Journal of Machine Learning Research 27 (2026) 1-7 Submitted 3/26; Revised 7/26; Published 10/26

CLASSIX: A High-Performance Library for Explainable Clustering

Xinye Chen CHENXY1110@CHINATELECOM.CN
China Telecom Cloud Technology Co., Ltd.
Beijing, China

Stefan Güttel STEFAN.GUETTEL@MANCHESTER.AC.UK
Department of Mathematics
The University of Manchester
Manchester, United Kingdom

Kaustubh Roy KAUSTUBH.ROY@MANCHESTER.AC.UK
Department of Mathematics
The University of Manchester
Manchester, United Kingdom

Editor: Zeyi Wen

Abstract

We present CLASSIX, an open-source Python library for fast and explainable clustering. The library provides a unified interface to three metric-specific implementations, compiled acceleration with portable Python fallbacks, and extensive user and developer documentation. CLASSIX accepts NumPy arrays and Pandas DataFrames and is available under the MIT license at `https://github.com/nla-group/classix`.

Keywords: Clustering, Explainability, Unsupervised learning, Open-Source, Python

1. Introduction

Clustering is an important task in data science and engineering, arising in applications such as computer vision (Cheng, 1995) and bioinformatics (Song and Zhong, 2020). Clustering methods are generally distance- or density-based, with many interpretations of what constitutes a "good clustering" dependent on the application and the evaluation metric used (Estivill-Castro, 2002). One of the most widely used distance-based clustering methods is k-means (Lloyd, 1982). While easy to use and often very efficient, this iterative method is limited to identifying clusters of approximately spherical shape, requires an a-priori guess of the number of clusters, and can be sensitive to initialization. On the other hand, density-based clustering methods such as DBSCAN (Ester et al., 1996) can identify clusters of arbitrary shape, but they usually assume that a global density threshold exists in the distribution of data, leading to failures on data with clusters of varying density. Furthermore, many algorithms, including DBSCAN, can become significantly less efficient as the number of data points increases because of the cost of pairwise distance computations. Most clustering algorithms, including the two just mentioned, do not provide any means to explain or justify the computed clustering results.

©2026 Xinye Chen, Stefan Güttel, and Kaustubh Roy.

License: CC-BY 4.0, see `https://creativecommons.org/licenses/by/4.0/`. Attribution requirements are provided at `http://jmlr.org/papers/v27/26-1057.html`.

The CLASSIX method introduced in (Chen and Güttel, 2024) seeks to overcome the aforementioned drawbacks by combining ingredients of both distance and density-based clustering. CLASSIX employs a two-phase clustering strategy, with an aggregation and a merging phase, combined with an early-search termination criterion based on sorting. CLASSIX clustering can currently be performed in the Euclidean (ℓ^2-norm), Manhattan (ℓ^1-norm), or Tanimoto (or Jaccard) metric (Güttel and Roy, 2026). The most appealing aspects of CLASSIX are its speed, low memory requirement, and explainability. The method is controlled by two main parameters, a `radius` hyperparameter controlling the coarseness of clusters, and a `minPts` parameter for dealing with outliers. Unlike many other clustering algorithms, the two-phase design of CLASSIX means that the parameter choices for `radius` and `minPts` are largely decoupled, significantly streamlining hyperparameter tuning.

The performance of CLASSIX in comparison to HDBSCAN (Campello et al., 2013), Quickshift++ (Jiang et al., 2018), and `k-means++` (Lloyd, 1982), on a dataset representing 5.7 million RNA sequences of coronavirus, is demonstrated in Figure 1, showcasing its superior scalability and accuracy with respect to the adjusted Rand index (ARI; Hubert and Arabie, 1985) and adjusted mutual information (AMI; Vinh et al., 2010).

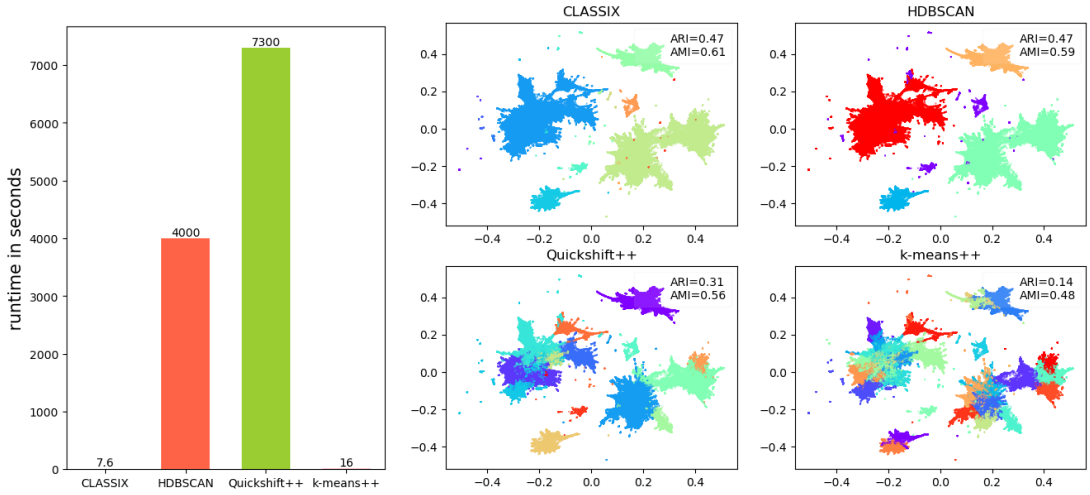


Figure 1: Performance on the Covid3MC dataset containing 5.7 million data points.

2. Software contribution

Chen and Güttel (2024) offered a basic proof-of-concept Python implementation of CLASSIX including the core explainability functionality. The present contribution packages this method into a versioned, cross-platform Python library that supports multiple distance metrics. The latest library version, CLASSIX 1.5.3, is fully aligned with scikit-learn's interface to clustering algorithms (Pedregosa et al., 2011), allowing for seamless user adoption.

Among the most recent functional additions is a new `metric` property allowing CLASSIX clustering in the `euclidean`, `manhattan`, or `tanimoto` distance. Changing the metric requires different aggregation and merging strategies under-the-hood, but the user does not need to be concerned with these technicalities.

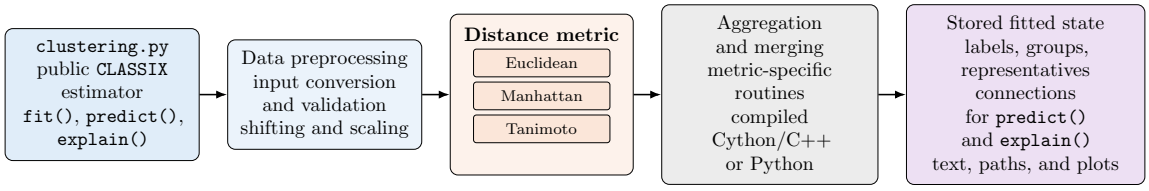


Figure 2: Modular design of CLASSIX. The public interface in CLASSIX validates and preprocesses the input, selects the Euclidean, Manhattan, or Tanimoto path, and calls compatible aggregation and merging routines. Compiled routines are used when available, with Python implementations as fallbacks. The returned groups, representatives, and labels are stored as fitted state and reused by `predict()` and `explain()`.

Another recent addition is a new method called `changeMinPts`, which allows the user to change the `minPts` hyperparameter without redoing the aggregation and merging phases, which typically take most of the computation time. This can significantly speed up hyperparameter tuning.

CLASSIX supports a modular software design for multiple distance metrics–the workflow is as depicted in Figure 2. The public `CLASSIX` class offers interfaces of `fit()`, `predict()`, `explain()` methods. When `fit()` is called, the input data are first checked and preprocessed. The selected metric then determines whether the Euclidean, Manhattan, or Tanimoto routines are used for the subsequent steps of aggregation and merging, which are performed using compiled implementations when available and Python implementations otherwise. Finally, the resulting labels, groups, and representative points are stored and reused by `predict()` and `explain()`.

The Euclidean-, Manhattan-, and Tanimoto-based aggregation and merging are kept in separate modules with compatible outputs. Performance-critical routines have Cython implementations, and a C++ sparse matrix–vector kernel supports the Tanimoto path. Our package automatically checks extension availability when constructing the estimator and falls back to Python implementations where needed. Further, the active mode can be inspected with `classix.cython_is_available()` and Cython can be explicitly disabled for debugging. This separation allows metric-specific optimization without changing the public API.

A metric backend must provide sorted aggregation labels, representative indices, the sorting permutation and values, the sorted data, group sizes, and a distance-evaluation count. Its merging routine must map those aggregation groups to final labels and preserve the correspondence with the original input order. These contracts, together with build and test commands, are documented in `https://github.com/nla-group/classix/blob/master/SOFTWARE_GUIDE.md`.

3. Distribution and documentation

CLASSIX is provided open-source in the form of Python and MATLAB implementations. Specifically, the Python version is released under the MIT license through PyPI and conda-

forge. The latest version 1.5.3 declares support for Python 3.8–3.14. Users can install CLASSIX in Python via `pip install classixclustering` in the command line. No user-side compilation is required on the most common architectures. For less common or unsupported architectures, the full source code is available for compilation, provided that a suitably configured `C++` compiler is available. The MATLAB implementation is available from File Exchange[1].

The user documentation includes installation instructions, a quick start, full API reference, tutorials, performance comparisons, and many examples. Developer-facing material is provided in `SOFTWARE_GUIDE.md` and `CONTRIBUTING.md`. These describe module responsibilities, supported inputs and limitations, build and test commands, backend contracts, documentation builds, and procedures for adding new distance metrics.

We permit reuse within proprietary software and welcome contributions of various forms. We encourage raising issues and proposing pull requests on the GitHub repository. References and examples can be found in the README.md file and the official documentation[2]. A comprehensive collection of code examples covering various applications is provided in `https://github.com/nla-group/classix/tree/master/demos`.

4. Usage demonstration

We now showcase the basic use of CLASSIX's clustering and explainability features. CLASSIX features an interface similar to scikit-learn (Buitinck et al., 2013) for ease of use. It comes with some real-world datasets from various domains which the user can import by calling the `classix.loadData` method and using the associated dataset name, e.g.,

```
import classix
data, labels = classix.loadData('Covid3MC') # load a built-in dataset
```

CLASSIX supports clustering of data provided as Pandas DataFrames and NumPy arrays. To perform clustering, users first instantiate a CLASSIX object, `clx = classix.CLASSIX()`, and then fit the data using the `clx.fit()` method. CLASSIX has three main constructor arguments: `radius`, `minPts` (optional), and `merge_scale` (optional). Optionally, to log detailed clustering information, users can use the Boolean variable `verbose`. Here is a simple code example to illustrate the basic use:

```
clx = classix.CLASSIX(radius=0.2, minPts=500, verbose=True)
clx.fit(data)
```

The above code will cluster 5.7 million data points. On a standard 2025 laptop with 8 GB RAM, this will only take about 30 seconds! The output is:

```
CLASSIX clustered 5726839 data points with 3 features. The radius parameter was
set to 0.20 and minPts was set to 500. As the provided data was auto-scaled by a
factor of 1/12.27, points within a radius R=0.20*12.27=2.45 were grouped together.
In total, 71926291 distances were computed (12.6 per data point). This resulted in
274 groups,  each with a unique group center. These 274 groups were subsequently
merged into 25 clusters.
```

1. MATLAB FileExchange: 153451-fast-and-explainable-clustering-with-classix
2. `https://classix.readthedocs.io/en/stable`

After fitting the model, users can view information about the group and cluster assignments by accessing the `clx.groups_` and `clx.labels_` attributes, respectively.

The explain functionality of CLASSIX is implemented in the `clx.explain` method, which supports a single or a pair of objects as input. Querying `clx.explain` using a pair of data points can be done by feeding either the native indices (the indices in the DataFrame or list) or the data vectors themselves as inputs. As an illustration, we consider two data points with indices `'hCoV-19/Norway/6348/2021'` and `'hCoV-19/USA/WY-WYPHL-20146677/2020'`. The example below shows how to call the `clx.explain` method:

```
clx.explain(data,'hCoV-19/Norway/6348/2021','hCoV-19/USA/WY-WYPHL-20146677/2020',
    plot=True)
```

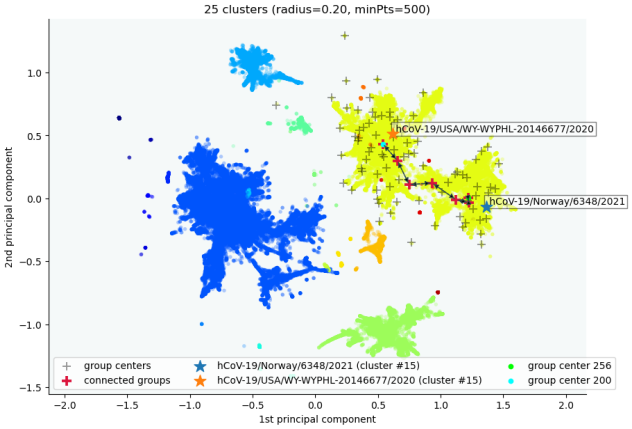


Figure 3: Visual explanation for why two data points `'hCoV-19/Norway/6348/2021'` and `'hCoV-19/USA/WY-WYPHL-20146677/2020'` are in the same cluster. CLASSIX provides a connecting path of data points within the cluster with small steps controlled by the `radius` parameter.

The explain method also provides textual justifications of the computed clusters, outputting the indices of data points that connect two points in a common cluster:

```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=200, centers=2, n_features=2, cluster_std=1.5,
    random_state=42)
clx = CLASSIX(radius=0.5, metric='manhattan', group_merging='distance', minPts=50,
     verbose=0)
clx.explain(X, 3, 10)
# Data point 3 is in group 11. Data point 10 is in group 37.
# Both groups were merged into cluster #1.
# The two groups are connected via groups  11 <-> 18 (minPts) <-> 28 <-> 37.
```

5. Conclusions

CLASSIX is an open-source library implementing a new fast and memory-efficient clustering algorithm and a user-friendly interface for explaining its clustering results both visually

and in textual form. CLASSIX comes with full documentation and a comprehensive set of example demonstrations.

Acknowledgments

Stefan Güttel acknowledges funding from the UK's Engineering and Physical Sciences Research Council (EPSRC grant EP/Z533786/1) and the Royal Society (RS Industry Fellowship IF/R1/231032).

References

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining*, pages 160–172. Springer, 2013.

Xinye Chen and Stefan Güttel. Fast and explainable clustering based on sorting. *Pattern Recognition*, page 110298, 2024. doi: https://doi.org/10.1016/j.patcog.2024.110298.

Yizong Cheng. Mean shift, mode seeking, and clustering. *Transactions on Pattern Analysis and Machine Intelligence*, 17:790–799, 1995.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

Vladimir Estivill-Castro. Why so many clustering algorithms: A position paper. *SIGKDD Explorations Newsletter*, 4(1):65–75, 2002.

Stefan Güttel and Kaustubh Roy. Fast and explainable clustering in the Manhattan and Tanimoto distance, 2026. URL `https://arxiv.org/abs/2601.08781`.

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1): 193–218, 1985. doi: 10.1007/BF01908075.

Heinrich Jiang, Jennifer Jang, and Samory Kpotufe. Quickshift++: Provably good initializations for sample-based mean shift. In *International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2294–2303. PMLR, 2018.

Stuart P. Lloyd. Least squares quantization in PCM. *Transactions on Information Theory*, 28:129–137, 1982.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,

M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Mingzhou Song and Hua Zhong. Efficient weighted univariate clustering maps outstanding dysregulated genomic zones in human cancers. *Bioinformatics*, 36(20):5027–5036, 2020.

Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11:2837–2854, 2010.