Package 'litc'

August 7, 2012

Title Procedures for Personality and Psychological Research

Version 1.0-7

Date 2006-06-15

Author William Revelle

Maintainer William Revelle <rxvalente@gmail.com>

Depends R (>= 2.10)

Description A number of routines for personality and experimental psychology, (This is my second draft of a package, so be patient,) Functions are primarily for scale construction and reliability analysis, although others are basic descriptive stats, For more information, see the personality-project.org/r

License GPL version 2 or newer

URL http://personality-project.org/r/

R topics documented:

`litc_package-package` *LITC package*

Description

More about what it does (maybe more than one line)

Details

Package:	litc_package
Type:	Package
Version:	1.0
Date:	2012-08-06
License:	What license is it under?
LazyLoad:	yes

Author(s)

aaabbb

References

~~ Literature or other references for background information ~~

See Also

~~ Optional links to other man pages, e.g. ~~ ~~ <pkg> ~~

`calculate_centers` *Calcula os centros*

Usage

```
calculate_centers(x, U, m)
```

Arguments

x
U
m

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(x,U,m){

  centers <- matrix(nrow=ncol(U),ncol=ncol(x))
  U_fuzzy <- U^m
  sum_clusters <- colSums(U_fuzzy)
  for(i in 1:nrow(centers)){
    fuzzy_factors <- U_fuzzy[,i]
    centers[i,] <- colSums(fuzzy_factors*x) / sum_clusters[i]
  }

  return(centers)
  }
```

`calculate_membership` *Calculate the Membership of the Patterns for a cluster.*

Description

Calculate Test!

Usage

```
calculate_membership(pattern, centers, m)
```

Arguments

`pattern`

`centers`

`m`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(pattern,centers,m){
  # Calculate the euclidian distance for all the centers
  membership <- matrix(nrow=1,ncol=nrow(centers));
  distances <- sqrt(rowSums((pattern - centers)^2))
  fuzzy_factor <- 2/(m-1)
  for(i in 1:ncol(membership)){
    factors <- distances[i] / distances
    membership[1,i] <- 1/sum(factors^(fuzzy_factor))
  }
  return(membership)
  }
```

`calculate_Uk` *Function that calculates blabla*

Description

Description of Calculate_UK

Usage

```
calculate_Uk(x, centers, m)
```

Arguments

`x`

`centers`

`m`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(x,centers,m){
  # For each pattern
  U <- matrix(nrow=nrow(x),ncol=nrow(centers))
  for(i in 1:nrow(x)){
    U[i,] <- calculate_membership(x[i,],centers,m)
  }
  return(U)
  }
```

---

fcm *FCM Funcion*

Usage

```
fcm(x, c, m = 2, iter.max = 100, tol = sqrt(.Machine$double.eps), centers)
```

Arguments

x

c

m

iter.max

tol

centers

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, c,
                 m = 2,
                 iter.max = 100,
                 tol = sqrt(.Machine$double.eps),
                 centers)
{

  x <- as.matrix(x)
  xrows <- nrow(x)
  xcols <- ncol(x)

  if (missing(c))
    stop("Number of clusters 'c' is not defined!")
```

```
if(missing(centers)){
  cmin <- range(x)[1]
  xmax <- range(x)[2]
  cvector <- runif(c*xcols)
  centers <- matrix(data=cvector,nrow=c,ncol=xcols)
}else{
  centers <- as.matrix(centers)
  crows <- nrow(centers)
  ccols <- ncol(centers)
  if(crows != c || ccols != xcols){
    stop("The center matrix must have 'c' rows and the 'n' columns!")
  }
}

initcenters <- centers
U <- matrix(0,nrow=nrow(x),ncol=nrow(centers))
# Algorithm starts here
for(i in 1:iter.max){
  Uk <- calculate_Uk(x,centers,m)
  centers <- calculate_centers(x,Uk,m)
  max_diff = max(abs(Uk - U))
  #print(max_diff)
  if(max_diff < tol){
    break;
  }
  U <- Uk
}
# Algorithm ends here

result = list(U=U,centers=centers,iter=i)
return(result)
}
```

Index