Package ‘leiden’

January 26, 2021

Type Package

Title R Implementation of Leiden Clustering Algorithm

Version 0.3.7

Date 2021-01-26

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>


License GPL-3 | file LICENSE

URL https://github.com/TomKellyGenetics/leiden

Imports methods, reticulate, Matrix, igraph

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Suggests bipartite, covr, data.table, devtools, graphsim, knitr, multiplex, multinet, markdown, network, RColorBrewer, markdown, 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 <tom.kelly@riken.jp>

Repository CRAN

Date/Publication 2021-01-26 16:00:02 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,
  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
)

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)
leiden

initial_membership, weights, node_sizes
Parameters to pass to the Python leidenalg function (defaults initial_membership=None, weights=None). Weights are derived from weighted igraph objects and non-zero integer values of adjacency matrices.

resolution_parameter
A parameter controlling the coarseness of the clusters

seed
Seed for the random number generator. By default uses a random seed if nothing is specified.

n_iterations
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.

max_comm_size
(non-negative int) – Maximal total size of nodes in a community. If zero (the default), then communities can be of any size.

degree_as_node_size
(defaults to FALSE). If True use degree as node size instead of 1, to mimic modularity for Bipartite graphs.

degree as node size

laplacian
(defaults to FALSE). Derive edge weights from the Laplacian matrix.

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(400, 1, 0.9)), 20, 20),
                               matrix(round(rbinom(400, 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