

# Package ‘tglkmeans’

April 20, 2022

**Title** Efficient Implementation of K-Means++ Algorithm

**Version** 0.3.4

**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, <<http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf>>, 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](https://doi.org/10.1145/2395116.2395117)>.

**License** MIT + file LICENSE

**BugReports** <https://github.com/tanaylab/tglkmeans/issues>

**OS\_type** unix

**Depends** R (>= 3.2.4)

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

**Suggests** covr, knitr, rlang, rmarkdown, testthat

**LinkingTo** Rcpp

**VignetteBuilder** knitr

**Encoding** UTF-8

**NeedsCompilation** yes

**RoxygenNote** 7.1.2

**SystemRequirements** C++11

**Repository** CRAN

**Date/Publication** 2022-04-20 08:40:02 UTC

## R topics documented:

simulate_data	2
tgkmeans	3
tgkmeans.set_parallel	3
TGL_kmeans	4
TGL_kmeans_tidy	5

<b>Index</b>	<b>8</b>
--------------	----------

---

simulate_data	<i>Simulate normal data for kmeans tests</i>
---------------	--

---

### 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
)
```

### 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

```
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)
```

---

tgkmeans	<i>tgkmeans</i>
----------	-----------------

---

### **Description**

tgkmeans package

---

tgkmeans.set_parallel	<i>Set parallel threads</i>
-----------------------	-----------------------------

---

### **Description**

Set parallel threads

### **Usage**

```
tgkmeans.set_parallel(thread_num)
```

### **Arguments**

thread\_num      number of threads. use '1' for non parallel behavior

### **Value**

None

### **Examples**

```
tgkmeans.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 = 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
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)

**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)
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 = TRUE,
```

```

reorder_func = "hclust",
add_to_data = FALSE,
hclust_intra_clusters = FALSE,
seed = NULL,
parallel = getOption("tglkmeans.parallel")
)

```

### Arguments

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

**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**

```
# 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