Package ‘supc’

November 27, 2019

Type Package
Title The Self-Updating Process Clustering Algorithms
Version 0.2.2
Maintainer Wush Wu <wush978@gmail.com>
URL https://github.com/wush978/supc
License GPL (>= 3)
LazyData TRUE
Depends R (>= 3.0.2)
Imports stats, Rcpp
Suggests amap, knitr, rmarkdown, fields, dbscan
LinkingTo Rcpp(>= 0.12), BH(>= 1.62)
VignetteBuilder knitr, rmarkdown
RoxygenNote 6.1.1
SystemRequirements C++11
Encoding UTF-8
NeedsCompilation yes
Author Wush Wu [aut, cre],
Shang-Ying Shiu [aut, ctb]
Repository CRAN
Date/Publication 2019-11-27 09:10:03 UTC

\textbf{R topics documented:}

\begin{verbatim}
D31 ................................................................. 2
dist.mode ....................................................... 2
dist.parallelization ......................................... 3
freq.poly ....................................................... 3
\end{verbatim}
The Artificial Data of Consisting of as Many as 31 Randomly Placed Gaussian Clusters

**Description**

This artificial data was generated to show the strength of SUPC. Clustering D31 dataset is difficult for the partition type of clustering algorithms that require an initial set. However, SUP correctly identifies the 31 major clusters.

**References**


---

**dist.mode**

Configure which package is used to compute the distance matrix

**Description**

Configure which package is used to compute the distance matrix or register one. Note that the speed depends on the data and the hardware.

**Usage**

```r
dist.mode(mode = c("stats", "amap"), FUN = NULL)
```

**Arguments**

- `mode` string. The available modes are "stats" and "amap" by default.
- `FUN` a function which has one argument x or NULL. The function should compute the pairwise distance of x and return a dist object. The user can skip this argument if the mode is registered. For example, "stats" and "amap" are registered by default.
Examples

# use stats::dist to compute the pairwise distance
dist.mode("stats")
## use gputools to compute the pairwise distance with GPU
# dist.mode("gputools", function(x) gputools::gpuDist(x, method = "euclidean", p = 2.0))

---

dist.parallelization  
*Configure how many cores will be used to calculate the distance matrix*

Description

Only affect *Dist*.

Usage

dist.parallelization(i)

Arguments

i  integer.

---

freq.poly  
*Plot the frequency polygon of pairwise distance*

Description

Plot the frequency polygon of the pairwise distance.

Usage

defreq.poly(x, ...)

Arguments

x  either dist object or matrix.

...  other parameters to be passed through to *hist*. 
freq.poly.supc  

Plot the frequency polygon of pairwise distance

Description

Plot the frequency polygon of the pairwise distance. The red dashed line is the used parameter \( r \).

Usage

## S3 method for class 'supc'
freq.poly(x, ...)

Arguments

x  either dist object or matrix.

... other parameters to be passed through to hist.

golub  Gene expression dataset from Golub et al. (1999)

Description

Gene expression data (3051 genes and 38 tumor mRNA samples) from the leukemia microarray study of Golub et al. (1999). Each row (gene) is scaled to mean 0 and standard deviation 1.

Value

golub  The matrix of scaled gene expression data.

golub.supc  The result of golub.supc <- supc1(golub, r = c(4, 4.3, 4.6, 4.7, 4.8), t = "dynamic")

References

\textit{plot.supc} \quad \textit{Draw plots of the clustering result}

\textbf{Description}

General function to draw plots for analysis

\textbf{Usage}

\begin{verbatim}
## S3 method for class 'supc'
plot(x, type = "heatmap", ...)
\end{verbatim}

\textbf{Arguments}

\begin{itemize}
\item \textbf{x} \quad \text{supc object to plot.}
\item \textbf{type} \quad \text{character value.}
\end{itemize}

\begin{itemize}
\item "heatmap" draw a heatmap to show the result of clustering. The clusters whose size is greater than parameter \texttt{major.size} are treated as major clusters.
\end{itemize}

\begin{itemize}
\item \texttt{...} \quad \text{other parameters to be passed through.}
\end{itemize}

\textbf{Examples}

\begin{verbatim}
## Not run:
data(golub, package = "supc")
golub.supc <- supc1(golub, rp = 0.0005, t = "dynamic")
table(golub.supc$size)
plot(golub.supc, type = "heatmap", major.size = 10)

## End(Not run)
\end{verbatim}

\textit{shape} \quad \textit{The Artificial Data of Five Different Clusters}

\textbf{Description}

This artificial data was generated to have five clusters: one big circle, two small circles, and two ellipses. It was to test if the clustering algorithm could identify and distinguish between the five different clusters or not. The dataset is generated from the following script:

\begin{verbatim}
makecircle <- function(N, seed) {
  n <- 0
  x <- NULL
  set.seed(seed)
  \texttt{...}
}
\end{verbatim}
while(n < N) {
    tmp <- runif(2, min = -1, max = 1)
    if (t(tmp) %*% tmp < 1) {
        n <- n + 1
        x <- rbind(x, tmp)
    }
}
return (x)
}

makedata <- function(n, seed) {
  f <- c(10, 3, 3, 1, 1)
  center <- matrix(
    c(-.3, -.3, -.55, .8, .55, .8, .9, 0, .9, -.6),
    nrow = 5, ncol = 2, byrow = TRUE
  )
  s <- matrix(
    c(.7, .7, .45, .2, .45, .2, .1, .1, .1, .1),
    nrow = 5, ncol = 2, byrow = TRUE
  )
  x <- NULL
  for (i in 1:5) {
    tmp <- makecircle(n * f[i], seed + i)
    tmp[,1] <- tmp[,1] * s[i,1] + center[i,1]
    tmp[,2] <- tmp[,2] * s[i,2] + center[i,2]
    x <- rbind(x, tmp)
  }
  line <- cbind(runif(floor(n / 3), min = -.1, max = .1), rep(.8, floor(n / 3)))
  noise <- matrix(runif(8 * n, min = -1, max = 1), nrow = 4 * n, ncol = 2)
  return(rbind(x, line, noise))
}

shape <- makedata(50, 1000)

References


supc.random

Randomized Self-Updating Process Clustering

Description

The Randomized Self-Updating Process Clustering (randomized SUP) is a modification of the original SUP algorithm. The randomized SUP randomly generates the partition of the instances during
each iterations. At each iteration, the self updating process is conducted independently in each partition in order to reduce the computation and the memory.

**Usage**

```r
supc.random(x, r = NULL, rp = NULL, t = c("static", "dynamic"),
            k = NULL, groups = NULL, tolerance = 1e-04,
            cluster.tolerance = 10 * tolerance, drop = TRUE,
            implementation = c("cpp", "R"), sort = TRUE, verbose = (nrow(x) >
            10000))
```

**Arguments**

- `x`: data matrix. Each row is an instance of the data.
- `r`: numeric vector or NULL. The parameter $r$ of the self-updating process.
- `rp`: numeric vector or NULL. If `r` is NULL, then `rp` will be used. The corresponding $r$ is the rp-percentile of the pairwise distances of the data. If both `r` and `rp` are NULL, then the default value is $r = c(0.0005, 0.001, 0.01, 0.1, 0.3)$.
- `t`: either numeric vector, list of function, or one of "static" or "dynamic". The parameter $T(t)$ of the self-updating process.
- `k`: integer value. The number of the partitions.
- `groups`: list. The first element is the partition of the first iteration, and the second element is the partition of the second iteration, etc. If the number of the iteration exceeds `length(groups)`, then new partition will be generated.
- `tolerance`: numeric value. The threshold of convergence.
- `cluster.tolerance`: numeric value. After iterations, if the distance of two points are smaller than `cluster.tolerance`, then they are identified as in the same cluster.
- `drop`: logical value. Whether to delete the list structure if its length is 1.
- `implementation`: either "R" or "cpp". Choose the engine to calculate result.
- `sort`: logical value. Whether to sort the cluster id by size.
- `verbose`: logical value. Whether to show the iteration history.

**Details**

Please check the vignettes via `vignette("supc", package = "supc")` for details.

**Value**

`supc1` returns a list of objects of class "supc".

Each "supc" object contains the following elements:

- `x`: The input matrix.
- `d0`: The pairwise distance matrix of `x`.
- `r`: The value of $r$ of the clustering.
t The function $T(t)$ of the clustering.
cluster The cluster id of each instance.
centers The center of each cluster.
size The size of each cluster.
iteration The number of iterations before convergence.
groups The partition of each iteration.
result The position of data after iterations.

References


Examples

```r
## Not run:
# The shape data has a structure of five clusters and a number of noise data points.
makecircle=function(N, seed){
  n=0
  x=matrix(NA, nrow=N, ncol=2)
  while (n<N){
    tmp=runif(2, min=0, max=1)*2-1
    if (sum(tmp^2)<1) {
      n=n+1
      x[n,]=tmp
    }
  }
  return(x)
}
makedata <- function(ns, seed) {
  size=c(10,3,3,1,1)
  mu=rbind(c(-0.3, -0.3), c(-0.55, 0.8), c(0.55, 0.8), c(0.9, 0), c(0.9, -0.6))
  sd=rbind(c(0.7, 0.7), c(0.45, 0.2), c(0.45, 0.2), c(0.1, 0.1), c(0.1, 0.1))
  x=NULL
  for (i in 1:5){
    tmp=makecircle(ns*size[i], seed+i)
    tmp[,1]=tmp[,1]*sd[i,1]+mu[i,1]
    tmp[,2]=tmp[,2]*sd[i,2]+mu[i,2]
    x=rbind(x, tmp)
  }
  tmp=runif(floor(ns/3), min=0, max=1)/5-0.1
  tmp=cbind(tmp, 0.8*rep(1, floor(ns/3)))
  x=rbind(x, tmp)
  x=rbind(x, matrix(1, nrow=2*ns, ncol=2)*2-1)
  return(x)
}
```
supc1 <- makedata(5000, 1000)
dim(shape1)
plot(shape1)

X.supc=supc.random(shape1, r=0.5, t="dynamic", k = 500)
plot(shape1, col=X.supc$cluster)

## End(Not run)

---

**supc1**

*Self-Updating Process Clustering*

### Description

The SUP is a distance-based method for clustering. The idea of this algorithm is similar to gravitational attraction: every sample gravitates towards one another. The algorithm mimics the process of gravitational attraction iteratively that eventually merges the samples into clusters on the sample space. During the iterations, all samples continue moving until the system becomes stable.

### Usage

```r
supc1(x, r = NULL, rp = NULL, t = c("static", "dynamic"),
       tolerance = 1e-04, cluster.tolerance = 10 * tolerance, drop = TRUE,
       implementation = c("cpp", "R", "cpp2"), sort = TRUE,
       verbose = (nrow(x) > 10000))
```

### Arguments

- **x**: data matrix. Each row is an instance of the data.
- **r**: numeric vector or NULL. The parameter $r$ of the self-updating process.
- **rp**: numeric vector or NULL. If $r$ is NULL, then $rp$ will be used. The corresponding $r$ is the $rp$-percentile of the pairwise distances of the data. If both $r$ and $rp$ are NULL, then the default value is $rp = c(0.0005, 0.001, 0.01, 0.1, 0.3)$.
- **t**: either numeric vector, list of function, or one of "static" or "dynamic". The parameter $T(t)$ of the self-updating process.
- **tolerance**: numeric value. The threshold of convergence.
- **cluster.tolerance**: numeric value. After iterations, if the distance of two points are smaller than cluster.tolerance, then they are identified as in the same cluster.
- **drop**: logical value. Whether to delete the list structure if its length is 1.
- **implementation**: either "R", "cpp" or "cpp2". Choose the engine to calculate result. The "cpp2" parallelly computes the distance in C++ with OpenMP, which is not supported under OS X, and uses the early-stop to speed up calculation.
- **sort**: logical value. Whether to sort the cluster id by size.
- **verbose**: logical value. Whether to show the iteration history.
Details

Please check the vignettes via vignette("supc", package = "supc") for details.

Value

`supc1` returns a list of objects of class "supc".

Each "supc" object contains the following elements:

- **x** The input matrix.
- **d0** The pairwise distance matrix of x or NULL.
- **r** The value of r of the clustering.
- **t** The function $T(t)$ of the clustering.
- **cluster** The cluster id of each instance.
- **centers** The center of each cluster.
- **size** The size of each cluster.
- **iteration** The number of iterations before convergence.
- **result** The position of data after iterations.

References


Examples

```r
## Not run:
set.seed(1)
X <- local({
  mu <- list(
    x = c(0, 2, 1, 6, 7, 3, 5, 4),
    y = c(0, 1, 0, 0, 1, 3, 3, 4)
  )
  X <- lapply(1:5, function(i) {
    cbind(rnorm(9, mu$x, 1/5), rnorm(9, mu$y, 1/5))
  })
  X <- do.call(rbind, X)
  n <- nrow(X)
  X <- rbind(X, matrix(0, 20, 2))
  k <- 1
  while(k <= 20) {
    tmp <- c(13*runif(1)-2.5, 8*runif(1)-2.5)
    y1 <- mu$x - tmp[1]
    y2 <- mu$y - tmp[2]
    y <- sqrt(y1^2+y2^2)
    if (min(y) > 2){
      X[k+n,] <- tmp
      k <- k+1
    }
  }
})
```
X.supcs <- supcl(X, r = c(0.9, 1.7, 2.5), t = "dynamic")
X.supcs$cluster
plot(X.supcs[[1]], type = "heatmap", major.size = 2)
plot(X.supcs[[2]], type = "heatmap", col = cm.colors(24), major.size = 5)

X.supcs <- supcl(X, r = c(1.7, 2.5), t = list("f(t) = 1.7 / 20 + \exp(t) * (1.7 / 50)",
function(t) {\exp(t)})
)
plot(X.supcs[[1]], type = "heatmap", major.size = 2)
plot(X.supcs[[2]], type = "heatmap", col = cm.colors(24), major.size = 5)

## End(Not run)
Index

class, 7, 10
D31, 2
Dist, 3
dist.mode, 2
dist.parallelization, 3
freq.poly, 3
freq.poly.subclist (freq.poly.supc), 4
freq.poly.supc, 4
golub, 4
hist, 3, 4
plot.supc, 5
shape, 5
supc.random, 6
supc1, 9