SoftwareX 15 (2021) 100722

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

genieclust: Fast and robust hierarchical clustering

Marek Gagolewski *

Deakin University, School of Information Technology, Geelong, VIC 3220, Australia
Warsaw University of Technology, Faculty of Mathematics and Information Science, ul. Koszykowa 75, 00-662 Warsaw, Poland

ARTICLE INFO

Article history:
Received 6 September 2020
Received in revised form 23 April 2021
Accepted 21 May 2021

Keywords:
Hierarchical clustering
Robust methods
Noise points
Python
R

ABSTRACT

`genieclust` is an open source Python and R package that implements the hierarchical clustering algorithm called *Genie*. This method frequently outperforms other state-of-the-art approaches in terms of clustering quality and speed, supports various distances over dense, sparse, and string data domains, and can be robustified even further with the built-in noise point detector. As domain-independent software, it can be used for solving problems arising in all data-driven research and development activities, including environmental, health, biological, physical, decision, and social sciences as well as technology and engineering. The Python version provides a `scikit-learn`-compliant API, whereas the R variant is compatible with the classic `hclust()`. Numerous tutorials, use cases, non-trivial examples, documentation, installation instructions, benchmark results and timings can be found at https://genieclust.gagolewski.com/.

Code metadata

Current code version	1.0.0
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-20-00039
Legal Code License	GNU AGPL v3
Code versioning system used	git
Software code languages, tools, and services used	C++11, Python, R
Compilation requirements, operating environments & dependencies	Python 3.7+ with `cython`, `numpy`, `scipy`, `matplotlib`, `nmslib`, `scikit-learn`, or R with `Rcpp`
Link to developer documentation/manual	https://genieclust.gagolewski.com/
Feature requests and bug tracker	https://github.com/gagolews/genieclust/issues

1. Introduction

Genie is a multi-purpose hierarchical clustering algorithm proposed in [1] and based on the following intuitive idea. First, make each individual point the sole member of its own cluster. Then, seek and merge the pairs of the closest clusters, one after another, applying the single linkage criterion except where the Gini index of the cluster size distribution raises above a predefined threshold. In such cases, to prevent the formation of degenerate and imbalanced clusters, match a point group of the smallest size with its nearest neighbour.

Genie is as universal and flexible as all other distance-based hierarchical clustering algorithms (such as the single, average, and Ward linkage). It only requires some similarity measure between a pair of observations. With `genieclust`, partitioning can be performed for dense, sparse, and string data as well as with various metrics, including the Euclidean, Manhattan, cosine, and Levenshtein ones. It also allows for the extraction of any number of clusters.

Genie's appealing simplicity goes hand in hand with its practical usefulness: it often outperforms other methods in terms of both speed and clustering quality. This is evidenced by data in Tables 1 and 2 where we provide some timings gathered on a mid-end laptop and in Table 3 where we quantify the agreement between the reference and the predicted clusterings of 73 benchmark data sets (with respect to the Euclidean metric on untransformed data, using the default algorithms' settings unless stated otherwise).

* Correspondence to: Deakin University, School of Information Technology, Geelong, VIC 3220, Australia.
E-mail address: m.gagolewski@deakin.edu.au.

https://doi.org/10.1016/j.softx.2021.100722

Table 1
Time (in seconds) to determine the complete cluster hierarchy for n randomly generated points (2 Gaussian blobs with i.i.d. coordinates) in $\mathbb{R}^{100}$; `genieclust` with the exact and approximate (based on fast nearest neighbour search using `nmslib`) version of Genie (6 threads).

Method	$n =$	10,000	50,000	100,000	500,000	1,000,000
Genie		1.00	26.07	132.47	4408.07	16,021.80
Genie (*exact* = *False*)		0.48	4.60	12.60	113.37	266.64

Table 2
Time (in seconds, 6 threads except for `fastcluster` which is single-threaded) to cluster the Fashion-MNIST data set (https://github.com/zalandoresearch/fashion-mnist; 70,000 points in $\mathbb{R}^{784}$); for all the hierarchical methods, extraction of any k-partition takes a similar amount of time; `genieclust` is particularly suited for solving of *extreme clustering* tasks (large data sets, many clusters, see [2]).

Method	$k =$	10	100	1000
Genie		445.81	445.86	446.32
Genie (*exact* = *False*)		38.02	38.03	38.03
k-means (`scikit-learn`)		24.90	225.04	1745.88
Ward's linkage (`fastcluster`)		4757.32	4757.39	4757.63

2. The genieclust package

The reference (basic) implementation of the algorithm introduced in [1] was available in form of an R-only package `genie`. Its superseder, the `genieclust` package (currently in version 1.0.0) has been rewritten from scratch. It brings many new features and is optimised for speed.

The Python 3.7+ version[1] of `genieclust`, which can be installed via `pip`, has an intuitive `scikit-learn`-compatible [3] interface:

```
import genieclust
X = ...                              # inputs (e.g., a numeric matrix)
g = genieclust.Genie(n_clusters=2) # provide method parameters here
labels_pred = g.fit_predict(X)     # generate the label vector
```

The R variant[2] is available via a call to `install.packages("genieclust")`. Users of the built-in `hclust()` function will find what follows very familiar:

```
library("genieclust")
X <- ...       # some data
h <- gclust(X) # no need to precompute the distance matrix with dist(X)
cutree(h, k=2) # extract the partition; alternatively: call genie(X, k=2)
plot(h)        # plot the cluster dendrogram
```

The core algorithm has been written in form of a header-only C++ library and relies on portable input and output formats (barebone C- and Fortran-contiguous arrays). Hence, new bindings can easily be added in the future, e.g., for Julia or Matlab. The new implementation of Genie now runs in amortised $O(n\sqrt{n})$ time and requires $O(n)$ memory provided that it is fed with a spanning forest of the graph representing the n input points. By default, the `genieclust` package computes the Euclidean minimum spanning tree based on a parallelised version of Prim's algorithm (see [4]) or, if `mlpack` [5] is available, a variant of a dual-tree Borůvka's method [6], which is very fast in low-dimensional spaces. The Python version of the package also offers an option for approximating the MST (*exact=False*) by means of the near-neighbour search routines offered by `nmslib` [7] which also supports sparse and string data.

For more details, the reader is kindly referred to the user-centric project website located at https://genieclust.gagolewski.com/. It contains numerous tutorials, use cases, non-trivial examples, detailed documentation of the package's API, installation instructions as well as reports on the benchmark results and timings.

3. Additional features

The *gini_threshold* parameter can be adjusted if the underlying cluster structure is expected to be imbalanced (e.g., featuring many small clusters and few large ones). The paper [1] recommends the threshold of 0.3, which is the default in `genieclust`, but 0.1 and 0.5 are often worth inspecting too.

The package also allows for clustering with respect to mutual reachability distances,[3] d_M, known from the HDBSCAN* algorithm [16]. The smoothing factor, M, controls how eagerly the points tend to be classified as noise. However, contrary to an excellent implementation of HDBSCAN* that is featured in the `hdbscan` package for Python [17] and which also relies on a minimum spanning tree with respect to d_M, we still have the robust hierarchical Genie algorithm underneath here. This means we can always fetch exactly the requested number of clusters and that partitions of finer granularity are properly nested within the coarser ones; by exploring the determined hierarchy we can quickly and easily get insight into the underlying structure of each data set. Nevertheless, `hdbscan` might be preferred to `genieclust` where a more automated discovery of the partition cardinality is needed.

The `genieclust` package also includes an implementation of many partition similarity scores that can be used as external cluster validity measures, e.g., adjusted Rand index, Fowlkes–Mallows index, adjusted and normalised mutual information scores [18], normalised accuracy (purity), and pair sets index, the latter two being based on the solution to the linear sum assignment problem of a transformed version of the confusion matrix (which makes them nicely interpretable), see [19].

4. Conclusions

`genieclust` is an open source project to which everyone is welcome to contribute. It is distributed under the terms of the free, copyleft GNU AGPL v3 license. Its development page at https://github.com/gagolews/genieclust includes a code repository managed with `git`, a bug and feature request tracker, and many options for other users to engage in the project (e.g., some tasks are marked as "good first issues"). Continuous Integration (CI)-related hooks have been set up to assure high source code quality (automated unit testing, code coverage estimation, linting, etc.) and wide portability (builds are run on various flavours of Windows, Linux, and macOS).

Forthcoming versions of the package will support more distance metrics and input data types, connectivity matrices, and

[1] The Python version of `genieclust` has been built with Cython and relies on `numpy`, `scipy`, `scikit-learn`, `matplotlib`, and `nmslib`. Platform independent sources and binary wheels for Linux, Windows, and macOS are available for download from PyPI, see https://pypi.org/project/genieclust/.

[2] The R version of `genieclust` binds to our header-only C++ library by means of `Rcpp`. Platform independent sources and binary builds for Windows and macOS can be downloaded from CRAN, see https://cran.r-project.org/web/packages/genieclust/.

[3] Formally, the mutual reachability distance is given for $M > 1$ by $d_M(i,j) = \max(d(i,j), c_M(i), c_M(j))$, where $d(i,j)$ is the "raw" (e.g., Euclidean) distance between the ith and the jth inputs and the "core" distance $c_M(i)$ is given by $d(i,k)$ with k being the $(M-1)$th nearest neighbour of i. As argued in [16], this makes "noise" and "boundary" points being "pulled away" from each other.

In `genieclust`, during the clustering procedure, all leaves of a minimum spanning tree with respect to d_M do not take part in the clustering process. They may be merged with the nearest clusters during postprocessing, or be left marked as "noise" observations. For more details, refer to the package documentation.

Table 3
Distribution of the adjusted Rand scores across 73 data sets from the *Benchmark Suite for Clustering Algorithms* [8] which aggregates data from various sources, including [9–12]; 1.0 designates perfect agreement between the obtained clustering and the reference one. We applied a few classical agglomerative hierarchical methods (implemented in the `fastcluster` package; [13]), *k*-means, expectation–maximisation (EM) for Gaussian mixtures, Birch, Spectral (implemented in `scikit-learn`; [14]), ITM (package `itm`; [15]), and Genie, which surpasses all of the former methods. See https://genieclust.gagolewski.com/ for more information on the experiment set-up, detailed results, and discussion.

Method	AR=	Mean	st.dev.	Min	25%	50%	75%	Max
Genie (*gini_threshold* = 0.3)		**0.77**	**0.28**	0	**0.59**	**0.92**	**1**	1
Genie (*gini_threshold* = 0.5)		0.76	0.31	0	0.56	0.88	**1**	1
ITM		0.68	0.27	0	0.53	0.69	0.99	1
Gaussian mixtures (*n_init* = 100)		0.65	0.37	0	0.40	0.82	0.98	1
Spectral (*affinity* = *RBF*)		0.63	0.37	0	0.33	0.73	0.99	1
Birch (*threshold* = 0.01)		0.54	0.35	0	0.19	0.54	0.90	1
k-means		0.54	0.35	0	0.20	0.51	0.89	1
Ward's linkage		0.53	0.34	0	0.19	0.54	0.91	1
Average linkage		0.50	0.38	0	0.11	0.50	0.92	1
Complete linkage		0.47	0.34	0	0.21	0.40	0.78	1
Single linkage		0.44	0.45	0	0	0.32	**1**	1

new heuristics to robustify the generated clusterings even further.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The author would like to thank Anna Cena and Maciej Bartoszuk for comments, discussion, and remarks. This research was partially supported by the Australian Research Council Discovery Project ARC DP 210100227.

References

[1] Gagolewski M, Bartoszuk M, Cena A. Genie: A new, fast, and outlier-resistant hierarchical clustering algorithm. Inform Sci 2016;363:8–23. http://dx.doi.org/10.1016/j.ins.2016.05.003.
[2] Kobren A, Monath N, Krishnamurthy A, McCallum A. A hierarchical algorithm for extreme clustering. In: Proc. 23rd ACM SIGKDD'17. 2017, p. 255–64. http://dx.doi.org/10.1145/3097983.3098079.
[3] Buitinck L, et al. API design for machine learning software: Experiences from the scikit-learn project. In: ECML PKDD workshop: Languages for data mining and machine learning. 2013, p. 108–22.
[4] Olson CF. Parallel algorithms for hierarchical clustering. Parallel Comput 1995;21:1313–25. http://dx.doi.org/10.1016/0167-8191(95)00017-I.
[5] Curtin RR, Edel M, Lozhnikov M, Mentekidis Y, Ghaisas S, Zhang S. Mlpack 3: A fast, flexible machine learning library. J Open Source Softw 2018;3(26):726. http://dx.doi.org/10.21105/joss.00726.
[6] March WB, Ram P, Gray AG. Fast euclidean minimum spanning tree: Algorithm, analysis, and applications. In: Proc. 16th ACM SIGKDD'10. 2010, p. 603–12. http://dx.doi.org/10.1145/1835804.1835882.
[7] Naidan B, Boytsov L, Malkov Y, Novak D. Non-metric space library (NMSLIB) manual, version 2.0. 2019, URL https://github.com/nmslib/nmslib/blob/master/manual/latex/manual.pdf.
[8] Gagolewski M, et al. Benchmark suite for clustering algorithms – version 1. 2020, http://dx.doi.org/10.5281/zenodo.3815066, URL https://github.com/gagolews/clustering_benchmarks_v1.
[9] Dua D, Graff C. UCI machine learning repository. 2019, URL http://archive.ics.uci.edu/ml.
[10] Fränti P, Sieranoja S. K-means properties on six clustering benchmark datasets. Appl Intell 2018;48(12):4743–59. http://dx.doi.org/10.1007/s10489-018-1238-7.
[11] Ultsch A. Clustering with SOM: U*c. In: Workshop on self-organizing maps. 2005, p. 75–82.
[12] Graves D, Pedrycz W. Kernel-based fuzzy clustering and fuzzy clustering: A comparative experimental study. Fuzzy Sets and Systems 2010;161:522–43. http://dx.doi.org/10.1016/j.fss.2009.10.021.
[13] Müllner D. Fastcluster: Fast hierarchical, agglomerative clustering routines for R and Python. J Stat Softw 2013;53(9):1–18. http://dx.doi.org/10.18637/jss.v053.i09.
[14] Pedregosa F, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12(85):2825–30, URL http://jmlr.org/papers/v12/pedregosa11a.html.
[15] Müller AC, Nowozin S, Lampert CH. Information theoretic clustering using minimum spanning trees. In: Proc. German conference on pattern recognition. 2012, URL https://github.com/amueller/information-theoretic-mst.
[16] Campello RJGB, Moulavi D, Zimek A, Sander J. Hierarchical density estimates for data clustering, visualization, and outlier detection. ACM Trans Knowl Discov Data 2015;10(1):5:1–51. http://dx.doi.org/10.1145/2733381.
[17] McInnes L, Healy J, Astels S. Hdbscan: Hierarchical density based clustering. J Open Source Softw 2017;2(11):205. http://dx.doi.org/10.21105/joss.00205.
[18] Vinh NX, Epps J, Bailey J. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. J Mach Learn Res 2010;11(95):2837–54, URL http://jmlr.org/papers/v11/vinh10a.html.
[19] Rezaei M, Fränti P. Set matching measures for external cluster validity. IEEE Trans Knowl Data Eng 2016;28(8):2173–86. http://dx.doi.org/10.1109/TKDE.2016.2551240.