Package ‘gama’
February 26, 2019

Type Package
Title Genetic Approach to Maximize Clustering Criterion
Version 1.0.3
Date 2019-02-15
Maintainer Jairson Rodrigues <jairson.rodrigues@univasf.edu.br>

Description An evolutionary approach to performing hard partitional clustering. The algorithm uses genetic operators guided by information about the quality of individual partitions. The method looks for the best barycenters/centroids configuration (encoded as real-value) to maximize or minimize one of the given clustering validation criteria: Silhouette, Dunn Index, C-Index or Calinski-Harabasz Index. As many other clustering algorithms, 'gama' asks for k: a fixed a priori established number of partitions. If the user does not know the best value for k, the algorithm estimates it by using one of two user-specified options: minimum or broad. The first method uses an approximation of the second derivative of a set of points to automatically detect the maximum curvature (the 'elbow') in the within-cluster sum of squares error (WCSSE) graph. The second method estimates the best k value through majority voting of 24 indices. One of the major advantages of 'gama' is to introduce a bias to detect partitions which attend a particular criterion. References: Scrucca, L. (2013) <doi:10.18637/jss.v053.i04>; CHARRAD, Malik et al. (2014) <doi:10.18637/jss.v061.i06>; Tsagris M, Papadakis M. (2018) <doi:10.7287/peerj.preprints.26605v1>; Kaufman, L., & Rousseeuw, P. (1990, ISBN:0-471-73578-7).

Imports ArgumentCheck, cluster, clusterCrit, NbClust, GA, ggplot2, methods, Rfast
Suggests knitr, rmarkdown

VignetteBuilder knitr
License GPL (>= 2)
LazyData yes

URL https://github.com/jairsonrodrigues/gama

NeedsCompilation no

Author Jairson Rodrigues [aut, cre]  (https://orcid.org/0000-0003-1176-3903),
Germano Vasconcelos [aut, ths]
(https://orcid.org/0000-0002-1899-1506),
Renato Tin’o’s [aut, rev] (https://orcid.org/0000-0003-4027-8851)
aggregation

Description

This is a synthetic dataset that contains features that are known to create difficulties for the selected algorithms such as, narrow bridges between clusters, uneven-sized clusters, etc. See references, for details.

Usage

data(aggregation)

Format

A data frame containing 788 observations and two dimensions, forming seven partitions:

1. x1: synthetically generated real positive values
2. x2: synthetically generated real positive values

Originally, the dataset had contained three dimensions. We intentionally removed the third dimension that corresponds to the label which the data point belongs. All description about the data set may be found in Clustering Aggregation article, in the references.

Source

The dataset was collected from Clustering basic benchmark site.
References


Description

This is a synthetic dataset that contains groups of different density points, varied shapes, and necks between partitions.

Usage

data(compound)

Format

A data frame containing 399 observations and two dimensions, forming six partitions:

1. x1: synthetically generated real positive values
2. x2: synthetically generated real positive values

Originally, the dataset had contained three dimensions. We intentionally removed the third dimension that corresponds to the label which the data point belongs. All description about the data set may be found in Graph-theoretical methods for detecting and describing gestalt clusters article, in the references.

Source

The dataset was collected from Clustering basic benchmark site.

References


**cpu.als**

*CPU usage metrics for distributed ALS algorithm*

---

**Description**

Consumption metrics gathered during an execution of the Distributed Machine Learning algorithm Alternating Least Squares (ALS) in an eighth-node cluster, by using the Spark framework.

**Usage**

cpu.als

**Format**

A data frame containing 308 observations and four dimensions:

1. user: CPU usage by the algorithm
2. system: CPU usage spent by Operating System (O.S.)
3. iowait: waiting time for Input/Output (I/O) operations
4. softirq: CPU time spent by software interrupt requests

The values comprise the domain from 0 to 100, for all dimensions. The dataset contains zero-values, however there is no missing or null values.

** Source **

A spark cluster of N nodes has 1 (one) master node and N-1 slave nodes.

** Source **

The data was measured and collected by the author by using Intel HiBench benchmark framework in an eighth-node Spark cluster hosted in Google Cloud DataProc engine. Each node had 16-core CPU annd 106 GB RAM. The algorithm Alternating Least Squares had consumed 3.7 min of runtime to execute over a sinteticaly generated dataset totalizing 1.68 Gigabytes.

** References **


**CPU usage metrics for distributed PCA algorithm**

**Description**

Consumption metrics gathered during an execution of the Distributed Machine Learning algorithm Principal Component Analysis (PCA) in an eighth-node cluster, by using the Spark framework.

**Usage**

cpu.pca

**Format**

A data frame containing 938 observations and four dimensions:

1. user: CPU usage by the algorithm
2. system: CPU usage spent by Operating System (O.S.)
3. iowait: waiting time for Input/Output (I/O) operations
4. softirq: CPU time spent by software interrupt requests

The values comprise the domain from 0 to 100, for all dimensions. The dataset contains zero-values, however there is no missing or null values.

** Source **

The data was measured and collected by the author by using Intel HiBench benchmark framework in an eighth-node Spark cluster hosted in Google Cloud DataProc engine. Each node had 16-core CPU and 106 GB RAM. The algorithm Principal Component Analysis had consumed 11.2 min of runtime to execute over a sintetically generated dataset totalizing 1.68 Gigabytes.

** References **


Description

All description about the data set may be found in FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data article, and the Clustering basic benchmark site, in the references.

Usage

data(flame)

Format

A data frame containing 240 observations and two dimensions, forming two partitions:

1. x1: real positive values
2. x2: real positive values

Originally, the dataset had contained three dimensions. We intentionally removed the third dimension that corresponds to the label which the data point belongs.

Source

The dataset was collected from Clustering basic benchmark site.

References


Description

This function realizes a genetic search in order to segment a given dataset. The genetic algorithm evaluates individuals of a population to maximize (or minimize) one of four user-specified criteria: Average Silhouette Width, Calinski-Harabasz index, C-index or Dunn index. The algorithm receives the optimal number of partitions (best k value) from the user or automatically suggests it.
Usage

gama(dataset = NULL, k = "broad", scale = FALSE, crossover.rate = 0.9, mutation.rate = 0.01, elitism = 0.05, pop.size = 25, generations = 100, seed.p = 42, fitness.criterion = "ASW", penalty.function = NULL, plot.internals = TRUE, ...)

Arguments

dataset the data to segment into k partitions. Must be numerical, because GAMA only works for integer or real values.

k the best value for the optimal number of partitions. May be:

1. an integer positive value: when known, the user may specify directly the best value for k;
2. a string value: 'minimal', for estimation based on 'elbow' in Within-cluster Sum of Squares Error graph or 'broad', the alternative to estimate the best k by majority voting between 24 distinct internal validation criteria for automatically estimate the best k value for the data.

The user may omit this argument, in this way, the default value 'broad' will be used.

scale if the dataset will be scaled and normalized before segmentation. The default value is FALSE.

crossover.rate the probability of crossover between pairs of chromosomes. The default value is 0.9.

mutation.rate the probability of mutation in a parent chromosome. Usually mutation occurs with a small probability. The default value is 0.01.

elitism the number of best individuals to survive. The default value is the top 5% individuals of population.

pop.size the number of individuals in the population. The default value is 25. Observation: this argument have a great impact on the performance of the search, due to increased number of matrix calculations when the population grows.

generations the number of generations to execute the search. The default value is 100.

seed.p an integer value containing the random number generator.

fitness.criterion the key point of the genetic search. The algorithm will search for the ideal centroids that maximizes one of the pre-specified criteria:

1. "ASW": Average Silhouette Width;
2. "CH": Calinski Harabasz index;
3. "DI": Dunn index;
4. "CI": C-index.

The default value is "ASW" and will be assumed if the user does not supply value or put an invalid entry.

penalty.function an optional user-specified function to be applied over fitness results, to penalize undesirable or incongruent individuals.
plot.internals if TRUE, the evolution of the fitness values and a Silhouette graph will be display after the genetic search. The default value is TRUE.

... other arguments that user may pass to the function.

Details

A call to gama function should at least contain the argument data. In this case, the clustering will be done without penalties over individuals, aiming to maximize the silhouette criterion, and the function will suggest the best value for k, the optimal number of partitions. For the others arguments, the default values applied will be as defined in the call: no normalization or scaling, rates for crossover, mutation, and elitism equals 90%, 1%, and 5%, respectively, population size of 25 individuals, 100 generations, and plot.internals = TRUE.

A lot of other combinations of calls is possible, especially choosing the criteria to guide the maximization of the search ("ASW", "CH", "CI", "DI"), the k value, if known, and adjustments in the genetic parameters, like number of generations, population size, and mutation/crossover/elitism rates.

The function gama returns an S4 object of class "gama". This object contains the following slots:

1. original.data: the original dataset used for clustering.
2. centers: the k (partitions) x d (dimensions) matrix representation of the centroids who best segment the data into k partitions.
3. cluster: a vector of integers (from 1:k) indicating the cluster to which each point is allocated.
4. silhouette: the value for Average Silhouette Width index.
5. calinski_harabasz: the value for Calinski\Harabasz index.
6. e_index: the value for C-index.
7. dunn_index: the value for Dunn index.
8. runtime: the total time spent by the clustering algorithm.
9. call: the string representation of the user call to the function gama and its parameters.

References


Examples

# loading flame dataset
data(flame)

# segmentation of the flame dataset in k = 2 partitions
# the 'plot.internals' says to the program do not plot the graphs about
# genetic evolution search and silhouette index
gama.obj <- gama(flame, k = 4, plot.internals = FALSE, generations = 30)
# ** use at least 100 generations for simple datasets and 500 for complex datasets

# it draws the partitions to which each element belongs
gama.plot.partitions(gama.obj)

## Not run:

# loads data about CPU execution metrics of a distributed
# version of Alternating Least Squares (ALS) algorithm
data(cpu.als)

# a user-defined function to calculate penalties for CPU execution metrics
# whose does not allow the sum of loads above 100%
my.penalty <- function(m.individual,...) {
  penalty <- 0

  # counts how many centroids results in overflow (inequality > 100)
  sums <- apply(m.individual, 1, sum)
  overflow <- which(sums > 100)
  num_constraints = length(overflow)

  # if there are overflows, subtract each dimension
  # by the maximum proportion of the excess x the number of overflows
  if (num_constraints > 0) {
    penalty <- num_constraints * max(abs(sums[overflow] -100)/sums[overflow])
  }

  return (penalty)
}

# call the gama clustering to maximize Dunn index criterion
# by using 500 generations and delegates to GAMA to choose the best k value
gama.obj <- gama(data = cpu.als, fitness.criterion = "DI",
                  generations = 500, penalty.function = my.penalty)

print(gama.obj)

## End(Not run)
Estimates the optimal number of partitions.

Description

This function estimates the best k value for the number of partitions the dataset should be segmented.

Usage

gama.how many k(dataset = NULL, method = "minimal")

Arguments

dataset the original dataset used for clustering.
method the method used to estimate the number of partitions. If 'minimal' is used, the function will perform estimation based on finding the 'elbow' in the Within-cluster Sum of Squares Error graphic. It uses a second derivative approximation, in order to suggest k. If 'broad' is used, the function will proceed an estimation by majority voting of 24 indices, by using the NbClust package.

References


See Also

gama.

Examples

# loads data about CPU execution metrics of a distributed
# version of Alternating Least Squares (ALS) algorithm
library(gama)
data(cpu.als)

# call estimation by using minimal method (Elbow graphic)
k <- gama.how many.k (cpu.als)
print(k)

# call estimation by using broad method (NbClust)
k <- gama.how many.k (cpu.als, method = 'broad')
print(k)
**gama.plot.partitions**  
*Plots results of a Gama clustering.*

**Description**

This function takes a gama object and plots the partitions found by the algorithm. The partitions will be coloured accordingly the distribution of the clusters.

**Usage**

```r
gama.plot.partitions(gama.obj = NULL, view = "pca", ...)```

**Arguments**

- `gama.obj`: an object of type 'gama' generated by an appropriate gama clustering.
- `view`: the representation used to plot the partitions. If 'pca' is used, the function will perform a Principal Component Analysis over the dataset and plot the two main components. If 'total.sum' is used, the function will sum all dimensions for each observation and plot this sum, highlighting its partitions. If omitted, the default value 'pca' will be used.
- `...`: other arguments that user may pass to the function.

**References**


**See Also**

`gama`, `print.gama`.

**Examples**

```r
# loads path_based dataset
library(gama)
data(path_based)

# the gama clustering algorithm call
gamaObj <- gama(path_based, k = 2, generations = 30)
# ** use at least 100 generations for simple datasets and 500 for complex datasets

# a call to gama.plot.partitions function with "Principal Component Analysis" view method
gama.plot.partitions(gamaObj)

# a call to gama.plot.partitions
# Not run: gama.plot.partitions(gamaObj)
```
**Description**

The data set consists of a circular cluster with an opening near the bottom and two Gaussian distributed clusters inside. All description about the data set may be found in *Robust path-based spectral clustering* article, and the Clustering basic benchmark site, in the references.

**Usage**

data(path_based)

**Format**

A data frame containing 240 observations and two dimensions, forming two partitions:

1. x1: real positive values
2. x2: real positive values

Originally, the dataset had contained three dimensions. We intentionally removed the third dimension that corresponds to the label which the data point belongs.

**Source**

The dataset was collected from Clustering basic benchmark site.

**References**


---

**Description**

This function takes a gama object and prints:

1. a sample of the original dataset used for clustering;
2. the cluster solution (partitions);
3. the centers of partitions;
4. the indices Average Silhouette Width (ASW), Calinski Harabasz (CH), C-Index (CI), Dunn index (DI).
print.gama

Usage

## S3 method for class 'gama'
print(x, ...)

Arguments

x an object of type 'gama' generated by an appropriate call to the clustering algorithm.

... other arguments that user may pass to the function.

See Also

gama, gama.plot.partitions.

Examples

## Not run:
# loads data about CPU execution metrics of a distributed
# version of Alternating Least Squares (ALS) algorithm
data(cpus.als)

# call the gama clustering algorithm
gamaObj <- gama(cpus.als, k = 4)

# call the print.gama function to detail the clustering results
# note: print.gama uses generic function concept, which allows a call to print, only.
print(gamaObj)

## End(Not run)
Index

∗Topic datasets
  aggregation, 2
  compound, 3
  cpu.als, 4
  cpu.pca, 5
  flame, 6
  path.based, 12

∗Topic file
  gama, 6
  gama.how_many.k, 10
  gama.plot.partitions, 11
  print.gama, 12

aggregation, 2

compound, 3
cpu.als, 4
cpu.pca, 5

flame, 6

gama, 6, 10, 11, 13
gama.how_many.k, 10

path.based, 12
print.gama, 11, 12