

# Package ‘leiden’

September 10, 2022

**Type** Package

**Title** R Implementation of Leiden Clustering Algorithm

**Version** 0.4.3

**Date** 2022-09-10

**Description** Implements the 'Python leidenalg' module to be called in R.

Enables clustering using the leiden algorithm for partition a graph into communities.

See the 'Python' repository for more details: <<https://github.com/vtraag/leidenalg>>

Traag et al (2018) From Louvain to Leiden: guaranteeing well-connected communities. <[arXiv:1810.08473](https://arxiv.org/abs/1810.08473)>.

**License** GPL-3 | file LICENSE

**URL** <https://github.com/TomKellyGenetics/leiden>

**BugReports** <https://github.com/TomKellyGenetics/leiden/issues>

**Imports** methods, reticulate, Matrix, igraph (>= 1.2.7)

**Encoding** UTF-8

**RoxygenNote** 7.1.2

**Suggests** bipartite, covr, data.table, devtools, graphsim, knitr,

markdown, multiplex, multinet, network, RColorBrewer, remotes,

rmarkdown, spelling, testthat, tibble

**Language** en-US

**VignetteBuilder** knitr

**Collate** 'find\_partition.R' 'leiden.R' 'py\_objects.R'

**NeedsCompilation** no

**Author** S. Thomas Kelly [aut, cre, trl],

Vincent A. Traag [com]

**Maintainer** S. Thomas Kelly <[tomkellygenetics@gmail.com](mailto:tomkellygenetics@gmail.com)>

**Repository** CRAN

**Date/Publication** 2022-09-10 17:22:53 UTC

## R topics documented:

leiden . . . . . 2

**Index** . . . . . 5

leiden *Run Leiden clustering algorithm*

### Description

Implements the Leiden clustering algorithm in R using reticulate to run the Python version. Requires the python "leidenalg" and "igraph" modules to be installed. Returns a vector of partition indices. Windows users can still this with `devtools::install_github("rstudio/reticulate", ref = "86ebb56"); reticulate::use_condaenv("r-reticulate"); reticulate::conda_install("r-reticulate", "leidenalg", channel = "vtraag")`

### Usage

```
leiden(
  object,
  partition_type = c("RBConfigurationVertexPartition", "ModularityVertexPartition",
    "RBERVertexPartition", "CPMVertexPartition", "MutableVertexPartition",
    "SignificanceVertexPartition", "SurpriseVertexPartition",
    "ModularityVertexPartition.Bipartite", "CPMVertexPartition.Bipartite"),
  initial_membership = NULL,
  weights = NULL,
  node_sizes = NULL,
  resolution_parameter = 1,
  seed = NULL,
  n_iterations = 2L,
  max_comm_size = 0L,
  degree_as_node_size = FALSE,
  laplacian = FALSE,
  legacy = FALSE
)
```

### Arguments

**object** An adjacency matrix compatible with [igraph](#) object or an input graph as an [igraph](#) object (e.g., shared nearest neighbours). A list of multiple graph objects can be passed for multiplex community detection.

**partition\_type** Type of partition to use. Defaults to `RBConfigurationVertexPartition`. Options include: `ModularityVertexPartition`, `RBERVertexPartition`, `CPMVertexPartition`, `MutableVertexPartition`, `SignificanceVertexPartition`, `SurpriseVertexPartition`, `ModularityVertexPartition.Bipartite`, `CPMVertexPartition.Bipartite` (see the Leiden python module documentation for more details)

<code>initial_membership, weights, node_sizes</code>	Parameters to pass to the Python <code>leidenalg</code> function (defaults <code>initial_membership=None</code> , <code>weights=None</code> ). Weights are derived from weighted <code>igraph</code> objects and non-zero integer values of adjacency matrices.
<code>resolution_parameter</code>	A parameter controlling the coarseness of the clusters
<code>seed</code>	Seed for the random number generator. By default uses a random seed if nothing is specified.
<code>n_iterations</code>	Number of iterations to run the Leiden algorithm. By default, 2 iterations are run. If the number of iterations is negative, the Leiden algorithm is run until an iteration in which there was no improvement.
<code>max_comm_size</code>	(non-negative int) – Maximal total size of nodes in a community. If zero (the default), then communities can be of any size.
<code>degree_as_node_size</code>	(defaults to <code>FALSE</code> ). If <code>True</code> use degree as node size instead of 1, to mimic modularity for Bipartite graphs.
<code>laplacian</code>	(defaults to <code>FALSE</code> ). Derive edge weights from the Laplacian matrix.
<code>legacy</code>	(defaults to <code>FALSE</code> ). Force calling python implementation via <code>reticulate</code> . Default behaviour is calling <code>cluster_leiden</code> in <code>igraph</code> with Modularity (for undirected graphs) and CPM cost functions.

## Value

A partition of clusters as a vector of integers

## Examples

```
#check if python is available
modules <- reticulate::py_module_available("leidenalg") && reticulate::py_module_available("igraph")
if(modules){
#generate example data
adjacency_matrix <- rbind(cbind(matrix(round(rbinom(4000, 1, 0.8)), 20, 20),
                                matrix(round(rbinom(4000, 1, 0.3)), 20, 20),
                                matrix(round(rbinom(400, 1, 0.1)), 20, 20)),
                          cbind(matrix(round(rbinom(400, 1, 0.3)), 20, 20),
                                matrix(round(rbinom(400, 1, 0.8)), 20, 20),
                                matrix(round(rbinom(4000, 1, 0.2)), 20, 20)),
                          cbind(matrix(round(rbinom(400, 1, 0.3)), 20, 20),
                                matrix(round(rbinom(4000, 1, 0.1)), 20, 20),
                                matrix(round(rbinom(4000, 1, 0.9)), 20, 20)))

rownames(adjacency_matrix) <- 1:60
colnames(adjacency_matrix) <- 1:60
#generate partitions
partition <- leiden(adjacency_matrix)
table(partition)

#generate partitions at a lower resolution
partition <- leiden(adjacency_matrix, resolution_parameter = 0.5)
table(partition)
```

```
#generate example weights
weights <- sample(1:10, sum(adjacency_matrix!=0), replace=TRUE)
partition <- leiden(adjacency_matrix, weights = weights)
table(partition)

#generate example weighted matrix
adjacency_matrix[adjacency_matrix == 1] <- weights
partition <- leiden(adjacency_matrix)
table(partition)

# generate (unweighted) igraph object in R
library("igraph")
adjacency_matrix[adjacency_matrix > 1] <- 1
my_graph <- graph_from_adjacency_matrix(adjacency_matrix)
partition <- leiden(my_graph)
table(partition)

# generate (weighted) igraph object in R
library("igraph")
adjacency_matrix[adjacency_matrix >= 1] <- weights
my_graph <- graph_from_adjacency_matrix(adjacency_matrix, weighted = TRUE)
partition <- leiden(my_graph)
table(partition)

# pass weights to python leidenalg
adjacency_matrix[adjacency_matrix >= 1 ] <- 1
my_graph <- graph_from_adjacency_matrix(adjacency_matrix, weighted = NULL)
weights <- sample(1:10, sum(adjacency_matrix!=0), replace=TRUE)
partition <- leiden(my_graph, weights = weights)
table(partition)

# run only if python is available (for testing)
}
```

# Index

- \* **graph**
    - leiden, [2](#)
  - \* **igraph**
    - leiden, [2](#)
  - \* **mvtnorm**
    - leiden, [2](#)
  - \* **network**
    - leiden, [2](#)
  - \* **simulation**
    - leiden, [2](#)
- [igraph](#), [2](#)
- [leiden](#), [2](#)