Package 'tglkmeans'

May 15, 2024

Title Efficient Implementation of K-Means++ Algorithm

Version 0.5.5

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

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

Description Efficient implementation of K-Means++ algorithm. For more information see (1) ``kmeans++ the advantages of the k-means++ algorithm" by David Arthur and Sergei Vassilvitskii (2007), Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 1027-1035, and (2) ``The Effectiveness of Lloyd-Type Methods for the k-Means Problem" by Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman and Chaitanya Swamy <doi:10.1145/2395116.2395117>.

License MIT + file LICENSE

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

URL https://tanaylab.github.io/tglkmeans/,
https://github.com/tanaylab/tglkmeans

Depends R (>= 4.0.0)

Imports cli, doFuture, doRNG, dplyr (>= 0.5.0), future, ggplot2 (>= 2.2.0), magrittr, Matrix, methods, parallel (>= 3.3.2), plyr (>= 1.8.4), purrr (>= 0.2.0), Rcpp (>= 0.12.11), RcppParallel, tgstat (>= 1.0.0), tibble (>= 3.1.2)

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

LinkingTo Rcpp, RcppParallel

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel false

Encoding UTF-8

NeedsCompilation yes

OS_type unix

RoxygenNote 7.3.1

SystemRequirements GNU make

Repository CRAN

Date/Publication 2024-05-15 08:40:02 UTC

R topics documented:

`downsample_matrix` *Downsample the columns of a matrix to a target number*

Description

This function takes a matrix and downsamples it to a target number of samples. It uses a random seed for reproducibility and allows for removing columns with small sums.

Usage

```
downsample_matrix(
  mat,
  target_n = NULL,
  target_q = NULL,
  seed = NULL,
  remove_columns = FALSE
)
```

Arguments

`mat`	An integer matrix to be downsampled. Can be a matrix or sparse matrix (dgCMatrix). If the matrix contains NAs, the function will run significantly slower. Values that are not integers will be coerced to integers using `floor()`.
`target_n`	The target number of samples to downsample to.
`target_q`	A target quantile of sums to downsample to. Only one of 'target_n' or 'target_q' can be provided.
`seed`	The random seed for reproducibility (default is NULL)
`remove_columns`	Logical indicating whether to remove columns with small sums (default is FALSE)

Value

The downsampled matrix

Examples

```
mat <- matrix(1:12, nrow = 4)
downsample_matrix(mat, 2)

# Remove columns with small sums
downsample_matrix(mat, 12, remove_columns = TRUE)

# sparse matrix
mat_sparse <- Matrix::Matrix(mat, sparse = TRUE)
downsample_matrix(mat_sparse, 2)

# with a quantile
downsample_matrix(mat, target_q = 0.5)
```

---

simulate_data *Simulate normal data for kmeans tests*

Description

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

Usage

```
simulate_data(
  n = 100,
  sd = 0.3,
  nclust = 30,
  dims = 2,
  frac_na = NULL,
  add_true_clust = TRUE,
  id_column = 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
id_column	add a column with the id

Value

simulated data

Examples

```
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)
```

---

tglkmeans.set_parallel

Set parallel threads

Description

Set parallel threads

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

```
TGL_kmeans(
  df,
  k,
  metric = "euclid",
  max_iter = 40,
  min_delta = 0.0001,
  verbose = FALSE,
  keep_log = FALSE,
  id_column = FALSE,
  reorder_func = "hclust",
  hclust_intra_clusters = FALSE,
  seed = NULL,
  parallel = getOption("tglkmeans.parallel"),
  use_cpp_random = FALSE
)
```

Arguments

df	a data frame or a matrix. 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), otherwise the rownames are used.
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.

`seed`
: seed for the c++ random number generator

`parallel`
: cluster every cluster parallelly (if hclust_intra_clusters is true)

`use_cpp_random`
: use c++ random number generator instead of R's. This should be used for only for backwards compatibility, as from version 0.4.0 onwards the default random number generator was changed o R.

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,
    id_column = FALSE
)

head(d)

# cluster
km <- TGL_kmeans(d, k = 5, "euclid", verbose = TRUE)
names(km)
km$centers
head(km$cluster)
km$size
```

TGL_kmeans_tidy *TGL kmeans with 'tidy' output*

Description

TGL kmeans with 'tidy' output

Usage

```
TGL_kmeans_tidy(
  df,
  k,
  metric = "euclid",
  max_iter = 40,
  min_delta = 0.0001,
  verbose = FALSE,
  keep_log = FALSE,
  id_column = FALSE,
  reorder_func = "hclust",
  add_to_data = FALSE,
  hclust_intra_clusters = FALSE,
  seed = NULL,
  parallel = getOption("tglkmeans.parallel"),
  use_cpp_random = FALSE
)
```

Arguments

df	a data frame or a matrix. 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), otherwise the rownames are used.
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)
use_cpp_random	use c++ random number generator instead of R's. This should be used for only for backwards compatibility, as from version 0.4.0 onwards the default random number generator was changed o R.

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.

data: tibble with 'clust' column the original data frame.

log: messages from the algorithm run (only if `id_column = FALSE`).

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

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

head(d)

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

Index