Title  Efficient Implementation of K-Means++ Algorithm

Version  0.3.10

Author  Aviezer Lifshitz [aut, cre],
        Amos Tanay [aut],
        Weizmann Institute of Science [cph]

Maintainer  Aviezer Lifshitz <aviezer.lifshitz@weizmann.ac.il>


License  MIT + file LICENSE

BugReports  https://github.com/tanaylab/tglkmeans/issues

OS_type  unix

Depends  R (>= 4.0.0)

Imports  Rcpp (>= 0.12.11), doFuture, future, dplyr (>= 0.5.0), ggplot2
         (>= 2.2.0), magrittr, tibble (>= 3.2.1), parallel (>= 3.3.2),
         plyr (>= 1.8.4), purrr (>= 0.2.0), tgstat (>= 1.0.0)

Suggests  covr, knitr, rlang, rmarkdown, testthat, withr

LinkingTo  Rcpp

VignetteBuilder  knitr

Encoding  UTF-8

NeedsCompilation  yes

RoxygenNote  7.2.3

Repository  CRAN

Date/Publication  2023-06-26 08:30:02 UTC
simulate_data

**R topics documented:**

- simulate_data ............................................. 2
- tglkmeans ................................................. 3
- tglkmeans.set_parallel ................................. 3
- TGL_kmeans ............................................... 4
- TGL_kmeans_tidy ......................................... 5

**Index**

<table>
<thead>
<tr>
<th>simulate_data</th>
<th>Simulate normal data for kmeans tests</th>
</tr>
</thead>
</table>

**Description**

Creates `nclust` clusters normally distributed around 1:`nclust`

**Usage**

```r
simulate_data(
  n = 100,
  sd = 0.3,
  nclust = 30,
  dims = 2,
  frac_na = NULL,
  add_true_clust = TRUE
)
```

**Arguments**

- `n` number of observations per cluster
- `sd` sd
- `nclust` number of clusters
- `dims` number of dimensions
- `frac_na` fraction of NA in the first dimension
- `add_true_clust` add a column with the true cluster ids

**Value**

simulated data

**Examples**

```r
simulate_data(n = 100, sd = 0.3, nclust = 5, dims = 2)
# add 20% missing data
simulate_data(n = 100, sd = 0.3, nclust = 5, dims = 2, frac_na = 0.2)
```
Description
tglkmeans package

Usage
tglkmeans.set_parallel(thread_num)

Arguments
  thread_num number of threads. use ‘1’ for non parallel behavior

Value
  None

Examples
tglkmeans.set_parallel(8)
TGL_kmeans

**kmeans++ with return value similar to R kmeans**

**Description**

kmeans++ with return value similar to R kmeans

**Usage**

```r
TGL_kmeans(
  df,
  k,
  metric = "euclid",
  max_iter = 40,
  min_delta = 1e-04,
  verbose = FALSE,
  keep_log = FALSE,
  id_column = TRUE,
  reorder_func = "hclust",
  hclust_intra_clusters = FALSE,
  seed = NULL,
  parallel = getOption("tglkmeans.parallel")
)
```

**Arguments**

- **df**
  - data frame. Each row is a single observation and each column is a dimension. the first column can contain id for each observation (if id_column is TRUE).
- **k**
  - number of clusters. Note that in some cases the algorithm might return less clusters than k.
- **metric**
  - distance metric for kmeans++ seeding. can be 'euclid', 'pearson' or 'spearman'
- **max_iter**
  - maximal number of iterations
- **min_delta**
  - minimal change in assignments (fraction out of all observations) to continue iterating
- **verbose**
  - display algorithm messages
- **keep_log**
  - keep algorithm messages in 'log' field
- **id_column**
  - df’s first column contains the observation id
- **reorder_func**
  - function to reorder the clusters. operates on each center and orders by the result. e.g. reorder_func = mean would calculate the mean of each center and then would reorder the clusters accordingly. If reorder_func = hclust the centers would be ordered by hclust of the euclidean distance of the correlation matrix, i.e. hclust(dist(cor(t(centers)))) if NULL, no reordering would be done.
- **hclust_intra_clusters**
  - run hierarchical clustering within each cluster and return an ordering of the observations.
TGL_kmeans_tidy

seed  seed for the c++ random number generator
parallel  cluster every cluster parallelly (if hclust_intra_clusters is true)

Value

list with the following components:

cluster: A vector of integers (from ‘1:k’) indicating the cluster to which each point is allocated.
centers: A matrix of cluster centers.
size: The number of points in each cluster.
log: messages from the algorithm run (only if id_column == TRUE).
order: A vector of integers with the new ordering if the observations. (only if hclust_intra_clusters = TRUE)

See Also
TGL_kmeans_tidy

Examples

# create 5 clusters normally distributed around 1:5
d <- simulate_data(n = 100, sd = 0.3, nclust = 5, dims = 2, add_true_clust = FALSE)
head(d)

# cluster
km <- TGL_kmeans(d, k = 5, "euclid", verbose = TRUE)
names(km)
k$m$centers
head(k$m$cluster)
k$m$size
min_delta = 1e-04,
verbose = FALSE,
keep_log = FALSE,
id_column = TRUE,
reorder_func = "hclust",
add_to_data = FALSE,
hclust_intra_clusters = FALSE,
seed = NULL,
parallel = getOption("tglkmeans.parallel")
)

Arguments

- **df**: data frame. Each row is a single observation and each column is a dimension. The first column can contain id for each observation (if id_column is TRUE).
- **k**: number of clusters. Note that in some cases the algorithm might return less clusters than k.
- **metric**: distance metric for kmeans++ seeding. Can be 'euclid', 'pearson' or 'spearman'
- **max_iter**: maximal number of iterations
- **min_delta**: minimal change in assignments (fraction out of all observations) to continue iterating
- **verbose**: display algorithm messages
- **keep_log**: keep algorithm messages in 'log' field
- **id_column**: df’s first column contains the observation id
- **reorder_func**: function to reorder the clusters. Operates on each center and orders by the result. E.g. reorder_func = mean would calculate the mean of each center and then would reorder the clusters accordingly. If reorder_func = hclust the centers would be ordered by hclust of the euclidean distance of the correlation matrix, i.e. hclust(dist(cor(t(centers)))) if NULL, no reordering would be done.
- **add_to_data**: return also the original data frame with an extra 'clust' column with the cluster ids ('id' is the first column)
- **hclust_intra_clusters**: run hierarchical clustering within each cluster and return an ordering of the observations.
- **seed**: seed for the c++ random number generator
- **parallel**: cluster every cluster parallelly (if hclust_intra_clusters is true)

Value

List with the following components:

- **cluster**: tibble with 'id' column with the observation id ('1:n' if no id column was supplied), and 'clust' column with the observation assigned cluster.
- **centers**: tibble with 'clust' column and the cluster centers.
- **size**: tibble with 'clust' column and 'n' column with the number of points in each cluster.
**TGL_kmeans_tidy**

- **data**: tibble with 'clust' column the original data frame.
- **log**: messages from the algorithm run (only if id_column = TRUE).
- **order**: tibble with 'id' column, 'clust' column, 'order' column with a new ordering if the observations and 'intra_clust_order' column with the order within each cluster. (only if hclust_intra_clusters = TRUE)

**See Also**

TGL_kmeans

**Examples**

```r
# create 5 clusters normally distributed around 1:5
d <- simulate_data(n = 100, sd = 0.3, nclust = 5, dims = 2, add_true_clust = FALSE)
head(d)

# cluster
km <- TGL_kmeans_tidy(d, k = 5, "euclid", verbose = TRUE)
km
```
Index

simulate_data, 2
TGL_kmeans, 4, 7
TGL_kmeans_tidy, 5, 5
tglkmeans, 3
tglkmeans.set_parallel, 3