Package 'ClusterR'

December 4, 2023

Type Package

Title Gaussian Mixture Models, K-Means, Mini-Batch-Kmeans, K-Medoids and Affinity Propagation Clustering

Version 1.3.2

Date 2023-12-04

Maintainer Lampros Mouselimis <mouselimislampros@gmail.com>

BugReports https://github.com/mlampros/ClusterR/issues

URL https://github.com/mlampros/ClusterR

Description Gaussian mixture models, k-means, mini-batch-kmeans, k-medoids and affinity propagation clustering with the option to plot, validate, predict (new data) and estimate the optimal number of clusters. The package takes advantage of 'RcppArmadillo' to speed up the computationally intensive parts of the functions. For more information, see (i) ``Clustering in an Object-Oriented Environment" by Anja Struyf, Mia Hubert, Peter Rousseeuw (1997), Journal of Statistical Software, <doi:10.18637/jss.v001.i04>; (ii) ``Web-scale k-means clustering" by D. Sculley (2010), ACM Digital Library, <doi:10.1145/1772690.1772862>; (iii) ``Armadillo: a template-based C++ library for linear algebra" by Sanderson et al (2016), The Journal of Open Source Software, <doi:10.21105/joss.00026>; (iv) ``Clustering by Passing Messages Between Data Points" by Brendan J. Frey and Delbert Dueck, Science 16 Feb 2007: Vol. 315, Issue 5814, pp. 972-976, <doi:10.1126/science.1136800>.

License GPL-3

Encoding UTF-8

SystemRequirements libarmadillo: apt-get install -y libarmadillo-dev (deb), libblas: apt-get install -y libblas-dev (deb), liblapack: apt-get install -y liblapack-dev (deb), libarpack++2: apt-get install -y libarpack++2-dev (deb), gfortran: apt-get install -y gfortran (deb), libgmp3: apt-get install -y libgmp3-dev (deb), libfftw3: apt-get install -y libfftw3-dev (deb), libtiff5: apt-get install -y libtiff5-dev (deb)

LazyData TRUE

Depends R(>= 3.2)

Imports Rcpp (>= 0.12.5), graphics, grDevices, utils, stats, gmp, ggplot2, lifecycle

LinkingTo Rcpp, RcppArmadillo (>= 0.9.1)

Suggests OpenImageR, FD, testthat, covr, knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.2.3

NeedsCompilation yes

Author Lampros Mouselimis [aut, cre] (<https://orcid.org/0000-0002-8024-1546>), Conrad Sanderson [cph] (Author of the C++ Armadillo library), Ryan Curtin [cph] (Author of the C++ Armadillo library), Siddharth Agrawal [cph] (Author of the C code of the Mini-Batch-Kmeans algorithm (https://github.com/siddharth-agrawal/Mini-Batch-K-Means)), Brendan Frey [cph] (Author of the matlab code of the Affinity propagation algorithm (for commercial use please contact the author of the matlab code)), Delbert Dueck [cph] (Author of the matlab code of the Affinity propagation algorithm), Vitalie Spinu [ctb] (Github Contributor)

Repository CRAN

Date/Publication 2023-12-04 18:00:02 UTC

R topics documented:

`AP_affinity_propagation` *Affinity propagation clustering*

Description

Affinity propagation clustering

Usage

```
AP_affinity_propagation(
  data,
  p,
  maxits = 1000,
  convits = 100,
  dampfact = 0.9,
  details = FALSE,
  nonoise = 0,
  time = FALSE
)
```

Arguments

`data`	a matrix. Either a similarity matrix (where number of rows equal to number of columns) or a 3-dimensional matrix where the 1st, 2nd and 3rd column correspond to (i-index, j-index, value) triplet of a similarity matrix.
`p`	a numeric vector of size 1 or size equal to the number of rows of the input matrix. See the details section for more information.
`maxits`	a numeric value specifying the maximum number of iterations (defaults to 1000)
`convits`	a numeric value. If the estimated exemplars stay fixed for convits iterations, the affinity propagation algorithm terminates early (defaults to 100)
`dampfact`	a float number specifying the update equation damping level in [0.5, 1). Higher values correspond to heavy damping, which may be needed if oscillations occur (defaults to 0.9)
`details`	a boolean specifying if details should be printed in the console
`nonoise`	a float number. The affinity propagation algorithm adds a small amount of noise to *data* to prevent degenerate cases; this disables that.
`time`	a boolean. If TRUE then the elapsed time will be printed in the console.

Details

The *affinity propagation* algorithm automatically determines the number of clusters based on the input preference *p*, a real-valued N-vector. p(i) indicates the preference that data point i be chosen as an exemplar. Often a good choice is to set all preferences to median(data). The number of clusters identified can be adjusted by changing this value accordingly. If *p* is a scalar, assumes all preferences are that shared value.

The number of clusters eventually emerges by iteratively passing messages between data points to update two matrices, A and R (Frey and Dueck 2007). The "responsibility" matrix R has values r(i, k) that quantify how well suited point k is to serve as the exemplar for point i relative to other candidate exemplars for point i. The "availability" matrix A contains values a(i, k) representing how "appropriate" point k would be as an exemplar for point i, taking into account other points' preferences for point k as an exemplar. Both matrices R and A are initialized with all zeros. The AP algorithm then performs updates iteratively over the two matrices. First, "Responsibilities" r(i, k) are sent from data points to candidate exemplars to indicate how strongly each data point favors the candidate exemplar over other candidate exemplars. "Availabilities" a(i, k) then are sent from candidate exemplars to data points to indicate the degree to which each candidate exemplar is available to be a cluster center for the data point. In this case, the responsibilities and availabilities are messages that provide evidence about whether each data point should be an exemplar and, if not, to what exemplar that data point should be assigned. For each iteration in the message-passing procedure, the sum of r(k; k) + a(k; k) can be used to identify exemplars. After the messages have converged, two ways exist to identify exemplars. In the first approach, for data point i, if r(i, i) + a(i, i) > 0, then data point i is an exemplar. In the second approach, for data point i, if r(i, i) + a(i, i) > r(i, j) + a(i, j) for all i not equal to j, then data point i is an exemplar. The entire procedure terminates after it reaches a predefined number of iterations or if the determined clusters have remained constant for a certain number of iterations... (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5650075/ – See chapter 2)

Excluding the main diagonal of the similarity matrix when calculating the median as preference ('p') value can be considered as another option too.

References

https://www.psi.toronto.edu/index.php?q=affinity

https://www.psi.toronto.edu/affinitypropagation/faq.html

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5650075/ (SEE chapter 2)

Examples

```
set.seed(1)
dat = matrix(sample(1:255, 2500, replace = TRUE), 100, 25)

smt = 1.0 - distance_matrix(dat, method = 'euclidean', upper = TRUE, diagonal = TRUE)
diag(smt) = 0.0

ap = AP_affinity_propagation(smt, p = median(as.vector(smt)))

str(ap)
```

AP_preferenceRange *Affinity propagation preference range*

Description

Affinity propagation preference range

Usage

```
AP_preferenceRange(data, method = "bound", threads = 1)
```

Arguments

data	a matrix. Either a similarity matrix (where number of rows equal to number of columns) or a 3-dimensional matrix where the 1st, 2nd and 3rd column correspond to (i-index, j-index, value) triplet of a similarity matrix.
method	a character string specifying the preference range method to use. One of 'exact', 'bound'. See the details section for more information.
threads	an integer specifying the number of cores to run in parallel (applies only if *method* is set to 'exact' which is more computationally intensive)

Details

Given a set of similarities, *data*, this function computes a lower bound, pmin, on the value for the preference where the optimal number of clusters (exemplars) changes from 1 to 2, and the exact value of the preference, pmax, where the optimal number of clusters changes from n-1 to n. For N data points, there may be as many as N^2-N pair-wise similarities (note that the similarity of data point i to k need not be equal to the similarity of data point k to i). These may be passed in an NxN matrix of similarities, *data*, where data(i,k) is the similarity of point i to point k. In fact, only a smaller number of relevant similarities need to be provided, in which case the others are assumed to be -Inf. M similarity values are known, can be passed in an Mx3 matrix *data*, where each row of *data* contains a pair of data point indices and a corresponding similarity value: data(j,3) is the similarity of data point data(j,1) to data point data(j,2).

A single-cluster solution may not exist, in which case pmin is set to NaN. The *AP_preferenceRange* uses one of the methods below to compute pmin and pmax:

exact : Computes the exact values for pmin and pmax (Warning: This can be quite slow) *bound* : Computes the exact value for pmax, but estimates pmin using a bound (default)

References

https://www.psi.toronto.edu/affinitypropagation/preferenceRange.m

Examples

```
set.seed(1)
dat = matrix(sample(1:255, 2500, replace = TRUE), 100, 25)

smt = 1.0 - distance_matrix(dat, method = 'euclidean', upper = TRUE, diagonal = TRUE)
diag(smt) = 0.0

ap_range = AP_preferenceRange(smt, method = "bound")
```

---

center_scale *Function to scale and/or center the data*

Description

Function to scale and/or center the data

Usage

```
center_scale(data, mean_center = TRUE, sd_scale = TRUE)
```

Arguments

data	matrix or data frame
mean_center	either TRUE or FALSE. If mean_center is TRUE then the mean of each column will be subtracted
sd_scale	either TRUE or FALSE. See the details section for more information

Details

If sd_scale is TRUE and mean_center is TRUE then each column will be divided by the standard deviation. If sd_scale is TRUE and mean_center is FALSE then each column will be divided by sqrt(sum(x^2) / (n-1)). In case of missing values the function raises an error. In case that the standard deviation equals zero then the standard deviation will be replaced with 1.0, so that NaN's can be avoided by division

Value

a matrix

Examples

```
data(dietary_survey_IBS)

dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

dat = center_scale(dat, mean_center = TRUE, sd_scale = TRUE)
```

Clara_Medoids *Clustering large applications*

Description

Clustering large applications

Usage

```
Clara_Medoids(
  data,
  clusters,
  samples,
  sample_size,
  distance_metric = "euclidean",
  minkowski_p = 1,
  threads = 1,
  swap_phase = TRUE,
  fuzzy = FALSE,
  verbose = FALSE,
  seed = 1
)
```

Arguments

data	matrix or data frame
clusters	the number of clusters
samples	number of samples to draw from the data set
sample_size	fraction of data to draw in each sample iteration. It should be a float number greater than 0.0 and less or equal to 1.0
distance_metric	a string specifying the distance method. One of, *euclidean*, *manhattan*, *chebyshev*, *canberra*, *braycurtis*, *pearson_correlation*, *simple_matching_coefficient*, *minkowski*, *hamming*, *jaccard_coefficient*, *Rao_coefficient*, *mahalanobis*, *cosine*
minkowski_p	a numeric value specifying the minkowski parameter in case that distance_metric = "minkowski"

threads	an integer specifying the number of cores to run in parallel. Openmp will be utilized to parallelize the number of the different sample draws
swap_phase	either TRUE or FALSE. If TRUE then both phases ('build' and 'swap') will take place. The 'swap_phase' is considered more computationally intensive.
fuzzy	either TRUE or FALSE. If TRUE, then probabilities for each cluster will be returned based on the distance between observations and medoids
verbose	either TRUE or FALSE, indicating whether progress is printed during clustering
seed	integer value for random number generator (RNG)

Details

The Clara_Medoids function is implemented in the same way as the 'clara' (clustering large applications) algorithm (Kaufman and Rousseeuw(1990)). In the 'Clara_Medoids' the 'Cluster_Medoids' function will be applied to each sample draw.

Value

a list with the following attributes : medoids, medoid_indices, sample_indices, best_dissimilarity, clusters, fuzzy_probs (if fuzzy = TRUE), clustering_stats, dissimilarity_matrix, silhouette_matrix

Author(s)

Lampros Mouselimis

References

Anja Struyf, Mia Hubert, Peter J. Rousseeuw, (Feb. 1997), Clustering in an Object-Oriented Environment, Journal of Statistical Software, Vol 1, Issue 4

Examples

```
data(dietary_survey_IBS)

dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

dat = center_scale(dat)

clm = Clara_Medoids(dat, clusters = 3, samples = 5, sample_size = 0.2, swap_phase = TRUE)
```

`Cluster_Medoids` *Partitioning around medoids*

Description

Partitioning around medoids

Usage

```
Cluster_Medoids(
  data,
  clusters,
  distance_metric = "euclidean",
  minkowski_p = 1,
  threads = 1,
  swap_phase = TRUE,
  fuzzy = FALSE,
  verbose = FALSE,
  seed = 1
)
```

Arguments

`data`	matrix or data frame. The data parameter can be also a dissimilarity matrix, where the main diagonal equals 0.0 and the number of rows equals the number of columns
`clusters`	the number of clusters
`distance_metric`	a string specifying the distance method. One of, *euclidean*, *manhattan*, *chebyshev*, *canberra*, *braycurtis*, *pearson_correlation*, *simple_matching_coefficient*, *minkowski*, *hamming*, *jaccard_coefficient*, *Rao_coefficient*, *mahalanobis*, *cosine*
`minkowski_p`	a numeric value specifying the minkowski parameter in case that distance_metric = "minkowski"
`threads`	an integer specifying the number of cores to run in parallel
`swap_phase`	either TRUE or FALSE. If TRUE then both phases ('build' and 'swap') will take place. The 'swap_phase' is considered more computationally intensive.
`fuzzy`	either TRUE or FALSE. If TRUE, then probabilities for each cluster will be returned based on the distance between observations and medoids
`verbose`	either TRUE or FALSE, indicating whether progress is printed during clustering
`seed`	‘r lifecycle::badge("deprecated")‘ ‘seed‘ (integer value for random number generator (RNG)) is no longer supported and will be removed in version 1.4.0

Details

Due to the fact that I didn't have access to the book 'Finding Groups in Data, Kaufman and Rousseeuw, 1990' (which includes the exact algorithm) I implemented the 'Cluster_Medoids' function based on the paper 'Clustering in an Object-Oriented Environment' (see 'References'). Therefore, the 'Cluster_Medoids' function is an approximate implementation and not an exact one. Furthermore, in comparison to k-means clustering, the function 'Cluster_Medoids' is more robust, because it minimizes the sum of unsquared dissimilarities. Moreover, it doesn't need initial guesses for the cluster centers.

Value

a list with the following attributes: medoids, medoid_indices, best_dissimilarity, dissimilarity_matrix, clusters, fuzzy_probs (if fuzzy = TRUE), silhouette_matrix, clustering_stats

Author(s)

Lampros Mouselimis

References

Anja Struyf, Mia Hubert, Peter J. Rousseeuw, (Feb. 1997), Clustering in an Object-Oriented Environment, Journal of Statistical Software, Vol 1, Issue 4

Examples

```
data(dietary_survey_IBS)

dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

dat = center_scale(dat)

cm = Cluster_Medoids(dat, clusters = 3, distance_metric = 'euclidean', swap_phase = TRUE)
```

---

cost_clusters_from_dissim_medoids

Compute the cost and clusters based on an input dissimilarity matrix and medoids

---

Description

Compute the cost and clusters based on an input dissimilarity matrix and medoids

Usage

```
cost_clusters_from_dissim_medoids(data, medoids)
```

Arguments

data	a dissimilarity matrix, where the main diagonal equals 0.0 and the number of rows equals the number of columns
medoids	a vector of output medoids of the 'Cluster_Medoids', 'Clara_Medoids' or any other 'partition around medoids' function

Value

a list object that includes the cost and the clusters

Author(s)

Lampros Mouselimis

Examples

```
data(dietary_survey_IBS)
dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]
dat = center_scale(dat)

cm = Cluster_Medoids(dat, clusters = 3, distance_metric = 'euclidean', swap_phase = TRUE)
res = cost_clusters_from_dissim_medoids(data = cm$dissimilarity_matrix, medoids = cm$medoid_indices)

# cm$best_dissimilarity == res$cost
# table(cm$clusters, res$clusters)
```

---

dietary_survey_IBS *Synthetic data using a dietary survey of patients with irritable bowel syndrome (IBS)*

Description

The data are based on the article "A dietary survey of patients with irritable bowel syndrome". The mean and standard deviation of the table 1 (Foods perceived as causing or worsening irritable bowel syndrome symptoms in the IBS group and digestive symptoms in the healthy comparative group) were used to generate the synthetic data.

Usage

```
data(dietary_survey_IBS)
```

Format

A data frame with 400 Instances and 43 attributes (including the class attribute, "class")

Details

The predictors are: bread, wheat, pasta, breakfast_cereal, yeast, spicy_food, curry, chinese_takeaway, chilli, cabbage, onion, garlic, potatoes, pepper, vegetables_unspecified, tomato, beans_and_pulses, mushroom, fatty_foods_unspecified, sauces, chocolate, fries, crisps, desserts, eggs, red_meat, processed_meat, pork, chicken, fish_shellfish, dairy_products_unspecified, cheese, cream, milk, fruit_unspecified, nuts_and_seeds, orange, apple, banana, grapes, alcohol, caffeine

The response variable ("class") consists of two groups: healthy-group (class == 0) vs. the IBS-patients (class == 1)

References

P. Hayes, C. Corish, E. O'Mahony, E. M. M. Quigley (May 2013). A dietary survey of patients with irritable bowel syndrome. Journal of Human Nutrition and Dietetics.

Examples

```
data(dietary_survey_IBS)

X = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

y = dietary_survey_IBS[, ncol(dietary_survey_IBS)]
```

---

distance_matrix *Distance matrix calculation*

Description

Distance matrix calculation

Usage

```
distance_matrix(
  data,
  method = "euclidean",
  upper = FALSE,
  diagonal = FALSE,
  minkowski_p = 1,
  threads = 1
)
```

Arguments

data	matrix or data frame
method	a string specifying the distance method. One of, *euclidean*, *manhattan*, *chebyshev*, *canberra*, *braycurtis*, *pearson_correlation*, *simple_matching_coefficient*, *minkowski*, *hamming*, *jaccard_coefficient*, *Rao_coefficient*, *mahalanobis*, *cosine*

upper	either TRUE or FALSE specifying if the upper triangle of the distance matrix should be returned. If FALSE then the upper triangle will be filled with NA's
diagonal	either TRUE or FALSE specifying if the diagonal of the distance matrix should be returned. If FALSE then the diagonal will be filled with NA's
minkowski_p	a numeric value specifying the minkowski parameter in case that method = "minkowski"
threads	the number of cores to run in parallel (if OpenMP is available)

Value

a matrix

Examples

```
data(dietary_survey_IBS)

dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

dat = distance_matrix(dat, method = 'euclidean', upper = TRUE, diagonal = TRUE)
```

---

external_validation *external clustering validation*

Description

external clustering validation

Usage

```
external_validation(
  true_labels,
  clusters,
  method = "adjusted_rand_index",
  summary_stats = FALSE
)
```

Arguments

true_labels	a numeric vector of length equal to the length of the clusters vector
clusters	a numeric vector (the result of a clustering method) of length equal to the length of the true_labels
method	one of *rand_index*, *adjusted_rand_index*, *jaccard_index*, *fowlkes_Mallows_index*, *mirkin_metric*, *purity*, *entropy*, *nmi* (normalized mutual information), *var_info* (variation of information), and *nvi* (normalized variation of information)
summary_stats	besides the available methods the summary_stats parameter prints also the specificity, sensitivity, precision, recall and F-measure of the clusters

Details

This function uses external validation methods to evaluate the clustering results

Value

if summary_stats is FALSE the function returns a float number, otherwise it returns also a summary statistics table

Author(s)

Lampros Mouselimis

Examples

```
data(dietary_survey_IBS)

dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

X = center_scale(dat)

km = KMeans_rcpp(X, clusters = 2, num_init = 5, max_iters = 100, initializer = 'kmeans++')

res = external_validation(dietary_survey_IBS$class, km$clusters, method = "adjusted_rand_index")
```

GMM *Gaussian Mixture Model clustering*

Description

Gaussian Mixture Model clustering

Usage

```
GMM(
  data,
  gaussian_comps = 1,
  dist_mode = "eucl_dist",
  seed_mode = "random_subset",
  km_iter = 10,
  em_iter = 5,
  verbose = FALSE,
  var_floor = 1e-10,
  seed = 1,
  full_covariance_matrices = FALSE
)
```

Arguments

data	matrix or data frame
gaussian_comps	the number of gaussian mixture components
dist_mode	the distance used during the seeding of initial means and k-means clustering. One of, *eucl_dist*, *maha_dist*.
seed_mode	how the initial means are seeded prior to running k-means and/or EM algorithms. One of, *static_subset*, *random_subset*, *static_spread*, *random_spread*.
km_iter	the number of iterations of the k-means algorithm
em_iter	the number of iterations of the EM algorithm
verbose	either TRUE or FALSE; enable or disable printing of progress during the k-means and EM algorithms
var_floor	the variance floor (smallest allowed value) for the diagonal covariances
seed	integer value for random number generator (RNG)
full_covariance_matrices	a boolean. If FALSE "diagonal" covariance matrices (i.e. in each covariance matrix, all entries outside the main diagonal are assumed to be zero) otherwise "full" covariance matrices will be returned. Be aware in case of "full" covariance matrices a cube (3-dimensional) rather than a matrix for the output "covariance_matrices" value will be returned.

Details

This function is an R implementation of the 'gmm_diag' class of the Armadillo library. The only exception is that user defined parameter settings are not supported, such as seed_mode = 'keep_existing'. For probabilistic applications, better model parameters are typically learned with dist_mode set to maha_dist. For vector quantisation applications, model parameters should be learned with dist_mode set to eucl_dist, and the number of EM iterations set to zero. In general, a sufficient number of k-means and EM iterations is typically about 10. The number of training samples should be much larger than the number of Gaussians. Seeding the initial means with static_spread and random_spread can be much more time consuming than with static_subset and random_subset. The k-means and EM algorithms will run faster on multi-core machines when OpenMP is enabled in your compiler (eg. -fopenmp in GCC)

Value

a list consisting of the centroids, covariance matrix (where each row of the matrix represents a diagonal covariance matrix), weights and the log-likelihoods for each gaussian component. In case of Error it returns the error message and the possible causes.

References

http://arma.sourceforge.net/docs.html

Examples

```
data(dietary_survey_IBS)

dat = as.matrix(dietary_survey_IBS[, -ncol(dietary_survey_IBS)])

dat = center_scale(dat)

gmm = GMM(dat, 2, "maha_dist", "random_subset", 10, 10)
```

KMeans_arma *k-means using the Armadillo library*

Description

k-means using the Armadillo library

Usage

```
KMeans_arma(
  data,
  clusters,
  n_iter = 10,
  seed_mode = "random_subset",
  verbose = FALSE,
  CENTROIDS = NULL,
  seed = 1
)
```

Arguments

data	matrix or data frame
clusters	the number of clusters
n_iter	the number of clustering iterations (about 10 is typically sufficient)
seed_mode	how the initial centroids are seeded. One of, *keep_existing*, *static_subset*, *random_subset*, *static_spread*, *random_spread*.
verbose	either TRUE or FALSE, indicating whether progress is printed during clustering
CENTROIDS	a matrix of initial cluster centroids. The rows of the CENTROIDS matrix should be equal to the number of clusters and the columns should be equal to the columns of the data. CENTROIDS should be used in combination with seed_mode 'keep_existing'.
seed	integer value for random number generator (RNG)

Details

This function is an R implementation of the 'kmeans' class of the Armadillo library. It is faster than the KMeans_rcpp function but it lacks some features. For more info see the details section of the KMeans_rcpp function. The number of columns should be larger than the number of clusters or CENTROIDS. If the clustering fails, the means matrix is reset and a bool set to false is returned. The clustering will run faster on multi-core machines when OpenMP is enabled in your compiler (eg. -fopenmp in GCC)

Value

the centroids as a matrix. In case of Error it returns the error message, whereas in case of an empty centroids-matrix it returns a warning-message.

References

http://arma.sourceforge.net/docs.html

Examples

```
data(dietary_survey_IBS)

dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

dat = center_scale(dat)

km = KMeans_arma(dat, clusters = 2, n_iter = 10, "random_subset")
```

KMeans_rcpp *k-means using RcppArmadillo*

Description

k-means using RcppArmadillo

Usage

```
KMeans_rcpp(
  data,
  clusters,
  num_init = 1,
  max_iters = 100,
  initializer = "kmeans++",
  fuzzy = FALSE,
  verbose = FALSE,
  CENTROIDS = NULL,
  tol = 1e-04,
```

```
  tol_optimal_init = 0.3,
  seed = 1
)
```

Arguments

data	matrix or data frame
clusters	the number of clusters
num_init	number of times the algorithm will be run with different centroid seeds
max_iters	the maximum number of clustering iterations
initializer	the method of initialization. One of, *optimal_init*, *quantile_init*, *kmeans++* and *random*. See details for more information
fuzzy	either TRUE or FALSE. If TRUE, then prediction probabilities will be calculated using the distance between observations and centroids
verbose	either TRUE or FALSE, indicating whether progress is printed during clustering.
CENTROIDS	a matrix of initial cluster centroids. The rows of the CENTROIDS matrix should be equal to the number of clusters and the columns should be equal to the columns of the data.
tol	a float number. If, in case of an iteration (iteration > 1 and iteration < max_iters) 'tol' is greater than the squared norm of the centroids, then kmeans has converged
tol_optimal_init	tolerance value for the 'optimal_init' initializer. The higher this value is, the far appart from each other the centroids are.
seed	integer value for random number generator (RNG)

Details

This function has the following features in comparison to the KMeans_arma function:

Besides optimal_init, quantile_init, random and kmeans++ initilizations one can specify the centroids using the CENTROIDS parameter.

The running time and convergence of the algorithm can be adjusted using the num_init, max_iters and tol parameters.

If num_init > 1 then KMeans_rcpp returns the attributes of the best initialization using as criterion the within-cluster-sum-of-squared-error.

————————initializers————————————-

optimal_init : this initializer adds rows of the data incrementally, while checking that they do not already exist in the centroid-matrix [experimental]

quantile_init : initialization of centroids by using the cummulative distance between observations and by removing potential duplicates [experimental]

kmeans++ : kmeans++ initialization. Reference : http://theory.stanford.edu/~sergei/papers/kMeansPP-soda.pdf AND http://stackoverflow.com/questions/5466323/how-exactly-does-k-means-work

random : random selection of data rows as initial centroids

Value

a list with the following attributes: clusters, fuzzy_clusters (if fuzzy = TRUE), centroids, total_SSE, best_initialization, WCSS_per_cluster, obs_per_cluster, between.SS_DIV_total.SS

Author(s)

Lampros Mouselimis

Examples

```
data(dietary_survey_IBS)

dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

dat = center_scale(dat)

km = KMeans_rcpp(dat, clusters = 2, num_init = 5, max_iters = 100, initializer = 'kmeans++')
```

MiniBatchKmeans *Mini-batch-k-means using RcppArmadillo*

Description

Mini-batch-k-means using RcppArmadillo

Usage

```
MiniBatchKmeans(
  data,
  clusters,
  batch_size = 10,
  num_init = 1,
  max_iters = 100,
  init_fraction = 1,
  initializer = "kmeans++",
  early_stop_iter = 10,
  verbose = FALSE,
  CENTROIDS = NULL,
  tol = 1e-04,
  tol_optimal_init = 0.3,
  seed = 1
)
```

Arguments

data	matrix or data frame
clusters	the number of clusters
batch_size	the size of the mini batches
num_init	number of times the algorithm will be run with different centroid seeds
max_iters	the maximum number of clustering iterations
init_fraction	percentage of data to use for the initialization centroids (applies if initializer is *kmeans++* or *optimal_init*). Should be a float number between 0.0 and 1.0.
initializer	the method of initialization. One of, *optimal_init*, *quantile_init*, *kmeans++* and *random*. See details for more information
early_stop_iter	continue that many iterations after calculation of the best within-cluster-sum-of-squared-error
verbose	either TRUE or FALSE, indicating whether progress is printed during clustering
CENTROIDS	a matrix of initial cluster centroids. The rows of the CENTROIDS matrix should be equal to the number of clusters and the columns should be equal to the columns of the data
tol	a float number. If, in case of an iteration (iteration > 1 and iteration < max_iters) 'tol' is greater than the squared norm of the centroids, then kmeans has converged
tol_optimal_init	tolerance value for the 'optimal_init' initializer. The higher this value is, the far appart from each other the centroids are.
seed	integer value for random number generator (RNG)

Details

This function performs k-means clustering using mini batches.

————————initializers————————————-

optimal_init : this initializer adds rows of the data incrementally, while checking that they do not already exist in the centroid-matrix [experimental]

quantile_init : initialization of centroids by using the cummulative distance between observations and by removing potential duplicates [experimental]

kmeans++ : kmeans++ initialization. Reference : http://theory.stanford.edu/~sergei/papers/kMeansPP-soda.pdf AND http://stackoverflow.com/questions/5466323/how-exactly-does-k-means-work

random : random selection of data rows as initial centroids

Value

a list with the following attributes: centroids, WCSS_per_cluster, best_initialization, iters_per_initialization

Author(s)

Lampros Mouselimis

References

http://www.eecs.tufts.edu/~dsculley/papers/fastkmeans.pdf, https://github.com/siddharth-agrawal/Mini-Batch-K-Means

Examples

```
data(dietary_survey_IBS)

dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

dat = center_scale(dat)

MbatchKm = MiniBatchKmeans(dat, clusters = 2, batch_size = 20, num_init = 5, early_stop_iter = 10)
```

---

mushroom *The mushroom data*

Description

This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like 'leaflets three, let it be' for Poisonous Oak and Ivy.

Usage

```
data(mushroom)
```

Format

A data frame with 8124 Instances and 23 attributes (including the class attribute, "class")

Details

The column names of the data (including the class) appear in the following order:

1. class: edible=e, poisonous=p

2. cap-shape: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s

3. cap-surface: fibrous=f, grooves=g, scaly=y, smooth=s

4. cap-color: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y

5. bruises: bruises=t, no=f

6. odor: almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s

7. gill-attachment: attached=a, descending=d, free=f, notched=n

8. gill-spacing: close=c, crowded=w, distant=d

9. gill-size: broad=b, narrow=n

10. gill-color: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y

11. stalk-shape: enlarging=e, tapering=t

12. stalk-root: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?

13. stalk-surface-above-ring: fibrous=f, scaly=y, silky=k, smooth=s

14. stalk-surface-below-ring: fibrous=f, scaly=y, silky=k, smooth=s

15. stalk-color-above-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y

16. stalk-color-below-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y

17. veil-type: partial=p, universal=u

18. veil-color: brown=n, orange=o, white=w, yellow=y

19. ring-number: none=n, one=o, two=t

20. ring-type: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z

21. spore-print-color: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y

22. population: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y

23. habitat: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

References

Mushroom records drawn from The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf

Donor: Jeff Schlimmer (Jeffrey.Schlimmer@a.gp.cs.cmu.edu)

download source: https://archive.ics.uci.edu/ml/datasets/Mushroom

Examples

```
data(mushroom)

X = mushroom[, -1]

y = mushroom[, 1]
```

`Optimal_Clusters_GMM` *Optimal number of Clusters for the gaussian mixture models*

Description

Optimal number of Clusters for the gaussian mixture models

Usage

```
Optimal_Clusters_GMM(
  data,
  max_clusters,
  criterion = "AIC",
  dist_mode = "eucl_dist",
  seed_mode = "random_subset",
  km_iter = 10,
  em_iter = 5,
  verbose = FALSE,
  var_floor = 1e-10,
  plot_data = TRUE,
  seed = 1
)
```

Arguments

`data`	matrix or data frame
`max_clusters`	either a numeric value, a contiguous or non-continguous numeric vector specifying the cluster search space
`criterion`	one of 'AIC' or 'BIC'
`dist_mode`	the distance used during the seeding of initial means and k-means clustering. One of, *eucl_dist*, *maha_dist*.
`seed_mode`	how the initial means are seeded prior to running k-means and/or EM algorithms. One of, *static_subset*, *random_subset*, *static_spread*, *random_spread*.
`km_iter`	the number of iterations of the k-means algorithm
`em_iter`	the number of iterations of the EM algorithm
`verbose`	either TRUE or FALSE; enable or disable printing of progress during the k-means and EM algorithms
`var_floor`	the variance floor (smallest allowed value) for the diagonal covariances
`plot_data`	either TRUE or FALSE indicating whether the results of the function should be plotted
`seed`	integer value for random number generator (RNG)

Details

AIC : the Akaike information criterion

BIC : the Bayesian information criterion

In case that the *max_clusters* parameter is a contiguous or non-contiguous vector then plotting is disabled. Therefore, plotting is enabled only if the *max_clusters* parameter is of length 1.

Value

a vector with either the AIC or BIC for each iteration. In case of Error it returns the error message and the possible causes.

Author(s)

Lampros Mouselimis

Examples

```
data(dietary_survey_IBS)

dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

dat = center_scale(dat)

opt_gmm = Optimal_Clusters_GMM(dat, 10, criterion = "AIC", plot_data = FALSE)


#----------------------------
# non-contiguous search space
#----------------------------

search_space = c(2,5)

opt_gmm = Optimal_Clusters_GMM(dat, search_space, criterion = "AIC", plot_data = FALSE)
```

Optimal_Clusters_KMeans

Optimal number of Clusters for Kmeans or Mini-Batch-Kmeans

Description

Optimal number of Clusters for Kmeans or Mini-Batch-Kmeans

Usage

```
Optimal_Clusters_KMeans(
  data,
  max_clusters,
  criterion = "variance_explained",
  fK_threshold = 0.85,
  num_init = 1,
  max_iters = 200,
  initializer = "kmeans++",
  tol = 1e-04,
  plot_clusters = TRUE,
  verbose = FALSE,
  tol_optimal_init = 0.3,
  seed = 1,
  mini_batch_params = NULL
)
```

Arguments

data	matrix or data frame
max_clusters	either a numeric value, a contiguous or non-continguous numeric vector specifying the cluster search space
criterion	one of *variance_explained*, *WCSSE*, *dissimilarity*, *silhouette*, *distortion_fK*, *AIC*, *BIC* and *Adjusted_Rsquared*. See details for more information.
fK_threshold	a float number used in the 'distortion_fK' criterion
num_init	number of times the algorithm will be run with different centroid seeds
max_iters	the maximum number of clustering iterations
initializer	the method of initialization. One of, *optimal_init*, *quantile_init*, *kmeans++* and *random*. See details for more information
tol	a float number. If, in case of an iteration (iteration > 1 and iteration < max_iters) 'tol' is greater than the squared norm of the centroids, then kmeans has converged
plot_clusters	either TRUE or FALSE, indicating whether the results of the *Optimal_Clusters_KMeans* function should be plotted
verbose	either TRUE or FALSE, indicating whether progress is printed during clustering
tol_optimal_init	tolerance value for the 'optimal_init' initializer. The higher this value is, the far appart from each other the centroids are.
seed	integer value for random number generator (RNG)
mini_batch_params	either NULL or a list of the following parameters : *batch_size*, *init_fraction*, *early_stop_iter*. If not NULL then the optimal number of clusters will be found based on the Mini-Batch-Kmeans. See the details and examples sections for more information.

Details

——————criteria————————————

variance_explained : the sum of the within-cluster-sum-of-squares-of-all-clusters divided by the total sum of squares

WCSSE : the sum of the within-cluster-sum-of-squares-of-all-clusters

dissimilarity : the average intra-cluster-dissimilarity of all clusters (the distance metric defaults to euclidean)

silhouette : the average silhouette width where first the average per cluster silhouette is computed and then the global average (the distance metric defaults to euclidean). To compute the silhouette width for each cluster separately see the 'silhouette_of_clusters()' function

distortion_fK : this criterion is based on the following paper, 'Selection of K in K-means clustering' (https://www.ee.columbia.edu/~dpwe/papers/PhamDN05-kmeans.pdf)

AIC : the Akaike information criterion

BIC : the Bayesian information criterion

Adjusted_Rsquared : the adjusted R^2 statistic

——————initializers————————————-

optimal_init : this initializer adds rows of the data incrementally, while checking that they do not already exist in the centroid-matrix [experimental]

quantile_init : initialization of centroids by using the cummulative distance between observations and by removing potential duplicates [experimental]

kmeans++ : kmeans++ initialization. Reference : http://theory.stanford.edu/~sergei/papers/kMeansPP-soda.pdf AND http://stackoverflow.com/questions/5466323/how-exactly-does-k-means-work

random : random selection of data rows as initial centroids

If the *mini_batch_params* parameter is not NULL then the optimal number of clusters will be found based on the Mini-batch-Kmeans algorithm, otherwise based on the Kmeans. The higher the *init_fraction* parameter is the more close the results between Mini-Batch-Kmeans and Kmeans will be.

In case that the *max_clusters* parameter is a contiguous or non-contiguous vector then plotting is disabled. Therefore, plotting is enabled only if the *max_clusters* parameter is of length 1. Moreover, the *distortion_fK* criterion can't be computed if the *max_clusters* parameter is a contiguous or non-continguous vector (the *distortion_fK* criterion requires consecutive clusters). The same applies also to the *Adjusted_Rsquared* criterion which returns incorrect output.

Value

a vector with the results for the specified criterion. If plot_clusters is TRUE then it plots also the results.

Author(s)

Lampros Mouselimis

References

https://www.ee.columbia.edu/~dpwe/papers/PhamDN05-kmeans.pdf

Examples

```
data(dietary_survey_IBS)

dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

dat = center_scale(dat)


#-------
# kmeans
#-------

opt_km = Optimal_Clusters_KMeans(dat, max_clusters = 10, criterion = "distortion_fK",

                                 plot_clusters = FALSE)

#------------------
# mini-batch-kmeans
#------------------


params_mbkm = list(batch_size = 10, init_fraction = 0.3, early_stop_iter = 10)

opt_mbkm = Optimal_Clusters_KMeans(dat, max_clusters = 10, criterion = "distortion_fK",

                                 plot_clusters = FALSE, mini_batch_params = params_mbkm)


#----------------------------
# non-contiguous search space
#----------------------------

search_space = c(2,5)

opt_km = Optimal_Clusters_KMeans(dat, max_clusters = search_space,

                                 criterion = "variance_explained",

                                 plot_clusters = FALSE)
```

`Optimal_Clusters_Medoids` *Optimal number of Clusters for the partitioning around Medoids functions*

Description

Optimal number of Clusters for the partitioning around Medoids functions

Usage

```
Optimal_Clusters_Medoids(
  data,
  max_clusters,
  distance_metric,
  criterion = "dissimilarity",
  clara_samples = 0,
  clara_sample_size = 0,
  minkowski_p = 1,
  swap_phase = TRUE,
  threads = 1,
  verbose = FALSE,
  plot_clusters = TRUE,
  seed = 1
)
```

Arguments

`data`	matrix or data.frame. If both clara_samples and clara_sample_size equal 0, then the data parameter can be also a dissimilarity matrix, where the main diagonal equals 0.0 and the number of rows equals the number of columns
`max_clusters`	either a numeric value, a contiguous or non-continguous numeric vector specifying the cluster search space
`distance_metric`	a string specifying the distance method. One of, *euclidean*, *manhattan*, *chebyshev*, *canberra*, *braycurtis*, *pearson_correlation*, *simple_matching_coefficient*, *minkowski*, *hamming*, *jaccard_coefficient*, *Rao_coefficient*, *mahalanobis*, *cosine*
`criterion`	one of 'dissimilarity' or 'silhouette'
`clara_samples`	number of samples to draw from the data set in case of clustering large applications (clara)
`clara_sample_size`	fraction of data to draw in each sample iteration in case of clustering large applications (clara). It should be a float number greater than 0.0 and less or equal to 1.0
`minkowski_p`	a numeric value specifying the minkowski parameter in case that distance_metric = "minkowski"
`swap_phase`	either TRUE or FALSE. If TRUE then both phases ('build' and 'swap') will take place. The 'swap_phase' is considered more computationally intensive.
`threads`	an integer specifying the number of cores to run in parallel. Openmp will be utilized to parallelize the number of sample draws
`verbose`	either TRUE or FALSE, indicating whether progress is printed during clustering

plot_clusters	TRUE or FALSE, indicating whether the iterative results should be plotted. See the details section for more information
seed	integer value for random number generator (RNG)

Details

In case of plot_clusters = TRUE, the first plot will be either a plot of dissimilarities or both dissimilarities and silhouette widths giving an indication of the optimal number of the clusters. Then, the user will be asked to give an optimal value for the number of the clusters and after that the second plot will appear with either the dissimilarities or the silhouette widths belonging to each cluster.

In case that the *max_clusters* parameter is a contiguous or non-contiguous vector then plotting is disabled. Therefore, plotting is enabled only if the *max_clusters* parameter is of length 1.

Value

a list of length equal to the max_clusters parameter (the first sublist equals NULL, as dissimilarities and silhouette widths can be calculated if the number of clusters > 1). If plot_clusters is TRUE then the function plots also the results.

Author(s)

Lampros Mouselimis

Examples

```
## Not run:
data(soybean)

dat = soybean[, -ncol(soybean)]

opt_md = Optimal_Clusters_Medoids(dat, 10, 'jaccard_coefficient', plot_clusters = FALSE)


#----------------------------
# non-contiguous search space
#----------------------------

search_space = c(2,5)

opt_md = Optimal_Clusters_Medoids(dat, search_space, 'jaccard_coefficient', plot_clusters = FALSE)


## End(Not run)
```

plot_2d *2-dimensional plots*

Description

2-dimensional plots

Usage

```
plot_2d(data, clusters, centroids_medoids)
```

Arguments

data	a 2-dimensional matrix or data frame
clusters	numeric vector of length equal to the number of rows of the data, which is the result of a clustering method
centroids_medoids	a matrix of centroids or medoids. The rows of the centroids_medoids should be equal to the length of the unique values of the clusters and the columns should be equal to the columns of the data.

Details

This function plots the clusters using 2-dimensional data and medoids or centroids.

Value

a plot

Author(s)

Lampros Mouselimis

Examples

```
# data(dietary_survey_IBS)

# dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

# dat = center_scale(dat)

# pca_dat = stats::princomp(dat)$scores[, 1:2]

# km = KMeans_rcpp(pca_dat, clusters = 2, num_init = 5, max_iters = 100)

# plot_2d(pca_dat, km$clusters, km$centroids)
```

predict_GMM *Prediction function for a Gaussian Mixture Model object*

Description

Prediction function for a Gaussian Mixture Model object

Usage

```
predict_GMM(data, CENTROIDS, COVARIANCE, WEIGHTS)

## S3 method for class 'GMMCluster'
predict(object, newdata, ...)
```

Arguments

data	matrix or data frame
CENTROIDS	matrix or data frame containing the centroids (means), stored as row vectors
COVARIANCE	matrix or data frame containing the diagonal covariance matrices, stored as row vectors
WEIGHTS	vector containing the weights
object, newdata, ...	arguments for the ‘predict‘ generic

Details

This function takes the centroids, covariance matrix and weights from a trained model and returns the log-likelihoods, cluster probabilities and cluster labels for new data.

Value

a list consisting of the log-likelihoods, cluster probabilities and cluster labels.

Author(s)

Lampros Mouselimis

Examples

```
data(dietary_survey_IBS)

dat = as.matrix(dietary_survey_IBS[, -ncol(dietary_survey_IBS)])

dat = center_scale(dat)

gmm = GMM(dat, 2, "maha_dist", "random_subset", 10, 10)
```

```
# pr = predict_GMM(dat, gmm$centroids, gmm$covariance_matrices, gmm$weights)
```

`predict_KMeans` *Prediction function for the k-means*

Description

Prediction function for the k-means

Usage

```
predict_KMeans(data, CENTROIDS, threads = 1, fuzzy = FALSE)

## S3 method for class 'KMeansCluster'
predict(object, newdata, fuzzy = FALSE, threads = 1, ...)
```

Arguments

`data`	matrix or data frame
`CENTROIDS`	a matrix of initial cluster centroids. The rows of the CENTROIDS matrix should be equal to the number of clusters and the columns should be equal to the columns of the data.
`threads`	an integer specifying the number of cores to run in parallel
`fuzzy`	either TRUE or FALSE. If TRUE, then probabilities for each cluster will be returned based on the distance between observations and centroids.
`object, newdata, ...`	arguments for the ‘predict‘ generic

Details

This function takes the data and the output centroids and returns the clusters.

Value

a vector (clusters)

Author(s)

Lampros Mouselimis

Examples

```
data(dietary_survey_IBS)

dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

dat = center_scale(dat)

km = KMeans_rcpp(dat, clusters = 2, num_init = 5, max_iters = 100, initializer = 'kmeans++')

pr = predict_KMeans(dat, km$centroids, threads = 1)
```

---

predict_MBatchKMeans *Prediction function for Mini-Batch-k-means*

Description

Prediction function for Mini-Batch-k-means

Usage

```
predict_MBatchKMeans(data, CENTROIDS, fuzzy = FALSE, updated_output = FALSE)

## S3 method for class 'MBatchKMeans'
predict(object, newdata, fuzzy = FALSE, ...)
```

Arguments

`data`	matrix or data frame
`CENTROIDS`	a matrix of initial cluster centroids. The rows of the CENTROIDS matrix should be equal to the number of clusters and the columns should equal the columns of the data.
`fuzzy`	either TRUE or FALSE. If TRUE then prediction probabilities will be calculated using the distance between observations and centroids.
`updated_output`	either TRUE or FALSE. If TRUE then the 'predict_MBatchKMeans' function will follow the same output object behaviour as the 'predict_KMeans' function (if fuzzy is TRUE it will return probabilities otherwise it will return the hard clusters). This parameter will be removed in version 1.4.0 because this will become the default output format.
`object, newdata, ...`	arguments for the 'predict' generic

Details

This function takes the data and the output centroids and returns the clusters.

Value

if fuzzy = TRUE the function returns a list with two attributes: a vector with the clusters and a matrix with cluster probabilities. Otherwise, it returns a vector with the clusters.

Author(s)

Lampros Mouselimis

Examples

```
data(dietary_survey_IBS)

dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

dat = center_scale(dat)

MbatchKm = MiniBatchKmeans(dat, clusters = 2, batch_size = 20, num_init = 5, early_stop_iter = 10)

pr = predict_MBatchKMeans(dat, MbatchKm$centroids, fuzzy = FALSE)
```

---

predict_Medoids *Predictions for the Medoid functions*

Description

Predictions for the Medoid functions

Usage

```
predict_Medoids(
  data,
  MEDOIDS = NULL,
  distance_metric = "euclidean",
  fuzzy = FALSE,
  minkowski_p = 1,
  threads = 1
)

## S3 method for class 'MedoidsCluster'
predict(object, newdata, fuzzy = FALSE, threads = 1, ...)
```

Arguments

`data`	matrix or data frame
`MEDOIDS`	a matrix of initial cluster medoids (data observations). The rows of the MEDOIDS matrix should be equal to the number of clusters and the columns of the MEDOIDS matrix should be equal to the columns of the data.
`distance_metric`	a string specifying the distance method. One of, *euclidean*, *manhattan*, *chebyshev*, *canberra*, *braycurtis*, *pearson_correlation*, *simple_matching_coefficient*, *minkowski*, *hamming*, *jaccard_coefficient*, *Rao_coefficient*, *mahalanobis*, *cosine*
`fuzzy`	either TRUE or FALSE. If TRUE, then probabilities for each cluster will be returned based on the distance between observations and medoids.
`minkowski_p`	a numeric value specifying the minkowski parameter in case that distance_metric = "minkowski"
`threads`	an integer specifying the number of cores to run in parallel. Openmp will be utilized to parallelize the number of initializations (num_init)
`object, newdata, ...`	arguments for the 'predict' generic

Value

a list with the following attributes will be returned : clusters, fuzzy_clusters (if fuzzy = TRUE), dissimilarity.

Author(s)

Lampros Mouselimis

Examples

```
data(dietary_survey_IBS)

dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]

dat = center_scale(dat)

cm = Cluster_Medoids(dat, clusters = 3, distance_metric = 'euclidean', swap_phase = TRUE)

pm = predict_Medoids(dat, MEDOIDS = cm$medoids, 'euclidean', fuzzy = TRUE)
```

`Silhouette_Dissimilarity_Plot` *Plot of silhouette widths or dissimilarities*

Description

Plot of silhouette widths or dissimilarities

Usage

```
Silhouette_Dissimilarity_Plot(evaluation_object, silhouette = TRUE)
```

Arguments

`evaluation_object`	the output of either a *Cluster_Medoids* or *Clara_Medoids* function
`silhouette`	either TRUE or FALSE, indicating whether the silhouette widths or the dissimilarities should be plotted

Details

This function takes the result-object of the *Cluster_Medoids* or *Clara_Medoids* function and depending on the argument *silhouette* it plots either the dissimilarities or the silhouette widths of the observations belonging to each cluster.

Value

TRUE if either the silhouette widths or the dissimilarities are plotted successfully, otherwise FALSE

Author(s)

Lampros Mouselimis

Examples

```
# data(soybean)

# dat = soybean[, -ncol(soybean)]

# cm = Cluster_Medoids(dat, clusters = 5, distance_metric = 'jaccard_coefficient')

# plt_sd = Silhouette_Dissimilarity_Plot(cm, silhouette = TRUE)
```

`silhouette_of_clusters` *Silhouette width based on pre-computed clusters*

Description

Silhouette width based on pre-computed clusters

Usage

```
silhouette_of_clusters(data, clusters)
```

Arguments

`data`	a matrix or a data frame
`clusters`	a numeric vector which corresponds to the pre-computed clusters (see the example section for more details). The size of the 'clusters' vector must be equal to the number of rows of the input data

Value

a list object where the first sublist is the 'silhouette summary', the second sublist is the 'silhouette matrix' and the third sublist is the 'global average silhouette' (based on the silhouette values of all observations)

Author(s)

Lampros Mouselimis

Examples

```
data(dietary_survey_IBS)
dat = dietary_survey_IBS[, -ncol(dietary_survey_IBS)]
dat = center_scale(dat)

clusters = 2

# compute k-means
km = KMeans_rcpp(dat, clusters = clusters, num_init = 5, max_iters = 100, initializer = 'kmeans++')

# compute the silhouette width
silh_km = silhouette_of_clusters(data = dat, clusters = km$clusters)

# silhouette summary
silh_summary = silh_km$silhouette_summary

# silhouette matrix (including cluster & dissimilarity)
silh_mtrx = silh_km$silhouette_matrix
```

```
# global average silhouette
glob_avg = silh_km$silhouette_global_average
```

---

soybean *The soybean (large) data set from the UCI repository*

Description

There are 19 classes, only the first 15 of which have been used in prior work. The folklore seems to be that the last four classes are unjustified by the data since they have so few examples. There are 35 categorical attributes, some nominal and some ordered. The value 'dna' means does not apply. The values for attributes are encoded numerically, with the first value encoded as '0', the second as '1', and so forth. Unknown values were imputated using the mice package.

Usage

```
data(soybean)
```

Format

A data frame with 307 Instances and 36 attributes (including the class attribute, "class")

Details

The column names of the data (including the class) appear in the following order:

date, plant-stand, precip, temp, hail, crop-hist, area-damaged, severity, seed-tmt, germination, plant-growth, leaves, leafspots-halo, leafspots-marg, leafspot-size, leaf-shread, leaf-malf, leaf-mild, stem, lodging, stem-cankers, canker-lesion, fruiting-bodies, external decay, mycelium, int-discolor, sclerotia, fruit-pods, fruit spots, seed, mold-growth, seed-discolor, seed-size, shriveling, roots, class

References

R.S. Michalski and R.L. Chilausky, Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis, International Journal of Policy Analysis and Information Systems, Vol. 4, No. 2, 1980.

Donor: Ming Tan & Jeff Schlimmer (Jeff.Schlimmer cs.cmu.edu)

download source: https://archive.ics.uci.edu/ml/datasets/Soybean+(Large)

Examples

```
data(soybean)

X = soybean[, -ncol(soybean)]

y = soybean[, ncol(soybean)]
```

Index