Package 'genieclust'

April 22, 2021

Type Package

Title The Genie++ Hierarchical Clustering Algorithm with Noise Points Detection

Version 1.0.0

Date 2021-04-22

Description A retake on the Genie algorithm - a robust hierarchical clustering method (Gagolewski, Bartoszuk, Cena, 2016 <DOI:10.1016/j.ins.2016.05.003>). Now faster and more memory efficient; determining the whole hierarchy for datasets of 10M points in low dimensional Euclidean spaces or 100K points in high-dimensional ones takes only 1-2 minutes. Allows clustering with respect to mutual reachability distances so that it can act as a noise point detector or a robustified version of 'HDBSCAN*' (that is able to detect a predefined number of clusters and hence it does not dependent on the somewhat fragile 'eps' parameter). The package also features an implementation of economic inequity indices (the Gini, Bonferroni index) and external cluster validity measures (partition similarity scores; e.g., the adjusted Rand, Fowlkes-Mallows, adjusted mutual information, pair sets index). See also the 'Python' version of 'genieclust' available on 'PyPI', which supports sparse data, more metrics, and even larger datasets.

BugReports https://github.com/gagolews/genieclust/issues

URL https://genieclust.gagolewski.com/

License AGPL-3

Imports Rcpp (>= 1.0.4), stats, utils

Suggests datasets, mlpack

LinkingTo Rcpp

Encoding UTF-8

SystemRequirements OpenMP, C++11

RoxygenNote 7.1.1

NeedsCompilation yes

Author Marek Gagolewski [aut, cre, cph]
(<https://orcid.org/0000-0003-0637-6028>),
Maciej Bartoszuk [ctb],
Anna Cena [ctb],
Peter M. Larsen [ctb]

Maintainer Marek Gagolewski <marek@gagolewski.com>

Repository CRAN

Date/Publication 2021-04-22 07:20:08 UTC

R topics documented:

`genieclust-package` *The Genie++ Hierarchical Clustering Algorithm (with Extras)*

Description

See `genie()` for more details.

Author(s)

Marek Gagolewski

`adjusted_rand_score` *Pairwise Partition Similarity Scores*

Description

Let `x` and `y` represent two partitions of a set of n elements into, respectively, K and L nonempty and pairwise disjoint subsets. For instance, these can be two clusterings of a dataset with n observations specified by two vectors of labels. The functions described in this section quantify the similarity between `x` and `y`. They can be used as external cluster validity measures, i.e., in the presence of reference (ground-truth) partitions.

Usage

```
adjusted_rand_score(x, y = NULL)

rand_score(x, y = NULL)

adjusted_fm_score(x, y = NULL)

fm_score(x, y = NULL)

mi_score(x, y = NULL)

normalized_mi_score(x, y = NULL)

adjusted_mi_score(x, y = NULL)

normalized_accuracy(x, y = NULL)

pair_sets_index(x, y = NULL)
```

Arguments

x	an integer vector of length n (or an object coercible to) representing a K-partition of an n-set, or a confusion matrix with K rows and L columns (see `table(x,y)`)
y	an integer vector of length n (or an object coercible to) representing an L-partition of the same set), or NULL (if x is an K*L confusion matrix)

Details

Every index except `mi_score()` (which computes the mutual information score) outputs 1 given two identical partitions. Note that partitions are always defined up to a bijection of the set of possible labels, e.g., (1, 1, 2, 1) and (4, 4, 2, 4) represent the same 2-partition.

`rand_score()` gives the Rand score (the "probability" of agreement between the two partitions) and `adjusted_rand_score()` is its version corrected for chance, see (Hubert, Arabie, 1985), its expected value is 0.0 given two independent partitions. Due to the adjustment, the resulting index might also be negative for some inputs.

Similarly, `fm_score()` gives the Fowlkes-Mallows (FM) score and `adjusted_fm_score()` is its adjusted-for-chance version, see (Hubert, Arabie, 1985).

Note that both the (unadjusted) Rand and FM scores are bounded from below by $1/(K+1)$ if $K = L$, hence their adjusted versions are preferred.

`mi_score()`, `adjusted_mi_score()` and `normalized_mi_score()` are information-theoretic scores, based on mutual information, see the definition of AMI_{sum} and NMI_{sum} in (Vinh et al., 2010).

`normalized_accuracy()` is defined as $(Accuracy(C_\sigma) - 1/L)/(1 - 1/L)$, where C_σ is a version of the confusion matrix for given `x` and `y`, $K \leq L$, with columns permuted based on the solution to the Maximal Linear Sum Assignment Problem. $Accuracy(C_\sigma)$ is sometimes referred to as Purity, e.g., in (Rendon et al. 2011).

`pair_sets_index()` gives the Pair Sets Index (PSI) adjusted for chance (Rezaei, Franti, 2016), $K \leq L$. Pairing is based on the solution to the Linear Sum Assignment Problem of a transformed version of the confusion matrix.

Value

A single real value giving the similarity score.

References

Hubert L., Arabie P., Comparing Partitions, Journal of Classification 2(1), 1985, 193-218, esp. Eqs. (2) and (4).

Rendon E., Abundez I., Arizmendi A., Quiroz E.M., Internal versus external cluster validation indexes, International Journal of Computers and Communications 5(1), 2011, 27-34.

Rezaei M., Franti P., Set matching measures for external cluster validity, IEEE Transactions on Knowledge and Data Mining 28(8), 2016, 2173-2186.

Vinh N.X., Epps J., Bailey J., Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance, Journal of Machine Learning Research 11, 2010, 2837-2854.

Examples

```
y_true <- iris[[5]]
y_pred <- kmeans(as.matrix(iris[1:4]), 3)$cluster
adjusted_rand_score(y_true, y_pred)
rand_score(table(y_true, y_pred)) # the same
adjusted_fm_score(y_true, y_pred)
fm_score(y_true, y_pred)
mi_score(y_true, y_pred)
normalized_mi_score(y_true, y_pred)
adjusted_mi_score(y_true, y_pred)
normalized_accuracy(y_true, y_pred)
pair_sets_index(y_true, y_pred)
```

---

emst_mlpack *Euclidean Minimum Spanning Tree*

Description

Provides access to the implementation of the Dual-Tree Borůvka algorithm from the `mlpack` package (if available). It is based on kd-trees and is fast for (very) low-dimensional Euclidean spaces. For higher dimensional spaces (say, over 5 features) or other metrics, use the parallelised Prim-like algorithm implemented in `mst()`.

Usage

```
emst_mlpack(X, leaf_size = 1, naive = FALSE, verbose = FALSE)
```

Arguments

X	a numeric matrix (or an object coercible to one, e.g., a data frame with numeric-like columns)
leaf_size	size of leaves in the kd-tree, controls the trade-off between speed and memory consumption
naive	logical; whether to use the naive, quadratic-time algorithm
verbose	logical; whether to print diagnostic messages

Value

An object of class mst, see mst() for details.

References

March W.B., Ram P., Gray A.G., Fast Euclidean Minimum Spanning Tree: Algorithm, Analysis, and Applications, Proc. ACM SIGKDD'10 (2010) 603-611, https://mlpack.org/papers/emst.pdf.

Curtin R.R., Edel M., Lozhnikov M., Mentekidis Y., Ghaisas S., Zhang S., mlpack 3: A fast, flexible machine learning library, Journal of Open Source Software 3(26), 726, 2018.

---

gclust *The Genie++ Hierarchical Clustering Algorithm*

Description

A reimplementation of *Genie* - a robust and outlier resistant clustering algorithm (see Gagolewski, Bartoszuk, Cena, 2016). The Genie algorithm is based on a minimum spanning tree (MST) of the pairwise distance graph of a given point set. Just like single linkage, it consumes the edges of the MST in increasing order of weights. However, it prevents the formation of clusters of highly imbalanced sizes; once the Gini index (see gini_index()) of the cluster size distribution raises above gini_threshold, a forced merge of a point group of the smallest size is performed. Its appealing simplicity goes hand in hand with its usability; Genie often outperforms other clustering approaches on benchmark data, such as https://github.com/gagolews/clustering_benchmarks_v1.

The clustering can now also be computed with respect to the mutual reachability distance (based, e.g., on the Euclidean metric), which is used in the definition of the HDBSCAN* algorithm (see Campello et al., 2015). If M > 1, then the mutual reachability distance $m(i,j)$ with smoothing factor M is used instead of the chosen "raw" distance $d(i,j)$. It holds $m(i,j) = \max(d(i,j), c(i), c(j))$, where $c(i)$ is $d(i,k)$ with k being the (M-1)-th nearest neighbour of i. This makes "noise" and "boundary" points being "pulled away" from each other.

The Genie correction together with the smoothing factor M > 1 (note that M = 2 corresponds to the original distance) gives a robustified version of the HDBSCAN* algorithm that is able to detect a predefined number of clusters. Hence it does not dependent on the DBSCAN's somewhat magical eps parameter or the HDBSCAN's min_cluster_size one.

Usage

```
gclust(d, ...)

## Default S3 method:
gclust(
  d,
  gini_threshold = 0.3,
  distance = c("euclidean", "l2", "manhattan", "cityblock", "l1", "cosine"),
  cast_float32 = TRUE,
  verbose = FALSE,
  ...
)

## S3 method for class 'dist'
gclust(d, gini_threshold = 0.3, verbose = FALSE, ...)

## S3 method for class 'mst'
gclust(d, gini_threshold = 0.3, verbose = FALSE, ...)

genie(d, ...)

## Default S3 method:
genie(
  d,
  k,
  gini_threshold = 0.3,
  distance = c("euclidean", "l2", "manhattan", "cityblock", "l1", "cosine"),
  M = 1L,
  postprocess = c("boundary", "none", "all"),
  detect_noise = M > 1L,
  cast_float32 = TRUE,
  verbose = FALSE,
  ...
)

## S3 method for class 'dist'
genie(
  d,
  k,
  gini_threshold = 0.3,
  M = 1L,
  postprocess = c("boundary", "none", "all"),
  detect_noise = M > 1L,
  verbose = FALSE,
  ...
)

## S3 method for class 'mst'
```

```
genie(
  d,
  k,
  gini_threshold = 0.3,
  postprocess = c("boundary", "none", "all"),
  detect_noise = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

d	a numeric matrix (or an object coercible to one, e.g., a data frame with numeric-like columns) or an object of class dist, see dist or an object of class mst, see mst().
...	further arguments passed to other methods.
gini_threshold	threshold for the Genie correction, i.e., the Gini index of the cluster size distribution; Threshold of 1.0 disables the correction. Low thresholds highly penalise the formation of small clusters.
distance	metric used to compute the linkage, one of: "euclidean" (synonym: "l2"), "manhattan" (a.k.a. "l1" and "cityblock"), "cosine".
cast_float32	logical; whether to compute the distances using 32-bit instead of 64-bit precision floating-point arithmetic (up to 2x faster).
verbose	logical; whether to print diagnostic messages and progress information.
k	the desired number of clusters to detect, k = 1 with M > 1 acts as a noise point detector.
M	smoothing factor; M <= 2 gives the selected distance; otherwise, the mutual reachability distance is used.
postprocess	one of "boundary" (default), "none" or "all"; in effect only if M > 1. By default, only "boundary" points are merged with their nearest "core" points (A point is a boundary point if it is a noise point and it's amongst its adjacent vertex's M-1 nearest neighbours). To force a classical k-partition of a data set (with no notion of noise), choose "all".
detect_noise	whether the minimum spanning tree's leaves should be marked as noise points, defaults to TRUE if M > 1 for compatibility with HDBSCAN*.

Details

Note that as in the case of all the distance-based methods, the standardisation of the input features is definitely worth giving a try.

If d is a numeric matrix or an object of class dist, mst() will be called to compute an MST, which generally takes at most $O(n^2)$ time (the algorithm we provide is parallelised, environment variable OMP_NUM_THREADS controls the number of threads in use). However, see emst_mlpack() for a very fast alternative in the case of Euclidean spaces of (very) low dimensionality and M = 1.

Given an minimum spanning tree, the algorithm runs in $O(n\sqrt{n})$ time. Therefore, if you want to test different gini_thresholds, (or ks), it is best to explicitly compute the MST first.

According to the algorithm's original definition, the resulting partition tree (dendrogram) might violate the ultrametricity property (merges might occur at levels that are not increasing w.r.t. a between-cluster distance). Departures from ultrametricity are corrected by applying `height = rev(cummin(rev(height)))`.

Value

`gclust()` computes the whole clustering hierarchy; it returns a list of class `hclust`, see `hclust`. Use `link{cutree}()` to obtain an arbitrary k-partition.

`genie()` returns a k-partition - a vector with elements in 1,...,k, whose i-th element denotes the i-th input point's cluster identifier. Missing values (`NA`) denote noise points (if `detect_noise` is `TRUE`).

References

Gagolewski M., Bartoszuk M., Cena A., Genie: A new, fast, and outlier-resistant hierarchical clustering algorithm, *Information Sciences* 363, 2016, 8-23.

Campello R., Moulavi D., Zimek A., Sander J., Hierarchical density estimates for data clustering, visualization, and outlier detection, ACM Transactions on Knowledge Discovery from Data 10(1), 2015, 5:1–5:51.

See Also

`mst()` for the minimum spanning tree routines.

`adjusted_rand_score()` (amongst others) for external cluster validity measures (partition similarity scores).

Examples

```
library("datasets")
data("iris")
X <- iris[1:4]
h <- gclust(X)
y_pred <- cutree(h, 3)
y_test <- iris[,5]
plot(iris[,2], iris[,3], col=y_pred,
   pch=as.integer(iris[,5]), asp=1, las=1)
adjusted_rand_score(y_test, y_pred)
pair_sets_index(y_test, y_pred)

# Fast for low-dimensional Euclidean spaces:
h <- gclust(emst_mlpack(X))
```

gini_index *Inequity (Inequality) Measures*

Description

`gini_index()` gives the normalised Gini index and `bonferroni_index()` implements the Bonferroni index.

Usage

```
gini_index(x)

bonferroni_index(x)
```

Arguments

`x`	numeric vector of non-negative values

Details

Both indices can be used to quantify the "inequity" of a numeric sample. They can be perceived as measures of data dispersion. For constant vectors (perfect equity), the indices yield values of 0. Vectors with all elements but one equal to 0 (perfect inequity), are assigned scores of 1. Both indices follow the Pigou-Dalton principle (are Schur-convex): setting $x_i = x_i - h$ and $x_j = x_j + h$ with $h > 0$ and $x_i - h \geq x_j + h$ (taking from the "rich" and giving to the "poor") decreases the inequity.

These indices have applications in economics, amongst others. The Gini clustering algorithm uses the Gini index as a measure of the inequality of cluster sizes.

The normalised Gini index is given by:

$$G(x_1, \ldots, x_n) = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} |x_i - x_j|}{(n-1) \sum_{i=1}^{n} x_i}.$$

The normalised Bonferroni index is given by:

$$B(x_1, \ldots, x_n) = \frac{\sum_{i=1}^{n} (n - \sum_{j=1}^{i} \frac{n}{n-j+1}) x_{\sigma(n-i+1)}}{(n-1) \sum_{i=1}^{n} x_i}.$$

Time complexity: $O(n)$ for sorted (increasingly) data. Otherwise, the vector will be sorted.

In particular, for ordered inputs, it holds:

$$G(x_1, \ldots, x_n) = \frac{\sum_{i=1}^{n} (n - 2i + 1) x_{\sigma(n-i+1)}}{(n-1) \sum_{i=1}^{n} x_i},$$

where σ is an ordering permutation of $(x_1, \ldots, x_n)$.

Value

The value of the inequity index, a number in $[0, 1]$.

References

Bonferroni C., Elementi di Statistica Generale, Libreria Seber, Firenze, 1930.

Gagolewski M., Bartoszuk M., Cena A., Genie: A new, fast, and outlier-resistant hierarchical clustering algorithm, Information Sciences 363, 2016, pp. 8-23. doi:10.1016/j.ins.2016.05.003

Gini C., Variabilita e Mutabilita, Tipografia di Paolo Cuppini, Bologna, 1912.

Examples

```
gini_index(c(2, 2, 2, 2, 2))  # no inequality
gini_index(c(0, 0, 10, 0, 0)) # one has it all
gini_index(c(7, 0, 3, 0, 0))  # give to the poor, take away from the rich
gini_index(c(6, 0, 3, 1, 0))  # (a.k.a. Pigou-Dalton principle)
bonferroni_index(c(2, 2, 2, 2, 2))
bonferroni_index(c(0, 0, 10, 0, 0))
bonferroni_index(c(7, 0, 3, 0, 0))
bonferroni_index(c(6, 0, 3, 1, 0))
```

---

mst *Minimum Spanning Tree of the Pairwise Distance Graph*

Description

An parallelised implementation of a Jarník (Prim/Dijkstra)-like algorithm for determining a(*) minimum spanning tree (MST) of a complete undirected graph representing a set of n points with weights given by a pairwise distance matrix.

(*) Note that there might be multiple minimum trees spanning a given graph.

Usage

```
mst(d, ...)

## Default S3 method:
mst(
  d,
  distance = c("euclidean", "l2", "manhattan", "cityblock", "l1", "cosine"),
  M = 1L,
  cast_float32 = TRUE,
  verbose = FALSE,
  ...
)

## S3 method for class 'dist'
mst(d, M = 1L, verbose = FALSE, ...)
```

Arguments

d	either a numeric matrix (or an object coercible to one, e.g., a data frame with numeric-like columns) or an object of class dist, see dist.
...	further arguments passed to or from other methods.
distance	metric used to compute the linkage, one of: "euclidean" (synonym: "l2"), "manhattan" (a.k.a. "l1" and "cityblock"), "cosine".
M	smoothing factor; M = 1 gives the selected distance; otherwise, the mutual reachability distance is used.
cast_float32	logical; whether to compute the distances using 32-bit instead of 64-bit precision floating-point arithmetic (up to 2x faster).
verbose	logical; whether to print diagnostic messages and progress information.

Details

If d is a numeric matrix of size np, the $n(n-1)/2$ distances are computed on the fly, so that $O(nM)$ memory is used.

The algorithm is parallelised; set the OMP_NUM_THREADS environment variable Sys.setenv to control the number of threads used.

Time complexity is $O(n^2)$ for the method accepting an object of class dist and $O(pn^2)$ otherwise.

If M >= 2, then the mutual reachability distance $m(i,j)$ with smoothing factor M (see Campello et al. 2015) is used instead of the chosen "raw" distance $d(i,j)$. It holds $m(i,j) = \max(d(i,j), c(i), c(j))$, where $c(i)$ is $d(i,k)$ with k being the (M-1)-th nearest neighbour of i. This makes "noise" and "boundary" points being "pulled away" from each other. Genie++ clustering algorithm (see gclust) with respect to the mutual reachability distance gains the ability to identify some observations are noise points.

Note that the case M = 2 corresponds to the original distance, but we are determining the 1-nearest neighbours separately as well, which is a bit suboptimal; you can file a feature request if this makes your data analysis tasks too slow.

Value

Matrix of class mst with n-1 rows and 3 columns: from, to and dist. It holds from < to. Moreover, dist is sorted nondecreasingly. The i-th row gives the i-th edge of the MST. (from[i],to[i]) defines the vertices (in 1,...,n) and dist[i] gives the weight, i.e., the distance between the corresponding points.

The method attribute gives the name of the distance used. The Labels attribute gives the labels of all the input points.

If M > 1, the nn attribute gives the indices of the M-1 nearest neighbours of each point.

References

Jarník V., O jistém problému minimálním, Práce Moravské Přírodovědecké Společnosti 6 (1930) 57–63.

Olson C.F., Parallel algorithms for hierarchical clustering, Parallel Comput. 21 (1995) 1313–1325.

Prim R., Shortest connection networks and some generalisations, Bell Syst. Tech. J. 36 (1957) 1389–1401.

Campello R., Moulavi D., Zimek A., Sander J., Hierarchical density estimates for data clustering, visualization, and outlier detection, ACM Transactions on Knowledge Discovery from Data 10(1) (2015) 5:1–5:51.

See Also

`emst_mlpack()` for a very fast alternative in case of (very) low-dimensional Euclidean spaces (and `M = 1`).

Examples

```
library("datasets")
data("iris")
X <- iris[1:4]
tree <- mst(X)
```

Index