Package 'rLDCP'

May 9, 2026

Type Package

Title Text Generation from Data

Version 1.0.2

Date 2017-11-10

Author Patricia Conde-Clemente [aut, cre], Jose M. Alonso [aut], Gracian Trivino [aut]

Maintainer Patricia Conde-Clemente <patricia.condeclemente@gmail.com>

Description Linguistic Descriptions of Complex Phenomena (LDCP) is an architecture and methodology that allows us to model complex phenomena, interpreting input data, and generating automatic text reports customized to the user needs (see <doi:10.1016/j.ins.2016.11.002> and <doi:10.1007/s00500-016-2430-5>). The proposed package contains a set of methods that facilitates the development of LDCP systems. It main goal is increasing the visibility and practical use of this research line.

License GPL (>= 2) | file LICENSE

URL http://phedes.com/rLDCP

LazyData FALSE

NeedsCompilation no

RoxygenNote 6.0.1

Suggests testthat

Imports XML (>= 3.98-1.4), methods

Repository CRAN

Date/Publication 2017-11-10 16:42:11 UTC

Contents

---

cp *Define the CP*

Description

In general, CP corresponds with specific parts of the analyzed phenomenon at a certain degree of granularity. To create a computational model of the analyzed phenomenon, the designer analyzes the everyday use of natural language about the monitored phenomenon with the aim of identifying different parts (units of information or granules) based on his/her subjective perceptions. According with Zadeh (1996), a granule is a clump of elements which are drawn together by indistinguishability, similarity, proximity or functionality. The GLMP handles granules by using CPs.

Usage

```
cp(name, a, b = NULL, r = NULL)
```

Arguments

name	is the identifier of the CP.
a	is a vector A = (a1, a2, . . . , an) of linguistic expressions that represents the whole linguistic domain of CP, e.g. we have the linguistic domain "statistical data" that is represented with three linguistic variables (bad, good, very good).
b	is a vector B = (b1, b2, . . . , bn) of linguistic expressions (words or sentences in natural language) that represents the reliability of the CP, e.g., the reliability of the "statistical data" are (low, moderate, high). By default (b = NULL), the CP does not manage information about reliability.

r — is a vector R = (r1, r2, ... , rn) of relevance degrees 0 <= ri <=1 assigned to each ai in the specific context, e.g., the relevance of the linguistic expressions (bad, good, very good) is (0.5, 0.5, 1) means the perception of "very good" is more relevant than the other two choices. By default (r = NULL), the function create a r vector with the maximum degree of relevance for all linguistic expression, e.g., (1,1,1).

Value

The generated CP = list(a, w, r, b, wb) where w and wb are vectors with the validity degrees (wi and wbi in [0,1]) of the linguistic expressions in a and b respectively. These vectors are initialized with 0.

Examples

```
myCP <-cp("myCP", c("bad", "good", "very good"))
myCP <- cp("myCP", c("bad", "good", "very good"), c("low", "moderate", "high"))
myCP <- cp("myCP", c("bad", "good", "very good"), r=c(1,0.8,0.9))
myCP <- cp("myCP", c("bad", "good", "very good"), c("low", "moderate", "high"), c(1,0.8,0.9))
```

---

data_structure *Define the data structure*

Description

Data structure provides the GLMP input. It constructor receives the input values and the method that defines the data structure, i.e., the set of preprocesing techniques.

Usage

```
data_structure(input, method)
```

Arguments

input — is the input data. May be a vector, list or matrix with numbers.

method — is the function with the data preprocesing techniques needed to prepare the GLMP input. The method must have one argument, the input data:

```
my_method <- function(input)
```

Value

The generated data_structure = list(input, method)

Examples

```
values <- matrix(c(34,11,9,32), ncol=2)

my_method <- function (input){
 output <- c(mean(input[,1]), mean(input[,2]))
 output
}
 my_data_structure <- data_structure(values,my_method)
```

`degree_mf` *Define generic calculation of fuzzy membership degrees*

Description

It is a generic function in charge of computing fuzzy membership degrees. Namely, it identifies the specific membership function to consider and run the related method for computing the membership degree for a given input value. It takes as input an object (`trapezoid_mf`, `triangle_mf` and `fuzzy_partitions`) and the related input values

Usage

```
degree_mf(shape, input)
```

Arguments

`shape`	is the object (`trapezoid_mf`, `triangle_mf` and `fuzzy_partitions`) to dispatch to.
`input`	is the value to be assess.

Value

the membership degree for a given input values.

Examples

```
w <- degree_mf(triangle_mf(450,450,550),450)

w <- degree_mf(fuzzy_partitions(triangle_mf(450,450,550),
                                triangle_mf(450,550,600),
                                trapezoid_mf(550,600,800,1000),
                                triangle_mf(800,1000,1300),
                                trapezoid_mf(1000,1300,1500,1500)),450)
```

fuzzy_partitions *Define the fuzzy parititions*

Description

It is a constructor of fuzzy partitions, it defines a set of membership functions. It takes as input a set of `trapezoid_mf` or `triangle_mf` or objects in the shape_mf class.

Usage

```
fuzzy_partitions(...)
```

Arguments

`...`	are the diferent partitions, e.g., `trapezoid_mf` or `triangle_mf`.

Value

the `(fuzzy_partitions <- list(...)`

Examples

```
fuzzy_partitions(triangle_mf(450,450,550),
                 triangle_mf(450,550,600),
                 trapezoid_mf(550,600,800, 1000),
                 triangle_mf(800,1000,1300),
                 trapezoid_mf(1000,1300,1500,1500))
```

fuzzy_rule *Define the fuzzy rule*

Description

We define a fuzzy rule using the numbers 1 and 0. rule(0,0,1,0,0, 0,0,1,0,0, 0,0,1,0,0, 0,0,1)

This is an example of fuzzy_rule(0,0,1,0,0,1). In the fuzzy rule the number 1 means that the linguistic expression is included and the number 0 means that the linguistic expression is not included.

Usage

```
fuzzy_rule(...)
```

Arguments

`...`	the 0 and 1 that compose the fuzzy rule.

Value

the `fuzzy_rule <- c(...)`

Examples

```
# For example, the rule "IF CPtemp IS warm THEN CPcomfort IS very comfortable"
#is coded as:

 fuzzy_rule(0,1,0,0,0,1)

# Where, the first three values (0,1,0) correspond with the linguistic
# expressions Atemp=(cold, warm, hot) that define the room temperature (CPtemp).
# The last three values (0,0,1) are related to the linguistic expressions
# Acomfort=(uncomfortable, comfortable and very comfortable) that define
# the room comfort (CPcomfort).
#
```

`fuzzy_rules` *Define the fuzzy rules*

Description

It is a constructor of fuzzy rules, the arguments are the diferent `fuzzy_rule` object.

Usage

```
fuzzy_rules(...)
```

Arguments

`...`	one or more `fuzzy_rule` objects.

Value

`fuzzy_rules <- list(...)`

Examples

```
fuzzy_rules(fuzzy_rule(0,0,1, 0,0,1, 0,0,1),
     fuzzy_rule(1,0,0, 1,0,0, 1,0,0),
     fuzzy_rule(0,1,0, 0,1,0, 0,1,0))
```

generate_code *Generate the R code*

Description

The function takes as input the path to a XML file that contains a LDCP system. Then it generates its corresponding in R code. This R code is stored in an output file. The output file path is another function parameter.

Usage

```
generate_code(input, output)
```

Arguments

input	is the XML source path file. E.g. "/folder/ldcp_system.xml".
output	is the R destination path file. E.g. "/folder/ldcp_system.R".

Value

If the process ends without error, the user will receive a message that indicates that the code has been generated successfully. Otherwise, the user will receive the detailed list of errors.

Examples

```
## Not run: generate_code('extdata/comfortableroom','comfortableroom')

## The code has been generated successfully
```

glmp *Define the GLMP*

Description

Granular Linguistic Model of Phenomena (GLMP) is a network of cp and pm objects. that allows the designer to model computationally her/his own perceptions. The input data are introduced into the model through 1PMs which interpret the input data and create CPs. Then, 2PMs take several CPs as input and generate 2CPs. Of course, following the same scheme, is possible to add additional upper levels.

The glmp constructor receive as arguments the list of pms and the method with the computational model.

Usage

```
glmp(pms, method)
```

Arguments

pms	is the list of pm objects included in the glmp.
method	is the function with the glmp computational model. The method must have two arguments: the list of pm objects defined in the glmp and the input data: my_glmp_method <- function(pm,input)

Value

The generated glmp = list(pm, method)

Examples

```
## Not run: glmp_method <- function(pm,input){

  pm$pm_depth   <- pm_infer(pm$pm_depth, input[1])
  pm$pm_height  <- pm_infer(pm$pm_height,input[2])
  pm$pm_width   <- pm_infer(pm$pm_width, input[3])

  pm$pm_frame  <- pm_infer(pm$pm_frame, list( pm$pm_depth$y,
                                              pm$pm_height$y,
                                              pm$pm_width$y)
  )
  pm
}

my_glmp <- glmp(list(pm_depth  = pm_depth,
                     pm_height = pm_height,
                     pm_width  = pm_width,
                     pm_frame  = pm_frame),
                glmp_method)

## End(Not run)
```

---

infer_rules *Make the inference*

Description

Make an inference with the fuzzy rules.

Usage

```
infer_rules(rules, operator, input)
```

Arguments

rules	the set of fuzzy rules.
operator	the operator object.
input	is the list of validity degrees related to the input cp objects.

Value

A vector that containd the result of the inference.

Examples

```
## In the example the input of the fuzzy rule correspond with two CPs and each CP has 3
## linguistic variables, e.g, {"bad", "good", "very good"}. The output also
## correspond with a CP with 3 linguistic variables.

 infer_rules(fuzzy_rules(fuzzy_rule(0,0,1, 0,0,1, 0,0,1),
                  fuzzy_rule(1,0,0, 1,0,0, 1,0,0),
                  fuzzy_rule(0,1,0, 0,1,0, 0,1,0)),
             operator(min, max),
             list(c(0,0.5,0.5),c(0.5,0.5,0)))
## [1] 0.0 0.5 0.0
```

---

ldcp *Define the LDCP system*

Description

Linguistic Descriptions of Complex Phenomena (LDCP) is a technology focused on modeling complex phenomena, interpreting input data and generating automatic text reports customized to the user needs. #' The `ldcp` constructor receive as arguments: the `data_structure`, the `glmp` and the `report_template`.

Usage

```
ldcp(data, glmp, report)
```

Arguments

`data`	is the `data_structure` object.
`glmp`	is the `glmp` object.
`report`	is the `report_template` object.

Value

The generated system `ldcp = list(data, glmp, report)`

See Also

`cp` and `pm`

Examples

```
## Not run: my_ldcp <- ldcp(my_data,my_glmp,my_report)
```

ldcp_run *Execute the LDCP system*

Description

Execute the ldcp system in order to obtain the linguistic report. This method follows these three sequential steps 1) Data acquisition, 2) Interpretation and 3) Report generation. Data acquisition process gets the input data and prepares the data structure. Then, the data are interpreted using the GLMP. The result is a set of computational perceptions (CP) that are valid to describe these data. Finally, the report generation process generates a linguistic report using the report template and the previous set of CPs.

Usage

```
ldcp_run(ldcp, input = NULL)
```

Arguments

ldcp	is the ldcp system.
input	is the system input data. May be a vector, list, or matrix with numbers. It is a new input to the data_structure object. By default, is NULL.

Value

The ldcp object that contains the execution results.

Examples

```
## Not run: my_ldcp <- ldcp_run(my_ldcp)
```

operator *Define the operator*

Description

The operator defines the conjunction and disjunction functions used in the fuzzy rules. It takes as input parameters the function used to implement the conjunction, and the function used to implement the disjunction, e.g., "operator(min, max)", where min and max are functions defined by the R language that calculate the maximum and minimum, respectively, from a set of values received as input. Note that, we implicitly assign to the fuzzy implication operator (THEN) the function given for conjunction

Usage

```
operator(conjunction, disjunction)
```

Arguments

conjunction	is the method used to make the conjunction.
disjunction	is the method used to make the disjunction.

Value

the opertator object my_op <- list(conjunction, disjunction).

Examples

```
operator <- operator(min, max)
```

---

pm *Define the PM*

Description

Perception Mapping (PM) is used to create and aggregate cp objects. Each PM receives a set of inputs (cp objects or numerical values) which are aggregated into a single CP.

Usage

```
pm(u = NULL, y, g, t = NULL)
```

Arguments

u	is a vector of n input cps u = (u1, u2, . . . , un). In the special case of first level perception mappings (1PM) the inputs are numerical values provided either by sensors or obtained from a database.
y	is the output cp.
g	is an aggregation function employed to calculate w from the input cps. For example, g might be implemented using a set of fuzzy rules. In the case of 1PMs, g is built using a set of membership functions.
t	is a text generation algorithm which allows generating the sentences in A. In simple cases, t is a linguistic template, e.g., cat("Alabama has", value, "the number of women in the last census".

Value

The generated pm = list(u,y,g,t)

See Also

cp

Examples

```
## Not run: cp_depth <- cp("cp_depth",c("far",
                           "bit far",
                           "good",
                           "close",
                           "very close"))

g_depth <- function(u,y){
  y$w <- degree_mf(fuzzy_partitions(triangle_mf(450,450,550),
                                    triangle_mf( 450,550,600),
                                    trapezoid_mf(550,600,800, 1000),
                                    triangle_mf( 800,1000,1300),
                                    trapezoid_mf( 1000,1300,1500,1500)),u)
  y
}

pm_depth  <- pm(y=cp_depth, g=g_depth)

############################ HEIGHT DEFINITION ##########################################
cp_height <- cp("cp_height", c("high",
                               "average high",
                               "centered",
                               "average low",
                               "low"))

g_height <- function(u,y){
  y$w <- degree_mf(fuzzy_partitions(trapezoid_mf(-1000,-1000,-600,-400),
                                                 triangle_mf(-600,-400,0),
                                                 trapezoid_mf(-400,0,200,400),
                                                 triangle_mf(200,400,600),
                                                 trapezoid_mf(400,600,1000,1000)),u)
  y
}

pm_height <- pm(y=cp_height, g=g_height)

############################ WIDTH DEFINITION ###########################################
cp_width <- cp("cp_width",  c("left",
                              "average left",
                               "centered",
                               "average right",
                                "right"))

g_width <- function(u,y){
  y$w <- degree_mf(fuzzy_partitions(triangle_mf(-1000,-600,-400),
                                                 triangle_mf(-600,-400,0),
                                                triangle_mf(-400,0,400),
                                                 triangle_mf(0,400,600),
                                                 triangle_mf(400,600,1000,1000)),
                              u)
  y
}
```

```
pm_width  <- pm(y=cp_width,  g=g_width)

########################### FRAME DEFINITION ##########################################
cp_frame <- cp("cp_frame", c("bad",
                             "middle",
                             "good"))

g_frame <- function(u,y){

  operator <- operator(min, max)

  y$w<- infer_rules(fuzzy_rules( fuzzy_rule(0,0,1,0,0, 0,0,1,0,0, 0,0,1,0,0, 0,0,1),
                           fuzzy_rule(1,1,1,1,1, 1,1,1,1,1, 1,1,0,1,1, 1,0,0),
                           fuzzy_rule(1,1,1,1,1, 1,0,0,0,1, 0,0,1,0,0, 1,0,0),
                           fuzzy_rule(1,0,0,0,1, 1,1,1,1,1, 0,0,1,0,0, 1,0,0),
                           fuzzy_rule(0,1,0,1,0, 0,1,0,1,0, 0,0,1,0,0, 0,1,0)),
                    operator,
                    list(u[[1]]$w,u[[2]]$w,u[[3]]$w))

  y
}

t_frame <- function(y){

  templates <- c("It has been taken a bad framed photo",
                 "It has been taken a middle framed photo",
                 "It has been taken a good framed photo")

  return( templates[which.max(y$w)])
}

pm_frame <-  pm(y=cp_frame, g=g_frame, t=t_frame)

## End(Not run)
```

`pm_infer` *Call the g function*

Description

It call the g function in order to make the inference, i.e., map inputs u to output y.

Usage

```
pm_infer(pm, u = NULL)
```

Arguments

pm	is the pm object.
u	is the new pm input. By default is NULL.

Value

the pm obtained after calling g.

See Also

cp

Examples

```
cp_depth <- cp("cp_depth", c("far",
                             "bit far",
                             "good",
                             "close",
                             "very close"))

g_depth <- function(u,y){
   y$w <- degree_mf(fuzzy_partitions(triangle_mf(450,450,550),
                                                 triangle_mf( 450,550,600),
                                                 trapezoid_mf(550,600,800, 1000),
                                                 triangle_mf( 800,1000,1300),
                                                 trapezoid_mf( 1000,1300,1500,1500)),u)
 y
}

pm_depth  <- pm(y=cp_depth, g=g_depth)
pm_depth   <- pm_infer(pm_depth, 650)
```

---

pm_multidimensional *Define the pm of a multidimensional cp*

Description

It is a set of pms that infer a multidimensional cp.

Usage

```
pm_multidimensional(...)
```

Arguments

...	the set of pms

Value

The generated pm_multidimensional <- list(...)

pm_report *Generate the report of y*

Description

It call the t function in order to generate the linguistic descriptions that better describe the output y.

Usage

```
pm_report(pm)
```

Arguments

pm	is the pm object.

Value

the description obtained after calling t.

Examples

```
cp_frame <- cp("cp_frame", c("bad",
                             "middle",
                             "good"))

                             g_frame <- function(u,y){

operator <- operator(min, max)

  y$w<- infer_rules(fuzzy_rules( fuzzy_rule(0,0,1,0,0, 0,0,1,0,0, 0,0,1,0,0, 0,0,1),
                             fuzzy_rule(1,1,1,1,1, 1,1,1,1,1, 1,1,0,1,1, 1,0,0),
                             fuzzy_rule(1,1,1,1,1, 1,0,0,0,1, 0,0,1,0,0, 1,0,0),
                             fuzzy_rule(1,0,0,0,1, 1,1,1,1,1, 0,0,1,0,0, 1,0,0),
                             fuzzy_rule(0,1,0,1,0, 0,1,0,1,0, 0,0,1,0,0, 0,1,0)),
                      operator,
                      list(u[[1]]$w,u[[2]]$w,u[[3]]$w))

  y
}

t_frame <- function(y){

 templates <- c("It has been taken a bad framed photo",
                "It has been taken a middle framed photo",
                "It has been taken a good framed photo")

  return( templates[which.max(y$w)])
}

pm_frame <-  pm(y=cp_frame, g=g_frame, t=t_frame)
```

```
pm_report(pm_frame)
```

`report_template` *Define the report template*

Description

The text generation algorithm contains the programming code capable of generating the appropriate report to each specific user. Algorithms must select and order the linguistic expressions to generate the text included in the report. #' The `report_template` constructor receive as arguments the list of properties and the method (programming code) capable of generating the appropriate report.

Usage

```
report_template(properties = NULL, method, description = NULL)
```

Arguments

`properties`	may be a vector, list or matrix with the user's needs, preferences and goals. By default `properties = NULL`.
`method`	is the function that generates the appropriate report. The method must have two arguments: the list of properties and the list of pms: `my_report_method <- function(properties, pm){...}`.
`description`	is the result of call the report template. By default is NULL

Value

The generated `report_template= list(properties, method,description)`

Examples

```
properties = NULL
report_method <- function(properties,pm){
  pm_report(pm$pm_frame)
}
my_report <- report_template(properties,
                    report_method)
```

trapezoid_mf *Define the trapezoid membership function*

Description

It is a constructor of trapezoidal shapes. They take as input the numerical values which define the anchor points in membership functions.

Usage

```
trapezoid_mf(a, b, c, d)
```

Arguments

a	the trapezoid point a.
b	the trapezoid point b.
c	the trapezoid point c.
d	the trapezoid point d.

Value

the `(trapezoid_mf <- list(a,b,c,d))`

Examples

```
trapezoid_mf(0, 1, 2, 3)
```

triangle_mf *Define the triangle membership function*

Description

It is a constructor of triangular shapes. They take as input the numerical values which define the anchor points in membership functions.

Usage

```
triangle_mf(a, b, c)
```

Arguments

a	the trapezoid point a.
b	the trapezoid point b.
c	the trapezoid point c.

Value

the (triangle_mf <- list(a,b,c))

Examples

```
triangle_mf(0, 1, 2)
```

---

validate_xml *Validate the XML file*

Description

The function takes as input the path to a XML file that contains a LDCP system. Then it validates the LDCP system.

Usage

```
validate_xml(xmlfile, schema = NULL)
```

Arguments

xmlfile	is the XML source path file. E.g. "/folder/ldcp_system.xml".
schema	is the ldcp schema path file. By default is "ldcpSchema.xsd".

Value

If the process ends without error, the user will receive the message that indicates that the XML is valid. Otherwise, the user will receive the detailed list of errors.

Examples

```
## Not run: validate_xml('extdata/comfortableroom.xml')

## The xml is valid
```

`xml2rldcp` *XML to rLDCP*

Description

The function takes as input the path to a XML file that contains a LDCP system. Then it validates the LDCP system and generates its corresponding in R code. This R code is stored in an output file. The output file path is another function parameter.

Usage

```
xml2rldcp(input, output)
```

Arguments

`input`	is the XML source path file. E.g. "/folder/ldcp_system.xml".
`output`	is the R destination path file. E.g. "/folder/ldcp_system.R".

Value

If the process ends without error, the user will receive two messages: one indicates that the XML is valid and the other indicates that the code has been generated successfully. Otherwise, the user will receive the detailed list of errors.

Examples

```
## Not run: xml2rldcp('extdata/comfortableroom.xml','comfortableroom.R')

## The xml is valid
## The code has been generated successfully
```

Index