and bi-residuum;
- "lukasiewicz": involutive negation and Łukasiewicz t-norm, t-conorm, residuum, and bi-residuum. Moreover, `algebra` returns supremum and infimum functions computed as maximum and minimum, respectively.

`is.algebra` tests whether the given argument is a valid algebra, i.e. a list returned by the `algebra` function.

**Value**

Functions for t-norms and t-conorms (such as `goedel.tnorm`) return a numeric vector of size 1 that is the result of the appropriate t-norm or t-conorm applied on all values of all arguments.

Element-wise versions of t-norms and t-conorms (such as `pgoedel.tnorm`) return a vector of results after applying the appropriate t-norm or t-conorm on argument in an element-wise (i.e. by indices) way. The resulting vector is of length of the longest argument (shorter arguments are recycled).

Residua and bi-residua functions return a numeric vector of length of the longest argument (shorter argument is recycled).

`strict.neg` and `invol.neg` compute negations and return a numeric vector of the same size as the argument $x$.

`algebra` returns a list of functions of the requested algebra: "n" (negation), "t" (t-norm), "pt" (element-wise t-norm), "c" (t-conorm), "pc" (element-wise t-conorm), "r" (residuum), "b" (bi-residuum), "s" (supremum), "ps" (element-wise supremum), "i" (infimum), and "pi" (element-wise infimum).
Author(s)

Michal Burda

Examples

```r
# direct and element-wise version of functions
goedel.tnorm(c(0.3, 0.2, 0.5), c(0.8, 0.1, 0.5)) # 0.1
pgoedel.tnorm(c(0.3, 0.2, 0.5), c(0.8, 0.1, 0.5)) # c(0.3, 0.1, 0.5)

# algebras
x <- runif(10)
y <- runif(10)
a <- algebra('goedel')
a$n(x)  # negation
a$t(x, y) # t-norm
a$pt(x, y) # element-wise t-norm
a$c(x, y) # t-conorm
a$pc(x, y) # element-wise t-conorm
a$r(x, y) # residuum
a$b(x, y) # bi-residuum
a$s(x, y) # supremum
a$ps(x, y) # element-wise supremum
a$i(x, y) # infimum
a$pi(x, y) # element-wise infimum

is.algebra(a) # TRUE
```

antecedents

Extract antecedent-part (left-hand side) of rules in a list

Description

Given a list of rules or an instance of the S3 `farules()` class, the function returns a list of their antecedents (i.e. left-hand side of rules).

Usage

`antecedents(rules)`

Arguments

- `rules` Either a list of character vectors or an object of class `farules()`.

Details

This function assumes rules to be a valid `farules()` object or a list of character vectors where the first element of each vector is a consequent part and the rest is an antecedent part of rules. Function returns a list of antecedents.
Value

A list of character vectors.

Author(s)

Michal Burda

See Also

consequents(), farules(), searchrules()

Examples

```r
rules <- list(c('a', 'b', 'c'), c('d'), c('a', 'e'))
antecedents(rules)
```

as.data.frame.farules

Convert the instance of the farules() S3 class into a data frame.

Empty farules() object is converted into an empty data.frame().

Description

Convert the instance of the farules() S3 class into a data frame. Empty farules() object is converted into an empty data.frame().

Usage

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

Arguments

x  An instance of class farules() to be transformed.

...  Unused.

Value

A data frame of statistics of the rules that are stored in the given farules() object. Row names of the resulting data frame are in the form: A1 & A2 & ... & An => C, where Ai are antecedent predicates and C is a consequent. An empty farules() object is converted into an empty data.frame() object.

Author(s)

Michal Burda

See Also

farules(), searchrules()
as.data.frame.fsets  

Convert an object of `fsets` class into a matrix or data frame. This function converts an instance of S3 class `fsets` into a matrix or a data frame. The `vars()` and `specs()` attributes of the original object are deleted.

### Description

Convert an object of `fsets` class into a matrix or data frame. This function converts an instance of S3 class `fsets` into a matrix or a data frame. The `vars()` and `specs()` attributes of the original object are deleted.

### Usage

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

## S3 method for class 'fsets'
as.matrix(x, ...)
```

### Arguments

- `x`  
  An instance of class `fsets` to be converted

- `...`  
  arguments further passed to `as.data.frame` after converting to matrix in `as.data.frame.fsets`. Unused in `as.matrix.fsets`.

### Value

A numeric matrix or data frame of membership degrees.

### Author(s)

Michal Burda

### See Also

`fsets()`, `fcut()`, `lcut()`
c.farules

Take a sequence of instances of S3 class `farules()` and combine them into a single object. An error is thrown if some argument does not inherit from the `farules()` class.

Description

Take a sequence of instances of S3 class `farules()` and combine them into a single object. An error is thrown if some argument does not inherit from the `farules()` class.

Usage

```r
## S3 method for class 'farules'
c(..., recursive = FALSE)
```

Arguments

- `...` A sequence of objects of class `farules()` to be concatenated.
- `recursive` This argument has currently no function and is added here only for compatibility with generic `c` function.

Value

An object of class `farules()` that is created by merging the arguments together, i.e. by concatenating the rules and row-binding the statistics of given objects.

Author(s)

Michal Burda

See Also

`farules()`, `searchrules()`

Examples

```r
ori1 <- farules(rules=list(letters[1:3],
letters[2:5]),
statistics=matrix(runif(16), nrow=2))
ori2 <- farules(rules=list(letters[4],
letters[3:8]),
statistics=matrix(runif(16), nrow=2))
res <- c(ori1, ori2)
print(res)
```
cbind.fsets  Combine several 'fsets' objects into a single one

Description

Take a sequence of objects of class 'fsets' and combine them by columns. This version of cbind takes care of the `vars()` and `specs()` attributes of the arguments and merges them to the result. If some argument does not inherit from the 'fsets' class, an error is thrown.

Usage

```r
## S3 method for class 'fsets'
cbind(..., deparse.level = 1, warn = TRUE)
```

Arguments

- `...` A sequence of objects of class 'fsets' to be merged by columns.
- `deparse.level` This argument has currently no function and is added here only for compatibility with generic `cbind()` function.
- `warn` Whether to issue warning when combining two fsets having the same vars about the fact that specs may not be accurate

Details

The `vars()` attribute is merged by concatenating the `vars()` attributes of each argument. Also the `specs()` attributes of the arguments are merged together.

Value

An object of class 'fsets' that is created by merging the arguments by columns. Also the arguments' attributes `vars()` and `specs()` are merged together.

Author(s)

Michal Burda

See Also

`vars()`, `specs()`, `fcut()`, `lcut()`

Examples

```r
d1 <- fcut(CO2[, 1:2])
d2 <- fcut(CO2[, 3:4], breaks=list(conc=1:4*1000/4))
r <- cbind(d1, d2)
print(colnames(d1))
print(colnames(d2))
```
compose

print(colnames(r))
print(vars(d1))
print(vars(d2))
print(vars(r))
print(specs(d1))
print(specs(d2))
print(specs(r))

---

**compose**  Composition of Fuzzy Relations

**Description**
Composition of Fuzzy Relations

**Usage**

```r
compose(
  x, 
  y, 
  e = NULL, 
  alg = c("goedel", "goguen", "lukasiewicz"),
  type = c("basic", "sub", "super", "square"),
  quantifier = NULL,
  sorting = sort
)
```

**Arguments**

- **x**  A first fuzzy relation to be composed. It must be a numeric matrix with values within the \([0, 1]\) interval. The number of columns must match with the number of rows of the \(y\) matrix.
- **y**  A second fuzzy relation to be composed. It must be a numeric matrix with values within the \([0, 1]\) interval. The number of columns must match with the number of rows of the \(x\) matrix.
- **e**  An excluding fuzzy relation. If not NULL, it must be a numeric matrix with dimensions equal to the \(y\) matrix.
- **alg**  An algebra to be used for composition. It must be one of 'goedel' (default), 'goguen', or 'lukasiewicz', or an instance of class `algebra` (see `algebra()`).
- **type**  A type of a composition to be performed. It must be one of 'basic' (default), 'sub', 'super', or 'square'.
- **quantifier**  If not NULL, it must be a function taking a single argument, a vector of relative cardinalities, that would be translated into membership degrees. A result of the `lingexpr()` function is a good candidate for that.
Sorting function used within quantifier application. The given function must sort the membership degrees and allow the decreasing argument as in `base::sort()`. This function have to be explicitly specified typically if performing compositions that handle NA values.

Details

Function composes a fuzzy relation \( x \) (i.e. a numeric matrix of size \( (u, v) \)) with a fuzzy relation \( y \) (i.e. a numeric matrix of size \( (v, w) \)) and possibly with the use of an exclusion fuzzy relation \( e \) (i.e. a numeric matrix of size \( (v, w) \)).

The style of composition is determined by the algebra \( \text{alg} \), the composition type \( \text{type} \), and possibly also by a quantifier.

Value

A matrix with \( v \) rows and \( w \) columns, where \( v \) is the number of rows of \( x \) and \( w \) is the number of columns of \( y \).

Author(s)

Michal Burda

See Also

`[algebra(), mult(), lingexpr()]`

Examples

```r
R <- matrix(c(0.1, 0.6, 1, 0, 0, 0,
              0, 0.3, 0.7, 0.9, 1, 1,
              0, 0.6, 0.8, 1, 0,
              0, 1, 0.5, 0, 0, 0,
              0, 0, 1, 1, 0, 0), byrow=TRUE, nrow=5)

S <- matrix(c(0.9, 1, 0.9, 1,
              1, 1, 1, 1,
              0.1, 0.2, 0, 0.2,
              0, 0, 0, 0,
              0.7, 0.6, 0.5, 0.4,
              1, 0.9, 0.7, 0.6), byrow=TRUE, nrow=6)

RS <- matrix(c(0.6, 0.6, 0.6, 0.6,
               0.6, 0.6, 0.6, 0.6,
               0.7, 0.6, 0.5, 0.4,
               0.8, 0.7, 0.6, 0.7,
               1, 0.5, 0.4, 0.3), byrow=TRUE, nrow=6)

compose(R, S, alg='goedel', type='basic') # should be equal to RS
```
consequents

Extract consequent-part (right-hand side) of rules in a list

Description

Given a list of rules or an instance of the S3 \texttt{farules()} class, the function returns a list of their consequents (i.e. right-hand side of rules).

Usage

\texttt{consequents(rules)}

Arguments

\texttt{rules} \\
Either a list of character vectors or an object of class \texttt{farules()}. 

Details

This function assumes \texttt{rules} to be a valid \texttt{farules()} object or a list of character vectors where the first element of each vector is a consequent part and the rest is an antecedent part of rules. Function returns a list of consequents.

Value

A list of character vectors.

Author(s)

Michal Burda

See Also

\texttt{antecedents()}, \texttt{farules()}, \texttt{searchrules()}

Examples

```r
rules <- list(c('a', 'b', 'c'), c('d'), c('a', 'e'))
consequents(rules)
unlist(consequents(rules))  # as vector
```
Description

A context describes a range of allowed values for a data column.

Usage

```r
ctx3(
  low = 0,
  center = low + (high - low) * relCenter,
  high = 1,
  relCenter = 0.5
)
```

```r
ctx3bilat(
  negMax = -1,
  negCenter = origin + (negMax - origin) * relCenter,
  origin = 0,
  center = origin + (max - origin) * relCenter,
  max = 1,
  relCenter = 0.5
)
```

```r
ctx5(
  low = 0,
  lowerCenter = mean(c(low, center)),
  center = low + (high - low) * relCenter,
  upperCenter = mean(c(center, high)),
  high = 1,
  relCenter = 0.5
)
```

```r
ctx5bilat(
  negMax = -1,
  negUpperCenter = mean(c(negCenter, negMax)),
  negCenter = origin + (negMax - origin) * relCenter,
  negLowerCenter = mean(c(origin, negCenter)),
  origin = 0,
  lowerCenter = mean(c(origin, center)),
  center = origin + (max - origin) * relCenter,
  upperCenter = mean(c(center, max)),
  max = 1,
  relCenter = 0.5
)
```
as.ctx3(x)

## S3 method for class 'ctx3'
as.ctx3(x)

## S3 method for class 'ctx3bilat'
as.ctx3(x)

## S3 method for class 'ctx5'
as.ctx3(x)

## S3 method for class 'ctx5bilat'
as.ctx3(x)

## Default S3 method:
as.ctx3(x)

as.ctx3bilat(x)

## S3 method for class 'ctx3bilat'
as.ctx3bilat(x)

## S3 method for class 'ctx3'
as.ctx3bilat(x)

## S3 method for class 'ctx5'
as.ctx3bilat(x)

## S3 method for class 'ctx5bilat'
as.ctx3bilat(x)

## Default S3 method:
as.ctx3bilat(x)

as.ctx5(x)

## S3 method for class 'ctx5'
as.ctx5(x)

## S3 method for class 'ctx3'
as.ctx5(x)

## S3 method for class 'ctx3bilat'
as.ctx5(x)

## S3 method for class 'ctx5bilat'
as.ctx5(x)
## Default S3 method:
as.ctx5(x)
as.ctx5bilat(x)

## S3 method for class 'ctx5bilat'
as.ctx5bilat(x)

## S3 method for class 'ctx3'
as.ctx5bilat(x)

## S3 method for class 'ctx3bilat'
as.ctx5bilat(x)

## S3 method for class 'ctx5'

## Default S3 method:
as.ctx5bilat(x)

is.ctx3(x)
is.ctx3bilat(x)
is.ctx5(x)
is.ctx5bilat(x)

### Arguments

- **low**  
  Lowest value of an unilateral context.
- **center**  
  A positive middle value of a bilateral context, or simply a middle value of an unilateral context.
- **high**  
  Highest value of an unilateral context.
- **relCenter**  
  A relative quantity used to compute the negCenter and/or center, if they are not specified explicitly. The sensible value is 0.5 for context symmetric around center, or 0.42 as proposed by Novak.
- **negMax**  
  Lowest negative value of a bilateral context.
- **negCenter**  
  A negative middle value.
- **origin**  
  Origin, i.e. the initial point of the bilateral context. It is typically a value of zero.
- **max**  
  Highest value of a bilateral context.
- **lowerCenter**  
  A typical positive value between origin and center.
- **upperCenter**  
  A typical positive value between center and maximum.
- **negUpperCenter**  
  A typical negative value between negMax and negCenter.
- **negLowerCenter**  
  A typical negative value between negCenter and negOrigin.
A value to be examined or converted. For as.ctx*, it can be an instance of any ctx* class or a numeric vector of size equal to the number of points required for the given context type.

Details

A context describes a range of allowed values for a data column. For that, only the borders of the interval, i.e. minimum and maximum, are usually needed, but we use contexts to hold more additional information that is crucial for the construction of linguistic expressions.

Currently, four different contexts are supported that determine the types of possible linguistic expressions, as constructed with lingexpr(). Unilateral or bilateral context is allowed in the variants of trichotomy or pentachotomy. Trichotomy distinguishes three points in the interval: the lowest value, highest value, and center. Pentachotomy adds lower center and upper center to them. As opposite to unilateral, the bilateral context handles explicitly the negative values. That is, bilateral context expects some middle point, the origin (usually 0), around which the positive and negative values are placed.

Concretely, the type of the context determines the allowed atomic expressions as follows:

- **ctx3**: trichotomy (low, center, high) enables atomic expressions: small, medium, big;
- **ctx5**: pentachotomy (low, lowerCenter, center, upperCenter, high) enables atomic expressions: small, lower medium, medium, upper medium, big;
- **ctx3bilat**: bilateral trichotomy (negMax, negCenter, origin, center, max) enables atomic expressions: negative big, negative medium, negative small, zero, small, medium, big;
- **ctx5bilat**: bilateral pentachotomy (negMax, negCenter, origin, center, max) enables atomic expressions: negative big, negative medium, negative small, zero, small, medium, big.

The as.ctx* functions return instance of the appropriate class. The functions perform the conversion so that missing points of the new context are computed from the old context that is being transformed. In the subsequent table, rows represent compatible values of different context types:

<table>
<thead>
<tr>
<th>ctx3</th>
<th>ctx5</th>
<th>ctx3bilat</th>
<th>ctx5bilat</th>
</tr>
</thead>
<tbody>
<tr>
<td>negMax</td>
<td>negMax</td>
<td>negUpperCenter</td>
<td>negMax</td>
</tr>
<tr>
<td>negCenter</td>
<td>negCenter</td>
<td>negLowerCenter</td>
<td>origin</td>
</tr>
<tr>
<td>low</td>
<td>low</td>
<td>origin</td>
<td>lowerCenter</td>
</tr>
<tr>
<td>center</td>
<td>center</td>
<td>center</td>
<td>center</td>
</tr>
<tr>
<td>high</td>
<td>high</td>
<td>max</td>
<td>max</td>
</tr>
</tbody>
</table>

The as.ctx* conversion is performed by replacing values by rows, as indicated in the table above. When converting from a context with less points to a context with more points (e.g. from unilateral to bilateral, or from trichotomy to pentachotomy), missing points are computed as follows:

- center is computed as a mean of origin (or low) and max (or high).
- lowerCenter is computed as a mean of origin (or low) and center.
- upperCenter is computed as a mean of max (or high) and center.
defaultHedgeParams

- negative points (such as negMax, negCenter etc.) are computed symmetrically around origin to the corresponding positive points.

The code as.ctx* functions allow the parameter to be also a numeric vector of size equal to the number of points required for the given context type, i.e. 3 (ctx3), 5 (ctx3bilat, ctx5), or 9 (ctx5bilat).

Value
ctx* and as.ctx* return an instance of the appropriate class. is.ctx* returns TRUE or FALSE.

Author(s)
Michal Burda

See Also

minmax(), lingexpr(), horizon(), hedge(), fcut(), lcut()

Examples

ctx3(low=0, high=10)
as.ctx3bilat(ctx3(low=0, high=10))

---

defaultHedgeParams A list of the parameters that define the shape of the hedges.

Description

A list of the parameters that define the shape of the hedges.

Usage
defaultHedgeParams

Format

An object of class list of length 9.
defuzz

Convert fuzzy set into a crisp numeric value

Description

Take a fuzzy set in the form of a vector of membership degrees and a vector of numeric values that correspond to that degrees and perform a selected type of defuzzification, i.e. conversion of the fuzzy set into a single crisp value.

Usage

defuzz(degrees, values, type = c("mom", "fom", "lom", "dee"))

Arguments

degrees  A fuzzy set in the form of a numeric vector of membership degrees of values provided as the values argument.
values    A universe for the fuzzy set.
type      Type of the requested defuzzification method. The possibilities are:

• 'mom': Mean of Maxima - maximum membership degrees are found and a mean of values that correspond to that degrees is returned;
• 'fom': First of Maxima - first value with maximum membership degree is returned;
• 'lom': Last of Maxima - last value with maximum membership degree is returned;
• 'dee': Defuzzification of Evaluative Expressions - method used by the pbld() inference mechanism that combines the former three approaches accordingly to the shape of the degrees vector: If degrees is non-increasing then 'lom' type is used, if it is non-decreasing then 'fom' is applied, else 'mom' is selected.

Details

Function converts input fuzzy set into a crisp value. The definition of input fuzzy set is provided by the arguments degrees and values. These arguments should be numeric vectors of the same length, the former containing membership degrees in the interval [0, 1] and the latter containing the corresponding crisp values: i.e., values[i] has a membership degree degrees[i].

Value

A defuzzified value.

Author(s)

Michal Burda
See Also
fire(), aggregateConsequents(), perceive(), pbld(), fcut(), lcut()

Examples

# returns mean of maxima, i.e., mean of 6, 7, 8
defuzz(c(0, 0, 0, 0.1, 0.3, 0.9, 0.9, 0.9, 0.2, 0),
1:10,
type='mom')

desc

Description
Take a FRBE forecast and compare it with real values using arbitrary error function.

Usage
evalfrbe(fit, real, error = c("smape", "mase", "rmse"))

Arguments

fit A FRBE model of class frbe as returned by the frbe() function.
real A numeric vector of real (known) values. The vector must correspond to the values being forecasted, i.e. the length must be the same as the horizon forecasted by frbe().
error Error measure to be computed. It can be either Symmetric Mean Absolute Percentage Error (smape), Mean Absolute Scaled Error (mase), or Root Mean Squared Error (rmse). See also smape(), mase(), and rmse() for more details.

Details
Take a FRBE forecast and compare it with real values by evaluating a given error measure. FRBE forecast should be made for a horizon of the same value as length of the vector of real values.

Value
Function returns a data.frame with single row and columns corresponding to the error of the individual forecasting methods that the FRBE is computed from. Additionally to this, a column "avg" is added with error of simple average of the individual forecasting methods and a column "frbe" with error of the FRBE forecasts.

Author(s)
Michal Burda
References
Štěpnička, M., Burda, M., Štěpničková, L. Fuzzy Rule Base Ensemble Generated from Data by Linguistic Associations Mining. FUZZY SET SYST. 2015.

See Also
frbe(), smape(), mase(), rmse()

Examples

# prepare data (from the forecast package)
library(forecast)
horizon <- 10
train <- wineind[-1 * (length(wineind)-horizon+1):length(wineind)]
test <- wineind[(length(wineind)-horizon+1):length(wineind)]
f <- frbe(ts(train, frequency=frequency(wineind)), h=horizon)
evalfrbe(f, test)

farules Create an instance of S3 class farules which represents a set of fuzzy association rules and their statistical characteristics.

Description
This function is a constructor that returns an instance of the farules S3 class. To search for fuzzy association rules, refer to the searchrules() function.

Usage
farules(rules, statistics)

Arguments
rules A list of character vectors, where each vector represents a rule and each value of the vector represents a predicate. The first value of the vector is assumed to be a rule’s consequent, the rest is a rule’s antecedent.

statistics A numeric matrix of various statistical characteristics of the rules. Each column of that matrix corresponds to some statistic (such as support, confidence, etc.) and each row corresponds to a rule in the list of rules.

Value
Returns an object of class farules.

Author(s)
Michal Burda
See Also

searchrules()

fcut

Transform data into a fsets S3 class using shapes derived from triangles or raised cosines

Description

This function creates a set of fuzzy attributes from crisp data. Factors, numeric vectors, matrix or data frame columns are transformed into a set of fuzzy attributes, i.e. columns with membership degrees. Unlike lcut(), for transformation is not used the linguistic linguistic approach, but partitioning using regular shapes of the fuzzy sets (such as triangle, raised cosine).

Usage

fcut(x, ...)

## Default S3 method:
fcut(x, ...)

## S3 method for class 'factor'
fcut(x, name = deparse(substitute(x)), ...)

## S3 method for class 'logical'
fcut(x, name = deparse(substitute(x)), ...)

## S3 method for class 'numeric'
fcut(
x,  
breaks,  
name = deparse(substitute(x)),  
type = c("triangle", "raisedcos"),  
merge = 1,  
parallel = FALSE,  
...  
)

## S3 method for class 'data.frame'
fcut(
x,  
breaks = NULL,  
name = NULL,  
type = c("triangle", "raisedcos"),  
merge = 1,  
parallel = FALSE,
## Arguments

- **x**
  
  Data to be transformed: a vector, matrix, or data frame. Non-numeric data are allowed.

- **...**
  
  Other parameters to some methods.

- **name**
  
  A name to be added as a suffix to the created fuzzy attribute names. This parameter can be used only if `x` is a vector. If `x` is a matrix or data frame, name should be NULL because the fuzzy attribute names are taken from column names of the argument `x`.

- **breaks**
  
  This argument determines the break-points of the positions of the fuzzy sets. It should be an ordered vector of numbers such that the `i`-th index specifies the beginning, `(i + 1)`-th the center, and `(i + 2)`-th the ending of the `i`-th fuzzy set. I.e. the minimum number of breaks-points is 3; `n - 2` elementary fuzzy sets would be created for `n` break-points.

  If considering an `i`-th fuzzy set (of `type='triangle'`), `x` values lower than `i`-th break (and greater than `(i + 2)`-th break) would result in zero membership degree, values equal to `(i + 1)`-th break would have membership degree equal 1 and values between them the appropriate membership degree between 0 and 1.

  The resulting fuzzy sets would be named after the original data by adding dot ("."), and a number `i` of fuzzy set.

  Unlike `base::cut()`, `x` values, that are lower or greater than the given break-points, will have all membership degrees equal to zero.

  For non-numeric data, this argument is ignored. For `x` being a numeric vector, it must be a vector of numeric values. For `x` being a numeric matrix or data frame, it must be a named list containing a numeric vector for each column - if not, the values are repeated for each column.

- **type**
  
  The type of fuzzy sets to create. Currently, 'triangle' or 'raisedcos' may be used. The type argument may be also a function with 3 or 4 arguments:

  - if `type` is a 4-argument function, it is assumed that it computes membership degrees from values of the first argument while considering the boundaries given by the next 3 arguments;
  
  - if `type` is a 3-argument function, it is assumed that it is a factory function similar to `triangular()` or `raisedcosinal()`, which, from given three boundaries, creates a function that computes membership degrees.

- **merge**
  
  This argument determines whether to derive additional fuzzy sets by merging the elementary fuzzy sets (whose position is determined with the `breaks` argument) into super-sets. The argument is ignored for non-numeric data in `x`.

  `merge` may contain any integer number from 1 to `length(breaks) - 2`. Value 1 means that the elementary fuzzy sets should be present in the output. Value 2 means that the two consecutive elementary fuzzy sets should be combined
by using the Lukasiewicz t-conorm, value 3 causes combining three consecutive elementary fuzzy sets etc.

The names of the derived (merged) fuzzy sets is derived from the names of the original elementary fuzzy sets by concatenating them with the "|" (pipe) separator.

**parallel**  
Whether the processing should be run in parallel or not. Parallelization is implemented using the `foreach::foreach()` function. The parallel environment must be set properly in advance, e.g. with the `doMC::registerDoMC()` function. Currently this argument is applied only if x is a matrix or data frame.

### Details

The aim of this function is to transform numeric data into a set of fuzzy attributes. The result is in the form of the object of class "fsets", i.e. a numeric matrix whose columns represent fuzzy sets (fuzzy attributes) with values being the membership degrees.

The function behaves differently to the type of input x.

If x is a factor or a logical vector (or other non-numeric data) then for each distinct value of an input, a fuzzy set is created, and data would be transformed into crisp membership degrees 0 or 1 only.

If x is a numeric vector then fuzzy sets are created accordingly to break-points specified in the breaks argument with 1st, 2nd and 3rd break-point specifying the first fuzzy set, 2nd, 3rd and 4th break-point specifying the second fuzzy set etc. The shape of the fuzzy set is determined by the type argument that may be equal either to a string 'triangle' or 'raisedcos' or it could be a function that computes the membership degrees for itself (see `triangular()` or `raisedcosinal()` functions for details). Additionally, super-sets of these elementary sets may be created by specifying the merge argument. Values of this argument specify how many consecutive fuzzy sets should be combined (by using the Lukasiewicz's t-conorm) to produce super-sets - see the description of merge above.

If a matrix (resp. data frame) is provided to this function instead of single vector, all columns are processed separately as described above and the result is combined with the `cbind.fsets()` function.

The function sets up properly the `vars()` and `specs()` properties of the result.

### Value

An object of class "fsets" is returned, which is a numeric matrix with columns representing the fuzzy attributes. Each source column of the x argument corresponds to multiple columns in the resulting matrix. Columns have names that indicate the name of the source as well as a index i of fuzzy set(s) – see the description of arguments breaks and merge above.

The resulting object would also have set the `vars()` and `specs()` properties with the former being created from original column names (if x is a matrix or data frame) or the name argument (if x is a numeric vector). The `specs()` incidency matrix would be created to reflect the superset-hood of the merged fuzzy sets.

### Author(s)

Michal Burda
See Also

lcut(), farules(), pbld(), vars(), specs(), cbind.fsets()

Examples

# fcut on non-numeric data
ff <- factor(substring("statistics", 1:10, 1:10), levels = letters)
fcut(ff)

# transform a single vector into a single fuzzy set
x <- runif(10)
fcut(x, breaks=c(0, 0.5, 1), name="age")

# transform single vector into a partition of the interval 0-1
# (the boundary triangles are right-angled)
fcut(x, breaks=c(0, 0, 0.5, 1, 1), name='age')

# also create supersets
fcut(x, breaks=c(0, 0, 0.5, 1, 1), name='age', merge=c(1, 2))

# transform all columns of a data frame
# with different breakpoints
data <- CO2[, c('conc', 'uptake')]
fcut(data, breaks=list(conc=c(95, 95, 350, 1000, 1000),
                      uptake=c(7, 7, 28.3, 46, 46)))

# using a custom 3-argument function (a function factory):
f <- function(a, b, c) {
  return(function(x) ifelse(a <= x & x <= b, 1, 0))
}
f(x, breaks=c(0, 0.5, 1), name='age', type=f)

# using a custom 4-argument function:
f <- function(x, a, b, c) {
  return(ifelse(a <= x & x <= b, 1, 0))
}
f(x, breaks=c(0, 0.5, 1), name='age', type=f)

fire

Evaluate rules and obtain truth-degrees

Description

Given truth degrees of predicates, compute the truth value of given list of rules.
Usage

fire(
  x,
  rules,
  tnorm = c("goedel", "goguen", "lukasiewicz"),
  onlyAnte = TRUE,
  parallel = FALSE
)

Arguments

x  Truth degrees of predicates. x could be either a numeric matrix or a numeric vector. If vector is given then each named element represents a truth value of a predicate. If matrix is given then each row of the matrix is evaluated sequentially as a vector. The values must be in the interval [0, 1].

rules Either an object of S3 class farules() or a list of character vectors where each vector is a rule in a conjunctive form. Elements of these character vectors (i.e., predicate names) must correspond to the x’s names (of elements resp. columns if x is a vector resp. matrix).

tnorm A character string representing a triangular norm to be used (either "goedel", "goguen", or "lukasiewicz") or an arbitrary function that performs element-wise computation on arbitrary number of vector parameters similarly as e.g. pgoedel.tnorm(), pgoguen.tnorm() or plukas.tnorm().

onlyAnte TRUE is useful if rules store both the antecedent and consequent and if only the antecedent-part of a rule should be included into the evaluated conjunction. Antecedent-part of a rule are all predicates in the vector starting from the 2nd position. TRUE value in this parameter causes the first element of each rule to be ignored.

If FALSE, all predicates in a rule will be included in the conjunction.

parallel Deprecated parameter. Computation is done sequentially.

Details

The aim of this function is to compute the truth value of each rule in a rules list by assigning truth values to rule’s predicates given by data x.

x is a numeric vector or numeric matrix of truth values of predicates. If x is vector then names(x) must correspond to the predicate names in rules. If x is a matrix then each column should represent a predicate and thus colnames(x) must correspond to predicate names in rules. Values of x are interpreted as truth values, i.e., they must be from the interval [0, 1]. If matrix is given, the resulting truth values are computed row-wisely.

rules may be a list of character vectors or an instance of the S3 class farules(). The character vectors in the rules list represent formulae in conjunctive form. If onlyAnte=FALSE, fire() treats the rule as a conjunction of all predicates, i.e., a conjunction of all predicates is computed. If onlyAnte=TRUE, the first element of each rule is removed prior evaluation, i.e., a conjunction of all predicates except the first are computed: this is useful if rules is a farules() object, since farules() objects save a rule’s consequent as the first element (see also antecedents() and consequents() functions).
The type of conjunction to be computed can be specified with the tnorm parameter.

Value

If \( x \) is a matrix then the result of this function is a list of numeric vectors with truth values of each rule, i.e., each element of the resulting list corresponds to a rule and each value of the vector in the resulting list corresponds to a row of the original data matrix \( x \).

\( x \) as a vector is treated as a single-row matrix.

Author(s)

Michal Burda

See Also

aggregateConsequents(), defuzz(), perceive(), pbld(), fcut(), lcut(), farules()

Examples

# fire whole rules on a vector
\[
\text{x} \leftarrow 1:10 / 10
\]
\[
\text{names(x)} \leftarrow \text{letters}[1:10]
\]
\[
\text{rules} \leftarrow \text{list(c('a', 'c', 'e'), c('b'), c('d', 'a'), c('c', 'a', 'b'))}
\]
\[
\text{fire(x, rules, tnorm='goguen', onlyAnte=FALSE)}
\]

# fire antecedents of the rules on a matrix
\[
\text{x} \leftarrow \text{matrix}(1:20 / 20, nrow=2)
\]
\[
\text{colnames(x)} \leftarrow \text{letters}[1:10]
\]
\[
\text{rules} \leftarrow \text{list(c('a', 'c', 'e'), c('b'), c('d', 'a'), c('c', 'a', 'b'))}
\]
\[
\text{fire(x, rules, tnorm='goedel', onlyAnte=TRUE)}
\]

# the former command should be equal to
\[
\text{fire(x, antecedents(rules), tnorm='goedel', onlyAnte=FALSE)}
\]
Usage

frbe(d, h = 10)

Arguments

d A source time-series in the ts time-series format. Note that the frequency of the time-series must be set properly.

h A forecasting horizon, i.e. the number of values to forecast.

Details

This function computes the fuzzy rule-based ensemble of time-series forecasts. The evaluation comprises of the following steps:

1. Several features are extracted from the given time-series d:
   - length of the time-series;
   - strength of trend;
   - strength of seasonality;
   - skewness;
   - kurtosis;
   - variation coefficient;
   - stationarity;
   - frequency. These features are used later to infer weights of the forecasting methods.

2. Several forecasting methods are applied on the given time-series d to obtain forecasts. Actually, the following methods are used:
   - ARIMA - by calling forecast::auto.arima();
   - Exponential Smoothing - by calling forecast::ets();
   - Random Walk with Drift - by calling forecast::rwf();
   - Theta - by calling forecast::thetaf().

3. Computed features are input to the fuzzy rule-based inference mechanism which yields into weights of the forecasting methods. The fuzzy rule base is hardwired in this package and it was obtained by performing data mining with the use of the farules() function.

4. A weighted sum of forecasts is computed and returned as a result.

Value

Result is a list of class frbe with the following elements:

- features - a data frame with computed features of the given time-series;
- forecasts - a data frame with forecasts to be ensembled;
- weights - weights of the forecasting methods as inferred from the features and the hard-wired fuzzy rule base;
- mean - the resulting ensembled forecast (computed as a weighted sum of forecasts).
Author(s)
Michal Burda

References
Štěpnička, M., Burda, M., Štěpničková, L. Fuzzy Rule Base Ensemble Generated from Data by Linguistic Associations Mining. FUZZY SET SYST. 2015.

See Also
evalfrbe()

Examples

# prepare data (from the forecast package)
library(forecast)
horizon <- 10
train <- wineind[-1 * (length(wineind)-horizon+1):length(wineind)]
test <- wineind[(length(wineind)-horizon+1):length(wineind)]

# perform FRBE
f <- frbe(ts(train, frequency=frequency(wineind)), h=horizon)

# evaluate FRBE forecasts
evalfrbe(f, test)

# display forecast results
f$mean

---

fsets

S3 class representing a set of fuzzy sets on the fixed universe

Description
The aim of the fsets S3 class is to store several fuzzy sets in the form of numeric matrix where columns represent fuzzy sets, rows are elements from the universe, and therefore a value of i-th row and j-th column is a membership degree of i-th element of the universe to j-th fuzzy set. The fsets object also stores the information about the origin of the fuzzy sets as well as a relation of specificity among them.

Usage

fsets(
x,
  vars = rep(deparse(substitute(x)), ncol(x)),
  specs = matrix(0, nrow = ncol(x), ncol = ncol(x))
)
vars(f)
vars(f) <- value
specs(f)
specs(f) <- value

Arguments

x
A matrix of membership degrees. Columns of the matrix represent fuzzy sets, colnames are names of the fuzzy sets (and must not be NULL). Rows of the matrix represent elements of the universe.

vars
A character vector that must correspond to the columns of x. It is a vector of names of original variables that the fuzzy sets were created from. In other words, the vars vector should contain the same value for each x's column that corresponds to the same variable. Names of the vars vector are ignored. For instance, an fcut() function can transform a single numeric vector into several different fuzzy sets. To indicate that all of them in fact describe the same original variable, the same name is stored on appropriate positions of the vars vector.

specs
A square numeric matrix containing values from \{0, 1\}. It is a specificity matrix, for which both rows and columns correspond to x's columns and where specs[i][j] == 1 if and only if i-th fuzzy set (i.e. x[,i]) is more specific (is a subset or equal to) than j-th fuzzy set (i.e. x[,j]).

f
An instance of S3 class fsets.

value
Attribute values to be set to the object.

Details

The fsets() function is a constructor of an object of type fsets. Each object stores two attributes: vars and specs. The functions vars() and specs() can be used to access these attributes.

It is assumed that the fuzzy sets are derived from some raw variables, e.g. numeric vectors or factors. vars attribute is a character vector of names of raw variables with size equal to the number of fuzzy sets in fsets object. It is then assumed that two fuzzy sets with the same name in vars() attribute are derived from the same variable.

specs attribute gives a square numeric matrix of size equal to the number of fuzzy sets in fsets. specs[i][j] == 1 if and only if the i-th fuzzy set is more specific than j-th fuzzy set. Specificity of fuzzy sets means the nestedness of fuzzy set: for instance, very small is more specific than small; similarly, extremely big is more specific than very big; on the other hand, very big and extremely small are incomparable. A necessary condition for specificity is subsethood.

Value

fsets() returns an object of S3 class fsets. vars() returns a vector of original variable names of the fsets object. specs returns the specificity matrix.
Author(s)
Michal Burda

See Also
fcut(), lcut(), is.specific()

Examples

```r
# create a matrix of random membership degrees
m <- matrix(runif(30), ncol=5)
colnames(m) <- c('a1', 'a2', 'a12', 'b1', 'b2')

# create vars - first three (a1, a2, a3) and next two (b1, b2)
# fuzzy sets originate from the same variable
v <- c('a', 'a', 'a', 'b', 'b')
names(v) <- colnames(m)

# create specificity matrix - a1 and a2 are more specific than a12,
# the rest is incomparable
s <- matrix(c(0, 0, 1, 0, 0,
             0, 0, 1, 0, 0,
             0, 0, 0, 0, 0,
             0, 0, 0, 0, 0,
             0, 0, 0, 0, 0), byrow=TRUE, ncol=5)
colnames(s) <- colnames(m)
rownames(s) <- colnames(m)

# create a valid instance of the fsets class
o <- fsets(m, v, s)
```

---

**hedge**

**Linguistic hedges**

**Description**

Returns a function that realizes linguistic hedging - i.e. transformation of linguistic horizon (see horizon()) into a linguistic expression.

**Usage**

```r
hedge(
  type = c("ex", "si", "ve", "ty", ",", "ml", "ro", "qr", "vr"),
  hedgeParams = defaultHedgeParams
)
```
Arguments

- **type**  The type of the required linguistic hedge
- **hedgeParams**  Parameters that determine the shape of the hedges

Details

hedge() returns a function that realizes the selected linguistic hedge on its parameter:

- **ex**: extremely,
- **si**: significantly,
- **ve**: very,
- **ty**: typically,
- **~**: empty hedge (no hedging),
- **ml**: more or less,
- **ro**: roughly,
- **qr**: quite roughly,
- **vr**: very roughly.

This function is quite low-level. Perhaps a more convenient way to create linguistic expressions is to use the `lingexpr()` function.

Value

Returns a function with a single argument, which has to be a numeric vector.

Author(s)

Michal Burda

See Also

`horizon()`, `lingexpr()`, `fcut()`, `lcut()`, `ctx()`

Examples

```r
a <- horizon(ctx3(), 'sm')
plot(a)
h <- hedge('ve')
plot(h)
verySmall <- function(x) h(a(x))
plot(verySmall)

# the last plot should be equal to:
plot(lingexpr(ctx3(), atomic='sm', hedge='ve'))
```
Create a function that computes linguistic horizons

Description

Based on given context and atomic expression, this function returns a function that computes a linguistic horizon, i.e. a triangular function representing basic limits of what humans treat as "small", "medium", "big" etc. within given context. Linguistic horizon stands as a base for creation of linguistic expressions. A linguistic expression is created by applying a `hedge()` on horizon. (Atomic linguistic expression is created from horizon by applying an empty (\-\) hedge).

Usage

```r
horizon(
  context,
  atomic = c("sm", "me", "bi", "lm", "um", "ze", "neg.sm", "neg.me", "neg.bi", "neg.lm", "neg.um")
)
```

Arguments

- `context`: A context of linguistic expressions (see `ctx3()`, `ctx5()`, `ctx3bilat()` or `ctx5bilat()`)
- `atomic`: An atomic expression whose horizon we would like to obtain

Details

The values of the `atomic` parameter have the following meaning (in ascending order):

- `neg.bi`: big negative (far from zero)
- `neg.um`: upper medium negative (between medium negative and big negative)
- `neg.me`: medium negative
- `neg.lm`: lower medium negative (between medium negative and small negative)
- `neg.sm`: small negative (close to zero)
- `ze`: zero
- `sm`: small
- `lm`: lower medium
- `me`: medium
- `um`: upper medium
- `bi`: big

Based on the context type, the following atomic expressions are allowed:

- `ctx3()` (trichotomy): small, medium, big;
- `ctx5()` (pentachotomy): small, lower medium, medium, upper medium, big;
is.farules

Test whether x inherits from the S3 farules class.

Description

Test whether x inherits from the S3 farules class.

Usage

is.farules(x)

Arguments

x       An object being tested.
is.frbe

Value

TRUE if x is a valid farules() object and FALSE otherwise.

Author(s)

Michal Burda

See Also

farules

is.frbe  Test whether x is a valid object of the S3 frbe class

Description

Test whether x has a valid format for the objects of the S3 frbe class.

Usage

is.frbe(x)

Arguments

x  An object being tested.

Details

This function tests wheter x inherits from frbe i.e. whether it is a list with the following elements: forecasts data frame, features data frame, weights vector, and mean vector. Instances of the S3 class frbe are usually created by the frbe() function.

Value

TRUE if x is a valid frbe object and FALSE otherwise.

Author(s)

Michal Burda

References

Štěpnička, M., Burda, M., Štěpničková, L. Fuzzy Rule Base Ensemble Generated from Data by Linguistic Associations Mining. FUZZY SET SYST. 2015.

See Also

frbe()
is.fsets

Test whether \( x \) is a valid object of the S3 \texttt{fsets} class

Description
This function tests whether \( x \) inherits from S3 \texttt{fsets} class.

Usage
\begin{verbatim}
is.fsets(x)
\end{verbatim}

Arguments
\begin{itemize}
\item \( x \) An object being tested.
\end{itemize}

Value
\begin{verbatim}
TRUE if \( x \) is a valid \texttt{fsets} object and FALSE otherwise.
\end{verbatim}

Author(s)
Michal Burda

See Also
\begin{verbatim}
fsets()
\end{verbatim}

is.specific

Determine whether the first set \( x \) of predicates is more specific (or equal) than \( y \) with respect to \texttt{vars} and \texttt{specs}.

Description
The function takes two character vectors of predicates and determines whether \( x \) is more specific (or equal w.r.t. the specificity) than \( y \). The specificity relation is fully determined with the values of the \texttt{vars()} vector and the \texttt{specs()} incidence matrix that is encapsulated in the given \texttt{fsets} object.

Usage
\begin{verbatim}
is.specific(x, y, fsets, vars = NULL, specs = NULL)
\end{verbatim}
is.specific

Arguments

- **x**: The first character vector of predicates.
- **y**: The second character vector of predicates.
- **fsets**: A valid instance of the `fsets()` class such that all values in `x` and `y` can be found in `colnames(fsets)`.
- **vars**: Deprecated parameter must be NULL.
- **specs**: Deprecated parameter must be NULL.

Details

Let \( x_i \) and \( y_j \) represent some predicates of vectors \( x \) and \( y \), respectively. Function assumes that each vector \( x \) and \( y \) does not contain two or more predicates with the same value of `vars()`.

This function returns TRUE iff all of the following conditions hold:

- for any \( y_j \) there exists \( x_i \) such that \( \text{vars}[y_j] = \text{vars}[x_i] \);
- for any \( x_i \) there either does not exist \( y_j \) such that \( \text{vars}[x_i] = \text{vars}[y_j] \), or \( x_i = y_j \), or \( \text{specs}[x_i, y_j] = 1 \).

Value

TRUE or FALSE (see description).

Author(s)

Michal Burda

See Also

`perceive()`, `pblld()`, `fsets()`, `vars()`, `specs()`

Examples

```r
# prepare fsets object
v <- c(rep('a', 3), rep('b', 3), rep('c', 3), rep('d', 3))
s <- matrix(c(0,1,0, 0,0,0, 0,0,0, 0,0,0,
             0,0,0, 0,0,0, 0,0,0, 0,0,0,
             0,0,0, 0,0,0, 0,0,0, 0,0,0,
             0,0,0, 0,0,0, 0,0,0, 0,0,0,
             0,0,0, 0,1,0, 0,0,0, 0,0,0,
             0,0,0, 0,0,0, 0,0,0, 0,0,0,
             0,0,0, 0,0,0, 0,0,0, 0,0,0,
             0,0,0, 0,0,0, 0,0,0, 0,0,0,
             0,0,0, 0,0,0, 0,0,0, 0,0,0),
             nrow = 10, byrow = TRUE)
```
byrow=TRUE,
ncol=12)
m <- matrix(0, nrow=1, ncol=12)
colnames(m) <- paste(rep(c('VeSm', 'Sm', 'Bi'), times=4),
rep(c('a', 'b', 'c', 'd'), each=3),
sep='.'
)f <- fsets(m, v, s)

# returns TRUE
is.specific(c('VeSm.a', 'Bi.c'),
c('VeSm.a', 'Bi.c'),
f)

# returns TRUE (x and y swapped return FALSE)
is.specific(c('VeSm.a', 'Bi.c', 'Sm.d'),
c('Sm.a', 'Bi.c', 'Sm.d'),
f)

# returns TRUE (x and y swapped return FALSE)
is.specific(c('VeSm.a', 'Bi.c', 'Sm.d'),
c('VeSm.a', 'Bi.c'),
f)

# returns TRUE (x and y swapped return FALSE)
is.specific(c('VeSm.a', 'Bi.c', 'Sm.d'),
character(),
f)

# returns FALSE
is.specific(c('Sm.a'), c('Bi.c'), f)

# returns FALSE
is.specific(c('VeSm.a', 'Sm.c'),
c('Sm.a', 'Bi.c'),
f)

lcut

Transform data into a fsets S3 class of linguistic fuzzy attributes

Description

This function creates a set of linguistic fuzzy attributes from crisp data. Numeric vectors, matrix
or data frame columns are transformed into a set of fuzzy attributes, i.e. columns with membership
degrees. Factors and other data types are transformed to fuzzy attributes by calling the fcut() function.

Usage

lcut(x, ...)
## Default S3 method:
lcut(x, ...)

## S3 method for class 'factor'
lcut(x, name = deparse(substitute(x)), ...)

## S3 method for class 'logical'
lcut(x, name = deparse(substitute(x)), ...)

## S3 method for class 'numeric'
lcut(
  x,
  context = minmax,
  atomic = c("sm", "me", "bi", "lm", "um", "ze", "neg.sm", "neg.me", "neg.bi", "neg.lm", "neg.um"),
  hedges = c("ex", "si", "ve", "ty", ",", "ml", "ro", "qr", "vr"),
  name = NULL,
  hedgesParams = defaultHedgeParams,
  ...
)

## S3 method for class 'data.frame'
lcut(
  x,
  context = minmax,
  atomic = c("sm", "me", "bi", "lm", "um", "ze", "neg.sm", "neg.me", "neg.bi", "neg.lm", "neg.um"),
  hedges = c("ex", "si", "ve", "ty", ",", "ml", "ro", "qr", "vr"),
  ...
)

## S3 method for class 'matrix'
lcut(x, ...)

### Arguments

**x**
Data to be transformed: if it is a numeric vector, matrix, or data frame, then
the creation of linguistic fuzzy attributes takes place. For other data types the
fcut() function is called implicitly.

**...**
Other parameters to some methods.

**name**
A name to be added as a suffix to the created fuzzy attribute names. This pa-
rameter can be used only if x is a numeric vector. If x is a matrix or data frame,
name should be NULL because the fuzzy attribute names are taken from column
names of parameter x. The name is also used as a value for the vars attribute of
the resulting fsets() instance.

**context**
A definition of context of a numeric attribute. It must be an instance of an S3
class ctx3(), ctx5(), ctx3bilat() or ctx5bilat().
If \( x \) is a matrix or data frame then \texttt{context} should be a named list of contexts for each \( x \)'s column.

\begin{itemize}
  \item \texttt{atomic} \hfill A vector of atomic linguistic expressions to be used for creation of fuzzy attributes.
  \item \texttt{hedges} \hfill A vector of linguistic hedges to be used for creation of fuzzy attributes.
  \item \texttt{hedgeParams} \hfill Parameters that determine the shape of the hedges
\end{itemize}

\section*{Details}

The aim of this function is to transform numeric data into a set of fuzzy attributes. The resulting fuzzy attributes have direct linguistic interpretation. This is a unique variant of fuzzification that is suitable for the inference mechanism based on Perception-based Linguistic Description (PbLD) – see \texttt{pbld()}.

A numeric vector is transformed into a set of fuzzy attributes accordingly to the following scheme: 
\[ < \text{hedge} > < \text{atomicexpression} > \]
where \( < \text{atomicexpression} > \) is an atomic linguistic expression, a value from the following possibilities (note that the allowance of atomic expressions is influenced with \texttt{context} being used - see \texttt{ctx} for details):

- \texttt{neg.bi}: big negative (far from zero)
- \texttt{neg.um}: upper medium negative (between medium negative and big negative)
- \texttt{neg.me}: medium negative
- \texttt{neg.lm}: lower medium negative (between medium negative and small negative)
- \texttt{neg.sm}: small negative (close to zero)
- \texttt{ze}: zero
- \texttt{sm}: small
- \texttt{lm}: lower medium
- \texttt{me}: medium
- \texttt{um}: upper medium
- \texttt{bi}: big A \( < \text{hedge} > \) is a modifier that further concretizes the atomic expression (note that not each combination of hedge and atomic expression is allowed - see \texttt{allowed.lingexpr} for more details):
  - \texttt{ex}: extremely,
  - \texttt{si}: significantly,
  - \texttt{ve}: very,
  - \texttt{ty}: typically,
  - \texttt{~}: empty hedge (no hedging),
  - \texttt{m1}: more or less,
  - \texttt{ro}: roughly,
  - \texttt{qr}: quite roughly,
  - \texttt{vr}: very roughly.
Accordingly to the theory developed by Novak (2008), not every hedge is suitable with each atomic 
expression (see the description of the hedges argument). The hedges to be used can be selected 
with the hedges argument. Function takes care of not to use hedge together with an un-applicable 
atomic expression by itself.

Obviously, distinct data have different meaning of what is "small", "medium", or "big" etc. Therefore, a context has to be set that specifies sensible values for these linguistic expressions.

If a matrix (resp. data frame) is provided to this function instead of a single vector, all columns are processed the same way.

The function also sets up properly the vars() and specs() properties of the result.

Value
An object of S3 class fsets is returned, which is a numeric matrix with columns representing the fuzzy attributes. Each source column of the x argument corresponds to multiple columns in the resulting matrix. Columns will have names derived from used hedges, atomic expression, and name specified as the optional parameter.

The resulting object would also have set the vars() and specs() properties with the former being created from original column names (if x is a matrix or data frame) or the name argument (if x is a numeric vector). The specs() incidency matrix would be created to reflect the following order of the hedges: "ex" < "si" < "ve" < "-" < "ml" < "ro" < "qr" < "vr" and "ty" < "" < "ml" < "ro" < "qr" < "vr". Fuzzy attributes created from the same source numeric vector (or column) would be ordered that way, with other fuzzy attributes (from the other source) being incomparable.

Author(s)
Michal Burda

References

See Also
fcut(), fsets(), vars(), specs()

Examples

# transform a single vector
x <- runif(10)
lcut(x, name='age')

# transform single vector with a custom context
lcut(x, context=ctx5(0, 0.2, 0.5, 0.7, 1), name='age')

# transform all columns of a data frame
# and do not use any hedges
data <- CO2[, c('conc', 'uptake')]
lcut(data)

# definition of custom contexts for different columns
# of a data frame while selecting only "ve" and "ro" hedges.
lcut(data,
    context=list(conc=minmax,
                 uptake=ctx3(0, 25, 50)),
    hedges=c('ve', 'ro'))

# lcut on non-numeric data is the same as fcut()
ff <- factor(substring("statistics", 1:10, 1:10), levels = letters)
lcut(ff)

---

lcut3

*Deprecated functions for transformation of data into a fsets S3 class of linguistic fuzzy attributes*

### Description

These functions are deprecated. Please use `lcut()` or `fcut()` instead.

### Usage

```r
lcut3(x, ...)
lcut5(x, ...)

## Default S3 method:
lcut3(x, ...)
lcut5(...)

## S3 method for class 'numeric'
lcut3(  
x,  
    context = NULL,  
    defaultCenter = 0.5,  
    atomic = c("sm", "me", "bi"),  
    hedges = c("ex", "si", "ve", "ml", "ro", "qr", "vr"),  
    name = NULL,  
    parallel = FALSE,  
    ...
)
```
## S3 method for class 'numeric'
lcut5(  
  x,  
  context = NULL,  
  defaultCenter = 0.5,  
  atomic = c("sm", "lm", "me", "um", "bi"),  
  hedges = c("ex", "ve", "ml", "ro", "ty"),  
  name = NULL,  
  parallel = FALSE,  
  ...
)

## S3 method for class 'data.frame'
lcut3(x, context = NULL, name = NULL, parallel = FALSE, ...)

## S3 method for class 'data.frame'
lcut5(x, context = NULL, name = NULL, parallel = FALSE, ...)

## S3 method for class 'matrix'
lcut3(x, ...)

## S3 method for class 'matrix'
lcut5(x, ...)

**Arguments**

- **x**
  - Data to be transformed: if it is a numeric vector, matrix, or data frame, then the creation of linguistic fuzzy attributes takes place. For other data types the `fcut()` function is called implicitly.

- **...**
  - Other parameters to some methods.

- **context**
  - A definition of context of a numeric attribute. Context determines how people understand the notions "small", "medium", or "big" with respect to that attribute. If `x` is a numeric vector then context should be a vector of 3 numbers: typical small, medium, and big value. If the context is set to NULL, these values are taken directly from `x` as follows:
    - small: \(= \min(x)\);
    - medium: \(= (\max(x) - \min(x)) \times \text{defaultCenter} + \min(x)\);
    - big: \(= \max(x)\). If `x` is a matrix or data frame then context should be a named list of contexts for each `x`'s column. If some context is omitted, it will be determined directly from data as explained above.
  - Regardless of the value of the `atomic` argument, all 3 numbers of the context must be provided everytime.

- **defaultCenter**
  - A value used to determine a typical "medium" value from data (see `context` above). If `context` is not specified then typical "medium" is determined as
    \[(\max(x) - \min(x)) \times \text{defaultCenter} + \min(x)\].
Default value of defaultCenter is 0.5, however, some literature specifies 0.42 as another sensible value with proper linguistic interpretation.

- **atomic**: A vector of atomic linguistic expressions to be used for creation of fuzzy attributes.
- **hedges**: A vector of linguistic hedges to be used for creation of fuzzy attributes.
- **name**: A name to be added as a suffix to the created fuzzy attribute names. This parameter can be used only if `x` is a numeric vector. If `x` is a matrix or data frame, name should be NULL because the fuzzy attribute names are taken from column names of parameter `x`. The name is also used as a value for the vars attribute of the resulting fsets() instance.
- **parallel**: Whether the processing should be run in parallel or not. Parallelization is implemented using the foreach::foreach() function. The parallel environment must be set properly in advance, e.g. with the doMC::registerDoMC() function.

### Value

An object of S3 class fsets is returned, which is a numeric matrix with columns representing the fuzzy attributes. Each source column of the `x` argument corresponds to multiple columns in the resulting matrix. Columns will have names derived from used `hedges`, atomic expression, and `name` specified as the optional parameter.

The resulting object would also have set the `vars()` and `specs()` properties with the former being created from original column names (if `x` is a matrix or data frame) or the `name` argument (if `x` is a numeric vector). The `specs()` incidence matrix would be created to reflect the following order of the hedges: "ex" < "si" < "ve" < "-" < "ml" < "ro" < "qr" < "vr" and "ty" < "n" < "ml" < "ro" < "qr" < "vr". Fuzzy attributes created from the same source numeric vector (or column) would be ordered that way, with other fuzzy attributes (from the other source) being incomparable.

### Author(s)

Michal Burda

---

**Description**

lfl
Description

A linguistic expression represents vague human terms such as "very small", "extremely big" etc. Such notions are always reasoned within a given context. `lingexpr` returns a function that models a selected linguistic expression. Accordingly to the given context, atomic expression (such as "small", "big") and a linguistic hedge (such as very, extremely), the returned function transforms numeric values into degrees (from [0, 1] interval), at which the values correspond to the expression.

Usage

```r
lingexpr(
  context,
  atomic = c("sm", "me", "bi", "lm", "um", "ze", "neg.sm", "neg.me", "neg.bi", "neg.lm", "neg.um"),
  hedge = c("ex", "si", "ve", "ty", "-", "ml", "ro", "qr", "vr"),
  hedgeParams = defaultHedgeParams
)
```

allowed.lingexpr

Arguments

- **context**: A context of linguistic expressions (see `ctx3()`, `ctx5()`, `ctx3bilat()` or `ctx5bilat()`)
- **atomic**: An atomic expression whose horizon we would like to obtain
- **hedge**: The type of the required linguistic hedge ("-" for no hedging)
- **hedgeParams**: Parameters that determine the shape of the hedges

Format

An object of class `matrix` with 9 rows and 11 columns.

Details

Based on the context type, the following atomic expressions are allowed:

- **ctx3()** (trichotomy): small, medium, big;
- **ctx5()** (pentachotomy): small, lower medium, medium, upper medium, big;
- **ctx3bilat()** (bilateral trichotomy): negative big, negative medium, negative small, zero, small, medium, big;
- **ctx5bilat()** (bilateral pentachotomy): negative big, negative medium, negative small, zero, small, medium, big.

The values of the atomic parameter have the following meaning (in ascending order):
• neg.bi: big negative (far from zero)
• neg.um: upper medium negative (between medium negative and big negative)
• neg.me: medium negative
• neg.lm: lower medium negative (between medium negative and small negative)
• neg.sm: small negative (close to zero)
• ze: zero
• sm: small
• lm: lower medium
• me: medium
• um: upper medium
• bi: big

hedge parameter has the following meaning:

• ex: extremely,
• si: significantly,
• ve: very,
• ty: typically,
• -: empty hedge,
• ml: more or less,
• ro: roughly,
• qr: quite roughly,
• vr: very roughly.

Accordingly to the theory of linguistic expressions by Novak, not every hedge is applicable to each atomic expression. The combinations of allowed pairs can be found in allowed.lingexpr. Trying to create forbidden combination results in error.

Value

Returns a function with a single argument, which has to be a numeric vector.

Author(s)

Michal Burda

See Also

horizon(), hedge(), fcut(), lcwt(), ctx()
Examples

```r
small <- lingexpr(ctx3(0, 0.5, 1), atomic='sm', hedge='-',
                 values=c(0, 0.8), a=1)
small(0)  # 1
small(0.8) # 0
plot(small)

verySmall <- lingexpr(ctx3(0, 0.5, 1), atomic='sm', hedge='ve',
                      values=c(0, 0.8), a=1)
plot(verySmall)
```

---

**mase**  
**Compute Mean Absolute Scaled Error (MASE)**

**Description**

MASE is computed as \( \frac{\sum|\text{validation} - \text{forecast}|}{\sum|\text{validation}_{-1} - \text{validation}_{n}|}/(n/(n-1)) \).

**Usage**

```r
mase(forecast, validation)
```

**Arguments**

- `forecast` A numeric vector of forecasted values
- `validation` A numeric vector of actual (real) values

**Value**

A Mean Absolute Scaled Error (MASE)

**Author(s)**

Michal Burda

**See Also**

`rmse()`, `smape()`, `frbe()`
Creating linguistic context directly from values

Description

This function creates a context (i.e. an instance of S3 class `ctx3()`, `ctx3bilat()`, `ctx5()`, or `ctx5bilat()`) based on values of the numeric vector `x`. In default, the context is based on minimum and maximum value of `x` in the following way:

- `ctx3`, `ctx5`: low = minimum, high = maximum value of `x`;
- `ctx3bilat`, `ctx5bilat`: negMax = minimum, max = maximum value of `x`, origin = mean of minimum and maximum.

Usage

```
minmax(x, type = c("ctx3", "ctx5", "ctx3bilat", "ctx5bilat"), ...)
```

Arguments

- `x` A numeric vector to compute the context from
- `type` A type of the context to be returned. Must be one of: `ctx3`, `ctx5`, `ctx3bilat` or `ctx5bilat`
- `...` other parameters to be passed to the appropriate constructor (`ctx3()`, `ctx3bilat()`, `ctx5()`, and `ctx5bilat()`) that is called internally. These values overwrite the defaults computed by `minmax` – see the examples.

Details

Other values are computed accordingly to defaults as defined in the constructors `ctx3()`, `ctx3bilat()`, `ctx5()`, and `ctx5bilat()`.

Examples

```
minmax(0:100)       # returns ctx3: 0, 50, 100
minmax(0:100, high=80) # returns ctx3: 0, 40, 80
minmax(0:100, relCenter=0.4) # returns ctx3: 0, 40, 100
minmax(0:100, type='ctx5')  # returns ctx5: 0, 25, 50, 75, 100
```
Callback-based Multiplication of Matrices

Description
Perform a custom multiplication of the matrices x and y by using the callback function f.

Usage
mult(x, y, f, ...)

Arguments
x
A first matrix. The number of columns must match with the number of rows of the y matrix.
y
A second matrix. The number of rows must match with the number of columns of the x matrix.
f
A function to be applied to the matrices in order to compute the multiplication. It must accept at least two arguments.
...
Additional arguments that are passed to the function f.

Details
For a matrix x of size (u, v) and a matrix y of size (v, w), mult calls the function f uw-times to create a resulting matrix of size (u, w). Each (i, j)-th element of the resulting matrix is obtained from a call of the function f with x's i-th row and y's j-th column passed as its arguments.

Value
A matrix with v rows and w columns, where v is the number of rows of x and w is the number of columns of y.

Author(s)
Michal Burda

See Also
compose()

Examples
x <- matrix(runif(24, -100, 100), ncol=6)
y <- matrix(runif(18, -100, 100), nrow=6)
mult(x, y, function(xx, yy) sum(xx * yy)) # the same as "x %*% y"
pbld Perform a Perception-based Logical Deduction (PbLD) with given rule-base on given dataset

Description
Take a set of rules (a rule-base) and perform a Perception-based Logical Deduction (PbLD) on each row of a given fsets() object.

Usage
pbld(
  x,  
  rules, 
  partition, 
  values, 
  type = c("global", "local"), 
  parallel = FALSE 
)

Arguments
x Input to the inference. It should be an object of class fsets() (e.g. created by using the fcut() or lcut() functions). It is basically a matrix with columns representing fuzzy sets. Each row represents a single case of inference. Columns should be named after predicates in rules' antecedents.

rules A rule-base (a.k.a. linguistic description) either in the form of the farules() object or as a list of character vectors where each element is a fuzzy set name (a predicate) and thus each such vector forms a rule.

partition A fsets() object with columns that are consequents in rules. These membership degrees must correspond to values.

values Crisp values that correspond to rows of membership degrees in the partition matrix. Function assumes that the values are sorted in the ascending order.

type The type of inference to use. It can be either "local" or "global" (default).

parallel Whether the processing should be run in parallel or not. Parallelization is implemented using the foreach::foreach() package. The parallel environment must be set properly in advance, e.g. with the doMC::registerDoMC() function.

Details
Perform a Perception-based Logical Deduction (PbLD) with given rule-base rules on each row of input x. Columns of x are truth values of predicates that appear in the antecedent part of rules, partition together with values determine the shape of predicates in consequents: each element in values corresponds to a row of membership degrees in partition.
Value

A vector of inferred defuzzified values. The number of resulting values corresponds to the number of rows of the x argument.

Author(s)

Michal Burda

References


See Also

lcut(), searchrules(), fire(), aggregateConsequents(), defuzz()

Examples

```r
# --- TRAINING PART ---
# custom context of the RHS variable
takeContext <- ctx3(7, 28.3, 46)

t # convert data into fuzzy sets
d <- lcut(CO2, context=list(uptake=uptakeContext))

# split data into the training and testing set
testingIndices <- 1:5
trainingIndices <- setdiff(seq_len(nrow(CO2)), testingIndices)
training <- d[trainingIndices, ]
testing <- d[testingIndices, ]

# search for rules
r <- searchrules(training, lhs=1:38, rhs=39:58, minConfidence=0.5)

# --- TESTING PART ---
# prepare values and partition
v <- seq(uptakeContext[1], uptakeContext[3], length.out=1000)
p <- lcut(v, name='uptake', context=uptakeContext)

# do the inference
pbld(testing, r, p, v)
```
peceive

From a set of rules, remove each rule for which another rule exists that is more specific.

Description

Examine rules in a list and remove all of them for whose other more specific rules are present in the list. The specificity is determined by calling the `is.specific()` function. This operation is a part of the `pbld()` inference mechanism.

Usage

```r
perceive(
  rules,
  fsets,
  type = c("global", "local"),
  fired = NULL,
  vars = NULL,
  specs = NULL
)
```

Arguments

- **rules**: A list of character vectors where each element is a fuzzy set name (a predicate) and thus each such vector forms a rule.
- **fsets**: A valid instance of the `fsets()` class such that all predicates in rules (i.e., all values of all character vectors in rules$rules) can be found in colnames(fsets)
- **type**: The type of perception to use. It can be either "local" or "global" (default).
- **fired**: If type="global" then this argument can be NULL. If type = "local" then fired must be a numeric vector of values in the interval [0, 1] indicating the truth values of all rules, i.e. the length of the vector must be equal to the number of rules in the rules argument.
- **vars**: A deprecated parameter that must be NULL. Formerly, it was a named (typically character) vector that determined which predicates originate from the same variable, i.e. which of them semantically deal with the same property. For that purpose, each value from any vector stored in the rules list must be present in names(vars). See also `vars()` function of the `fsets()` class.
- **specs**: A deprecated parameter that must be NULL. Formerly, it was a square numeric matrix containing values from \{0, 1\}. It is a specificity matrix for which each row and column corresponds to an rules’es predicate specs[i][j] = 1 if and only if the i-th predicate is more specific (i.e. the corresponding fuzzy set is a subset of) than the j-th predicate (i.e. x[,j]). See also `specs()` function of the `fsets()` class.
Details

In other words, for each rule \( x \) in the rules list, it searches for another rule \( y \) such that \( \text{is.specific}(y, x) \) returns TRUE. If yes then \( x \) is removed from the list.

Value

A modified list of rules for which no other more specific rule exists. (Each rule is a vector.)

Author(s)

Michal Burda

See Also

\( \text{is.specific}() \), \( \text{fsets}() \), \( \text{fcut}() \), \( \text{lcut}() \)

Examples

```r
# prepare fsets
f <- lcut(data.frame(a=0:1, b=0:1, c=0:1, d=0:1))

# run perceive function: (sm.a, bi.c) has
# more specific rule (ve.sm.a, bi.c)
perceive(list(c('sm.a', 'bi.c'),
               c('ve.sm.a', 'bi.c'),
               c('sm.b', 'sm.d')),
               f)
```

Description

This function plots the membership degrees stored in the instance of the \( \text{fsets}() \) class. Internally, the membership degrees are transformed into a time-series object and viewed in a plot using the \( \text{ts.plot}() \) function. This function is useful mainly to see the shape of fuzzy sets on regularly sampled inputs.

Usage

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

Arguments

- **x**: An instance of class \( \text{fsets}() \)
- **...**: Other arguments that are passed to the underlying \( \text{ts.plot}() \) function.
Value

Result of the `ts.plot()` method.

Author(s)

Michal Burda

See Also

`fsets()`, `fcut()`, `lcut()`, `ts.plot()`

Examples

```r
# Additional arguments are passed to the ts.plot method
# Here thick lines represent atomic linguistic expressions, 
# i.e. "small", "medium", and "big".
plot(d, 
     ylab="membership degree", 
     xlab="values", 
     gpars=list(lwd=c(rep(1, 3), 5, rep(1, 5), 5, rep(1, 7), 5, rep(1,4))))
```

---

**Description**

Print an instance of the `farules()` S3 class in a human readable form.

**Usage**

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

**Arguments**

- `x` An instance of the `farules()` S3 class
- `...` Unused.

**Value**

None.

**Author(s)**

Michal Burda
**print.frbe**

**See Also**

`farules()`, `searchrules()`

---

**print.frbe**

*Print an instance of the frbe() class*

**Description**

Format an object of the `frbe()` class into human readable form and print it to the output.

**Usage**

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

**Arguments**

- `x` An instance of `frbe()` class
- `...` Unused.

**Details**

Format an object of the `frbe()` class into human readable form and print it to the output.

**Value**

None.

**Author(s)**

Michal Burda

**References**

Štěpnička, M., Burda, M., Štěpničková, L. Fuzzy Rule Base Ensemble Generated from Data by Linguistic Associations Mining. FUZZY SET SYST. 2015.

**See Also**

`frbe()`
Examples

# prepare data (from the forecast package)
library(forecast)
horizon <- 10
train <- wineind[-1 * (length(wineind)-horizon+1):length(wineind)]
test <- wineind[(length(wineind)-horizon+1):length(wineind)]
f <- frbe(ts(train, frequency=frequency(wineind)), h=horizon)
print(f)
print(test)

print.fsets

---

print.fsets  

*Print an instance of the* fsets() *class*

Description

Format an object of the *fsets*() class into human readable form and print it to the output.

Usage

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

Arguments

x  
An instance of the *fsets*() class

...  
Unused.

Value

Nothing

Author(s)

Michal Burda

See Also

fsets(), fcut(), lcut()

Examples

d <- fcut(CO2[, 1:2])
print(d)
rbcoverage  

**Compute rule base coverage of data**

### Description

This function computes rule base coverage, i.e. an average of maximum membership degree at which each row of data fires the rules in rule base.

### Usage

```r
rbcovage(
  x,
  rules,
  tnorm = c("goedel", "goguen", "lukasiewicz"),
  onlyAnte = TRUE
)
```

### Arguments

- **x**: Data for the rules to be evaluated on. Could be either a numeric matrix or numeric vector. If matrix is given then the rules are evaluated on rows. Each value of the vector or column of the matrix represents a predicate - it’s numeric value represents the truth values (values in the interval \([0,1]\)).

- **rules**: Either an object of class "farules" or list of character vectors where each vector is a rule with consequent being the first element of the vector. Elements of the vectors (predicate names) must correspond to the x’s names (of columns if x is a matrix).

- **tnorm**: A character string representing a triangular norm to be used (either "goedel", "goguen", or "lukasiewicz") or an arbitrary function that takes a vector of truth values and returns a t-norm computed of them.

- **onlyAnte**: TRUE if only antecedent-part of a rule should be evaluated. Antecedent-part of a rule are all predicates in rule vector starting from the 2nd position. (First element of a rule is the consequent - see above.)

If FALSE, then the whole rule will be evaluated (antecedent part together with consequent).

### Details

Let \(f_{ij}\) be a truth value of \(i\)-th rule on \(j\)-th row of data \(x\). Then \(m_j = \max(f_{ij})\) is a maximum truth value that is reached for the \(j\)-th data row with the rule base. Then the rule base coverage is a mean of that truth values, i.e. \(rbcoverage = \text{mean}(m_j)\).

### Value

A numeric value of the rule base coverage of given data.
Author(s)

Michal Burda

References


See Also

fire(), reduce()

Examples

```r
x <- matrix(1:20 / 20, nrow=2)
colnames(x) <- letters[1:10]

rules <- list(c('a', 'c', 'e'),
               c('d', 'a'),
               c('c', 'a', 'b'))
rbcov(x, rules, "goguen", TRUE) # returns 1

rules <- list(c('d', 'a'),
               c('c', 'a', 'b'))
rbcov(x, rules, "goguen", TRUE) # returns 0.075)
```

reduce

Reduce the size of rule base

Description

From given rule base, select such set of rules that influence mostly the rule base coverage of the input data.

Usage

```r
reduce(
  x,
  rules,
  ratio,
  tnorm = c("goedel", "goguen", "lukasiewicz"),
  tconorm = c("goedel", "goguen", "lukasiewicz"),
  numThreads = 1
)
```
reduce

Arguments

\( x \) Data for the rules to be evaluated on. Could be either a numeric matrix or numeric vector. If matrix is given then the rules are evaluated on rows. Each value of the vector or column of the matrix represents a predicate - it’s numeric value represents the truth values (values in the interval \([0, 1]\)).

\( \text{rules} \) Either an object of class "farules" or list of character vectors where each vector is a rule with consequent being the first element of the vector. Elements of the vectors (predicate names) must correspond to the \( x \)’s names (of columns if \( x \) is a matrix).

\( \text{ratio} \) A percentage of rule base coverage that must be preserved. It must be a value within the \([0, 1]\) interval. Value of 1 means that the rule base coverage of the result must be the same as coverage of input \( \text{rules} \). A sensible value is e.g. 0.9.

\( \text{tnorm} \) Which t-norm to use as a conjunction of antecedents. The default is "goedel".

\( \text{tconorm} \) Which t-norm to use as a disjunction, i.e. to combine multiple antecedents to get coverage of the rule base. The default is "goedel".

\( \text{numThreads} \) How many threads to use for computation. Value higher than 1 causes that the algorithm runs in several parallel threads (using the OpenMP library).

Details

From a given rulebase, a rule with greatest coverage is selected. After that, additional rules are selected that increase the rule base coverage the most. Addition stops after the coverage exceeds \( \text{originalcoverage} \times \text{ratio} \).

Note that the size of the resulting rule base is not necessarily minimal because the algorithm does not search all possible combination of rules. It only finds a local minimum of rule base size.

Value

Function returns an instance of class \text{farules()} or a list depending on the type of the \( \text{rules} \) argument.

Author(s)

Michal Burda

References


See Also

\text{rbcoverage()}, \text{farules()}

### rmse

**Compute Root Mean Squared Error (RMSE)**

**Description**

RMSE is computed as $\sqrt{\text{mean}((\text{forecast} - \text{validation})^2)}$.

**Usage**

```r
rmse(forecast, validation)
```

**Arguments**

- `forecast`: A numeric vector of forecasted values
- `validation`: A numeric vector of actual (real) values

**Value**

A Root Mean Squared Error (RMSE)

**Author(s)**

Michal Burda

**See Also**

`smape()`, `mase()`, `frbe()`

---

### searchrules

**Searching for fuzzy association rules**

**Description**

This function searches the given `fsets()` object `d` for all fuzzy association rules that satisfy defined constraints. It returns a list of fuzzy association rules together with some statistics characterizing them (such as support, confidence etc.).
searchrules

Usage

searchrules(
  d,
  lhs = 2:ncol(d),
  rhs = 1,
  tnorm = c("goedel", "goguen", "lukasiewicz"),
  n = 100,
  best = c("confidence"),
  minSupport = 0.02,
  minConfidence = 0.75,
  maxConfidence = 1,
  maxLength = 4,
  numThreads = 1,
  trie = (maxConfidence < 1)
)

Arguments

d  An object of class fsets() - it is basically a matrix where columns represent the fuzzy sets and values are the membership degrees. For creation of such object, use fcut() or lcutf() function.

lhs  Indices of fuzzy attributes that may appear on the left-hand-side (LHS) of association rules, i.e. in the antecedent.

rhs  Indices of fuzzy attributes that may appear on the right-hand-side (RHS) of association rules, i.e. in the consequent.

tnorm  A t-norm to be used for computation of conjunction of fuzzy attributes. (Allowed are even only starting letters of "lukasiewicz", "goedel" and "goguen").

n  The non-negative number of rules to be found. If zero, the function returns all rules satisfying the given conditions. If positive, only n best rules are returned. The criterium of what is “best” is specified with the best argument.

best  Specifies measure accordingly to which the rules are ordered from best to worst. This argument is used mainly in combination with the n argument. Currently, only single value ("confidence") can be used.

minSupport  The minimum support degree of a rule. Rules with support below that number are filtered out. It must be a numeric value from interval [0, 1]. See below for details on how the support degree is computed.

minConfidence  The minimum confidence degree of a rule. Rules with confidence below that number are filtered out. It must be a numeric value from interval [0, 1]. See below for details on how the confidence degree is computed.

maxConfidence  Maximum confidence threshold. After finding a rule that has confidence degree above the maxConfidence threshold, no other rule is resulted based on adding some additional attribute to its antecedent part. I.e. if "Sm.age & Me.age => Sm.height" has confidence above maxConfidence threshold, no another rule containing "Sm.age & Me.age" will be produced regardless of its interest measures.

If you want to disable this feature, set maxConfidence to 1.
**searchrules**

- **maxLength**
  Maximum allowed length of the antecedent, i.e., maximum number of predicates that are allowed on the left-hand side of the rule. If negative, the maximum length of rules is unlimited.

- **numThreads**
  Number of threads used to perform the algorithm in parallel. If greater than 1, the OpenMP library (not to be confused with Open MPI) is used for parallelization. Please note that there are known problems of using OpenMP together with another means of parallelization that may be used within R. Therefore, if you plan to use the `searchrules` function with some of the external parallelization mechanisms such as library doMC, make sure that `numThreads` equals 1. This feature is available only on systems that have installed the OpenMP library.

- **trie**
  Whether or not to use internal mechanism of Tries. If FALSE, then in the output may appear such rule that is a descendant of a rule that has confidence above `maxConfidence` threshold.
  Tries consume very much memory, so if you encounter problems with insufficient memory, set this argument to FALSE. On the other hand, the size of result (if `n` is set to 0) can be very high if `trie` is set to FALSE.

### Details

The function searches data frame `d` for fuzzy association rules that satisfy conditions specified by the parameters.

### Value

A list of the following elements: `rules` and `statistics`.

- `rules` is a list of mined fuzzy association rules. Each element of that list is a character vector with consequent attribute being on the first position.

- `statistics` is a data frame of statistical characteristics about mined rules. Each row corresponds to a rule in the `rules` list. Let us consider a rule "a & b => c", let $\otimes$ be a t-norm specified with the `tnorm` parameter and $i$ goes over all rows of a data table `d`. Then columns of the `statistics` data frame are as follows:

  - support: a rule’s support degree: $1/nrow(d) \times \sum_{i \in d} a(i) \otimes b(i) \otimes c(i)$
  - lhsSupport: a support of rule’s antecedent (LHS): $1/nrow(d) \times \sum_{i \in d} a(i) \otimes b(i)$
  - rhsSupport: a support of rule’s consequent (RHS): $1/nrow(d) \times \sum_{i \in d} c(i)$
  - confidence: a rule’s confidence degree: $\text{support}/\text{lhsSupport}$

### Author(s)

Michal Burda

### See Also

`fcut()`, `lcut()`, `farules()`, `fsets()`, `pblclib()`
Examples

```r
d <- lcutf(C02)
searchrules(d, lhs=1:ncol(d), rhs=1:ncol(d))
```

---

`slices`  
返回给定区间内等间距的值的向量

**Description**

返回给定区间内等间距的值的向量，大小为给定的值。

**Usage**

`slices(from, to, n)`

**Arguments**

- `from`: 区间下界
- `to`: 区间上界
- `n`: 向量的长度

**Details**

返回从 `from` 到 `to` (含) 区间内的值，等间距，总长度为 `n`。此函数可用于 `pbld`或 `defuzz` 函数（对于值参数；参见 `lcut` 或 `fcut`）。

**Value**

一个等间距的向量。

**Author(s)**

Michal Burda

**See Also**

`pbld, defuzz, fcut, lcut`
Compute Symmetric Mean Absolute Percentage Error (SMAPE)

SMAPE is computed as \( \text{mean}(\text{abs}(\text{forecast} - \text{validation})/((\text{abs}(\text{forecast})+\text{abs}(\text{validation}))/2)). \)

Usage

smape(forecast, validation)

Arguments

- forecast: A numeric vector of forecasted values
- validation: A numeric vector of actual (real) values

Value

A Symmetric Mean Absolute Percentage Error (SMAPE)

Author(s)

Michal Burda

See Also

rmse(), mase(), frbe()
Modify algebra’s way of computing with NA values.

Description

By default, the objects created with the `algebra()` function represent a mathematical algebra capable to work on the \([0, 1]\) interval. If NA appears as a value instead, it is propagated to the result. That is, any operation with NA results in NA, by default. This scheme of handling missing values is also known as Bochvar’s.

Usage

```r
sobocinski(algebra)
kleeene(algebra)
dragonfly(algebra)
```

Arguments

- `algebra` the underlying algebra object to be modified – see the `algebra()` function

Details

The `sobocinski()`, `kleeene()` and `dragonfly()` functions modify the algebra to handle the NA in a different way than default. Sobocinski’s algebra simply ignores NA values whereas Kleene’s algebra treats NA as "unknown value". Dragonfly approach is a combination of Sobocinski’s and Bochvar’s approach, which preserves the ordering 0 <= NA <= 1 to obtain from compositions (see `compose()`) the lower-estimate in the presence of missing values.

In detail, the behaviour of the algebra modifiers is defined as follows:

Sobocinski’s negation for `n` being the underlying algebra:

\[
\begin{array}{c|c|c}
\text{a} & \text{n(a)} & \text{NA} \\
\hline
\text{NA} & 0 & 0
\end{array}
\]

Sobocinski’s operation for `op` being one of `t`, `pt`, `c`, `pc`, `i`, `pi`, `s`, `ps` from the underlying algebra:

\[
\begin{array}{c|c|c|c}
\text{b} & \text{NA} & \text{a} & \text{a} \\
\hline
\text{NA} & \text{b} & \text{NA} & \text{NA}
\end{array}
\]

Sobocinski’s operation for `r` from the underlying algebra:

\[
\begin{array}{c|c|c|c}
\text{b} & \text{NA} & \text{a} & \text{n(a)} \\
\hline
\text{NA} & \text{b} & \text{NA} & \text{NA}
\end{array}
\]
Kleene’s negation is identical to \( n \) from the underlying algebra.

Kleene’s operation for \( \text{op} \) being one of \( t, pt, i, pi \) from the underlying algebra:

\[
\begin{array}{c|c|c|c|c}
\text{b} & \text{NA} & 0 \\
\hline
\text{a \ op(a, b)} & \text{NA} & 0 \\
\hline
\text{NA} & \text{NA} & \text{NA} & 0 \\
\hline
0 & 0 & 0 & 0
\end{array}
\]

Kleene’s operation for \( \text{op} \) being one of \( c, pc, s, ps \) from the underlying algebra:

\[
\begin{array}{c|c|c|c|c}
\text{b} & \text{NA} & 1 \\
\hline
\text{a \ op(a, b)} & \text{NA} & 1 \\
\hline
\text{NA} & \text{NA} & \text{NA} & 1 \\
\hline
1 & 1 & 1 & 1
\end{array}
\]

Kleene’s operation for \( r \) from the underlying algebra:

\[
\begin{array}{c|c|c|c|c}
\text{b} & \text{NA} & 1 \\
\hline
\text{a \ r(a, b)} & \text{NA} & 1 \\
\hline
\text{NA} & \text{NA} & \text{NA} & 1 \\
\hline
0 & 1 & 1 & 1
\end{array}
\]

Dragonfly negation is identical to \( n \) from the underlying algebra.

Dragonfly operation for \( \text{op} \) being one of \( t, pt, i, pi \) from the underlying algebra:

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c}
\text{b} & \text{NA} & 0 & 1 \\
\hline
\text{a \ op(a, b)} & \text{NA} & 0 & a \\
\hline
\text{NA} & \text{NA} & \text{NA} & 0 & 0 & NA \\
\hline
0 & 0 & 0 & 0 & 0 & \\
\hline
1 & b & NA & 0 & 1
\end{array}
\]

Dragonfly operation for \( \text{op} \) being one of \( c, pc, s, ps \) from the underlying algebra:

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c}
\text{b} & \text{NA} & 0 & 1 \\
\hline
\text{a \ op(a, b)} & a & a & 1 \\
\hline
\text{NA} & b & NA & NA & 1 \\
\hline
0 & b & NA & 0 & 1 \\
\hline
1 & 1 & 1 & 1 & 1
\end{array}
\]

Dragonfly operation for \( r \) from the underlying algebra:

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c}
\text{b} & \text{NA} & 0 & 1 \\
\hline
\text{a \ r(a, b)} & \text{NA} & n(a) & 1 \\
\hline
\text{NA} & b & 1 & NA & 1 \\
\hline
0 & 1 & 1 & 1 & 1 \\
\hline
1 & b & NA & 0 & 1
\end{array}
\]
### triangle

**Value**

A list of functions of the same structure as is the list returned from the `algebra()` function.

**Author(s)**

Michal Burda

**Examples**

```r
a <- algebra('lukas')
b <- sobocinski(a)

a$t(0.3, NA)  # NA
b$t(0.3, NA)  # 0.3
```

---

#### triangle

*Deprecated functions to compute membership degrees of numeric fuzzy sets*

**Description**

These functions compute membership degrees of numeric fuzzy sets with triangular or raised-cosinal shape. These functions are *deprecated*. Please use `triangular()` or `raisedcosinal()` functions instead.

**Usage**

```r
triangle(x, lo, center, hi)

raisedcos(x, lo, center, hi)
```

**Arguments**

- `x`: A numeric vector to be transformed.
- `lo`: A lower bound (can be `-Inf`).
- `center`: A peak value.
- `hi`: An upper bound (can be `Inf`).

**Value**

A numeric vector of membership degrees of `x` to a fuzzy set with the shape determined with `lo`, `center`, `hi`.

**Author(s)**

Michal Burda
See Also

`triangular()`, `raisedcosinal()`

---

**triangular**

Factories for functions that convert numeric data into membership degrees of fuzzy sets

---

**Description**

These functions create functions with a single argument \( x \) that compute membership degrees of \( x \) to a fuzzy set of either triangular or raised-cosine shape that is defined by \( \text{lo} \), \( \text{center} \), and \( \text{hi} \).

**Usage**

```r
triangular(lo, center, hi)

raisedcosinal(lo, center, hi)
```

**Arguments**

- \( \text{lo} \)  
  A lower bound (can be \(-\text{Inf}\)).
- \( \text{center} \)  
  A peak value.
- \( \text{hi} \)  
  An upper bound (can be \(\text{Inf}\)).

**Details**

The arguments must satisfy \( \text{lo} \leq \text{center} \leq \text{hi} \). Functions compute membership degrees of triangular or raised-cosine fuzzy sets. \( x \) values equal to \( \text{center} \) obtain membership degree equal to 1, \( x \) values lower than \( \text{lo} \) or greater than \( \text{hi} \) obtain membership degree equal to 0. A transition of the triangular (resp. raised cosinal) shape (with peak at \( \text{center} \)) is computed for \( x \) values between \( \text{lo} \) and \( \text{hi} \).

If \( \text{lo} = -\text{Inf} \) then any value that is lower or equal to \( \text{center} \) gets membership degree 1. Similarly, if \( \text{hi} = \text{Inf} \) then any value that is greater or equal to \( \text{center} \) gets membership degree 1. \( \text{NA} \) and \( \text{NaN} \) values remain unchanged.

`triangular()` produces fuzzy sets of a triangular shape (with peak at \( \text{center} \)), `raisedcosinal()` produces fuzzy sets defined as a raised cosine hill.

**Value**

A function with single argument \( x \) that should be a numeric vector to be converted.

**Author(s)**

Michal Burda

**See Also**

`fcut()`
Examples

tr <- triangular(1, 2, 3)
tr(1:30 / 3)

rc <- raisedcosinal(1, 2, 3)
rc(1:30 / 3)

plot(triangular(-1, 0, 1), from=-2, to=3)
plot(triangular(-1, 0, 2), from=-2, to=3)
plot(triangular(-Inf, 0, 1), from=-2, to=3)
plot(triangular(-1, 0, Inf), from=-2, to=3)

plot(raisedcosinal(-1, 0, 1), from=-2, to=3)
plot(raisedcosinal(-1, 0, 2), from=-2, to=3)
plot(raisedcosinal(-Inf, 0, 1), from=-2, to=3)
plot(raisedcosinal(-1, 0, Inf), from=-2, to=3)
Index

*Topic datasets
  defaultHedgeParams, 20
  lingexpr, 47
*Topic models
  aggregateConsequents, 3
  algebra, 5
  antecedents, 8
  as.data.frame.farules, 9
  as.data.frame.fsets, 10
  c.farules, 11
  cbind.fsets, 12
  compose, 13
  consequents, 15
  ctx, 16
  defuzz, 21
  evalfrbe, 22
  farules, 23
  fcut, 24
  fire, 27
  frbe, 29
  fsets, 31
  hedge, 33
  horizon, 35
  is.farules, 36
  is.frbe, 37
  is.fsets, 38
  is.specific, 38
  lcut, 40
  lcut3, 44
  lingexpr, 47
  mult, 51
  pbld, 52
  perceive, 54
  plot.fsets, 55
  print.farules, 56
  print.frbe, 57
  print.fsets, 58
  rbcoverage, 59
  reduce, 60
  searchrules, 62
  slices, 65
  sobocinski, 67
  triangular, 70
*Topic multivariate
  as.data.frame.farules, 9
  compose, 13
  fcut, 24
  fire, 27
  lcut, 40
  mult, 51
  plot.fsets, 55
  rbcoverage, 59
  reduce, 60
  searchrules, 62
  triangular, 70
*Topic robust
  aggregateConsequents, 3
  algebra, 5
  antecedents, 8
  as.data.frame.farules, 9
  as.data.frame.fsets, 10
  c.farules, 11
  cbind.fsets, 12
  compose, 13
  consequents, 15
  ctx, 16
  defuzz, 21
  evalfrbe, 22
  farules, 23
  fcut, 24
  fire, 27
  frbe, 29
  fsets, 31
  hedge, 33
  horizon, 35
  is.farules, 36
  is.frbe, 37
  is.fsets, 38
  pebb, 52
  perceive, 54
  plot.fsets, 55
  print.farules, 56
  print.frbe, 57
  print.fsets, 58
  rbcoverage, 59
  reduce, 60
  searchrules, 62
  slices, 65
  sobocinski, 67
  triangular, 70
aggregateConsequents, 3
aggregateConsequents(), 22, 29, 53
algebra, 5
algebra(), 13, 67, 69
allowed.lingexpr, 42, 48
allowed.lingexpr (lingexpr), 47
antecedents, 8
antecedents(), 15, 28
as.ctx3 (ctx), 16
as.ctx3bilat (ctx), 16
as.ctx5 (ctx), 16
as.ctx5bilat (ctx), 16
as.data.frame.farules, 9
as.data.frame.fsets, 10
as.matrix.fsets (as.data.frame.fsets), 10
base::cut(), 25
base::sort(), 14
c, 11
c.farules, 11
cbind(), 12
cbind.fsets, 12
cbind.fsets(), 26, 27
compose, 13
compose(), 51, 67
consequents, 15
consequents(), 9, 28
ctx, 16, 42
ctx(), 34, 48
data.frame(), 9
defaultHedgeParams, 20
defuzz, 21, 65
defuzz(), 4, 29, 53
doMC::registerDoMC(), 26, 46, 52
dragonfly (sobocinski), 67
evalfrbe, 22
evalfrbe(), 31
farules, 23, 37
farules(), 8, 9, 11, 15, 27–30, 37, 52, 56, 57, 61, 64
fcut, 24, 65
fcut(), 3, 4, 10, 12, 20, 22, 29, 32–34, 36, 40, 41, 43–45, 48, 52, 55, 56, 58, 63, 64, 70
fire, 27
fire(), 4, 22, 53, 60
foreach::foreach(), 26, 46, 52
forecast::auto.arima(), 30
forecast::ets(), 30
forecast::ruf(), 30
frbe, 29
frbe(), 22, 23, 37, 49, 57, 62, 66
fsets, 10, 31
fsets(), 10, 32, 38, 39, 41, 43, 46, 52, 54–56, 58, 62–64
goedel.biresiduum (algebra), 5
goedel.residuum (algebra), 5
goedel.tconorm (algebra), 5
goedel.tnorm (algebra), 5
goguen.biresiduum (algebra), 5
goguen.residuum (algebra), 5
goguen.tconorm (algebra), 5
goguen.tnorm (algebra), 5
hedge, 33
hedge(), 20, 35, 36, 48
INDEX

horizon, 35
horizon(), 20, 33, 34, 48

invol.neg(algebra), 5
is.algebra(algebra), 5
is.ctx3(ctx), 16
is.ctx3bilat(ctx), 16
is.ctx5(ctx), 16
is.ctx5bilat(ctx), 16
is.farules, 36
is.frbe, 37
is.fsets, 38
is.specific, 38
is.specific(), 33, 54, 55

kleene (sobocinski), 67

lcut, 40, 65
lcut(), 3, 4, 10, 12, 20, 22, 24, 27, 29, 33, 34, 36, 44, 48, 52, 53, 55, 56, 58, 63, 64

lcut3, 44
lcut5(lcut3), 44
lf1, 46
lingexpr, 47
lingexpr(), 13, 14, 19, 20, 34, 36
lukas.biresiduum(algebra), 5
lukas.residuum(algebra), 5
lukas.residuum(), 3
lukas.tconorm(algebra), 5
lukas.tnorm(algebra), 5

mase, 49
mase(), 22, 23, 62, 66
minmax, 50
minmax(), 20
mult, 51
mult(), 14

pbld, 52, 65
pbld(), 21, 22, 27, 29, 39, 42, 54, 64
perceive, 54
perceive(), 4, 22, 29, 39
pgoedel.tconorm(algebra), 5
pgoedel.tconorm(), 4
pgoedel.tnorm(algebra), 5
pgoedel.tnorm(), 3, 4, 28
pgoguen.tconorm(algebra), 5
pgoguen.tnorm(algebra), 5
pgoguen.tnorm(), 28

plot.fsets, 55
plukas.tconorm(algebra), 5
plukas.tnorm(algebra), 5
plukas.tnorm(), 28
print.farules, 56
print.frbe, 57
print.fsets, 58

raisedcos(triangle), 69
raisedcosinal(triangular), 70
raisedcosinal(), 25, 26, 69, 70
rbcoverage, 59
rbcoverage(), 61
reduce, 60
reduce(), 60
rmse, 62
rmse(), 22, 23, 49, 66

searchrules, 62
searchrules(), 9, 11, 15, 23, 24, 53, 57
slices, 65
smape, 66
smape(), 22, 23, 49, 62
sobocinski, 67
specs(fsets), 31
specs(), 10, 12, 26, 27, 32, 38, 39, 43, 46, 54
specs<- (fsets), 31
strict.neg(algebra), 5

triangle, 69
triangular, 70
triangular(), 25, 26, 69, 70
ts.plot(), 53, 56

vars(fsets), 31
vars(), 10, 12, 26, 27, 32, 38, 39, 43, 46, 54
vars<- (fsets), 31