

Fast and explainable clustering based on sorting[★]

Xinye Chen^a, Stefan Güttel^{a,*}

^aThe University of Manchester, Manchester, United Kingdom

ARTICLE INFO

Keywords:
Clustering
Fast aggregation
Sorting
Explainability

ABSTRACT

We introduce a fast and explainable clustering method called CLASSIX. It consists of two phases, namely a greedy aggregation phase of the sorted data into groups of nearby data points, followed by the merging of groups into clusters. The algorithm is controlled by two scalar parameters, namely a distance parameter for the aggregation and another parameter controlling the minimal cluster size. Extensive experiments are conducted to give a comprehensive evaluation of the clustering performance on synthetic and real-world datasets, with various cluster shapes and low to high feature dimensionality. Our experiments demonstrate that CLASSIX competes with state-of-the-art clustering algorithms. The algorithm has linear space complexity and achieves near linear time complexity on a wide range of problems. Its inherent simplicity allows for the generation of intuitive explanations of the computed clusters.

1. Introduction

Clustering is a widely-used unsupervised learning technique to find patterns and structure in data. Clustering algorithms group the data points into distinct clusters such that points within a cluster share similar characteristics on the basis of their distance, density or other spatial properties, while points in two distinct clusters are less similar. Applications of clustering methods are wide-ranging, including areas like finance (Bensmail and DeGennaro, 2004), traffic (Erman, Arlitt and Mahanti, 2006), civil engineering (de Oliveira, Garrett and Soibelman, 2011), and bioinformatics (Song and Zhong, 2020). Clustering algorithms can be categorized in different ways, but for our purposes distance-based and density-based clustering methods are the most relevant. Distance-based clustering methods, such as k-means (Lloyd, 1982), consider the pair-wise distance between points in order to decide whether they should be contained in the same cluster. Density-based clustering algorithms, such as DBSCAN (Ester, Kriegel, Sander and Xu, 1996), take a more global view by assuming that data occurs in continuous areas of high density surrounded by low-density regions. A key advantage of many density-based clustering algorithms is that they can handle arbitrarily shaped clusters without specifying the number of clusters in advance. On the other hand, they typically require more parameter tuning.

We propose a novel clustering method called CLASSIX¹ which shares features with both distance and density-based methods. The method comprises two phases: *aggregation* and *merging*. In the aggregation phase, data points are sorted along their first principal component (Hotelling, 1933) and then grouped using a greedy aggregation technique. The sorting is key to traversing the data with almost linear complexity, provided that the number of pairwise distance computations is small. While the initial sorting requires an average-case complexity of $O(n \log n)$, it is only performed on scalar values irrespective of the dimensionality of the data points. As a consequence, the cost of this initial sorting is almost negligible compared to computations on the full-dimensional data. As we will demonstrate experimentally, an almost linear complexity is indeed observed in practice. The aggregation phase is followed by the merging of overlapping groups into clusters using either a distance or density-based criterion. The density-based merging criterion usually results in slightly better clusters than the distance-based criterion, but the latter has a significant speed advantage. CLASSIX depends on two parameters and its tuning is relatively straightforward. In brief, there is a distance parameter `radius` that serves as a tolerance for the grouping in the aggregation phase, while an `minPts` parameter specifies the smallest acceptable cluster size. This is similar to the parameters used in DBSCAN but, owing to the initial sorting of the data points, CLASSIX does not perform spatial range queries for each data point.

[★]S. G. has been supported by a Fellowship of the Alan Turing Institute, EPSRC grant EP/W001381/1.

*Corresponding author

✉ xinye.chen@manchester.ac.uk (X. Chen); stefan.guettel@manchester.ac.uk (S. Güttel)

🌐 <https://personalpages.manchester.ac.uk/staff/stefan.guettel/> (S. Güttel)

ORCID(s): 0000-0003-1778-393X (X. Chen); 0000-0003-1494-4478 (S. Güttel)

¹CLASSIX is a contrived acronym of “CLustering by Aggregation with Sorting-based Indexing” and the letter “X” for “explainability.”

Our comparisons demonstrate that CLASSIX achieves excellent performance against popular methods like k-means++, meanshift, DBSCAN, HDBSCAN*, and Quickshift++ with respect to speed and several metrics of cluster quality. In addition, CLASSIX's inherent simplicity allows for the easy explainability of the clustering results. The Python code available at

<https://github.com/nla-group/classix>

implements CLASSIX with all these features, including the ability to query textual explanations for the clustering.

The rest of this paper is structured as follows. Section 2 reviews some of the existing popular clustering algorithms which share features with CLASSIX. Section 3 introduces the CLASSIX algorithm, starting with the greedy aggregation approach and a discussion of two different merging techniques. We also show how to treat outliers and out-of-sample data. Section 4 is devoted to theoretical considerations of the computational complexity of CLASSIX and how its performance depends on parameters such as the data dimension, Section 5 contains comprehensive performance tests of CLASSIX and comparisons with other popular clustering algorithms. Concluding remarks and a discussion of possible future work are contained in section 6. Finally, the Appendix gives an overview of using the Python implementation and a demonstration of the explainability feature.

2. Related work

CLASSIX shares features with at least two classes of clustering methods, namely distance-based and density-based methods. Here we briefly review some of the most important algorithms in these classes, highlighting the common features shared with CLASSIX and also their differences.

Two of the most popular distance-based clustering algorithms are k-means (Lloyd, 1982) and k-medoids (Kaufman and Rousseeuw, 1990, Chp. 2). These algorithms partition the data into k clusters by iteratively relocating the cluster centers and reassigning data points to their nearest center. As a consequence, these distance-based methods tend to form clusters of spherical shapes and it is generally hard to cluster data with more complicated cluster shapes (Jain, 2010). In terms of computational complexity, it is rather hard to predict how slow or fast k-means performs (Arthur and Vassilvitskii, 2006), but variants like k-means++ (Arthur and Vassilvitskii, 2007), mini-batch k-means (Sculley, 2010), or the contribution in (Elkan, 2003) can significantly improve on the original algorithm in practice. Both k-means and k-medoids require the user to specify k , the expected number of clusters, and this is often not easy in practice.

Density-based clustering methods do not require an a-priori knowledge of the number of clusters. Among these methods are the popular mean shift (Cheng, 1995; Comaniciu and Meer, 2002) and DBSCAN (Ester et al., 1996) algorithms. These methods rely on the property that the density of points at the “core” of a cluster is higher than in its surrounding. Mean shift performs clustering by iteratively shifting data points to nearby peaks (modes) of the empirical density function, using kernel density estimation. Data associated with the same mode are assigned to a cluster. Quick shift (Vedaldi and Soatto, 2008) provides a faster alternative of mean shift clustering using Euclidean medoids as shifts. The recently introduced Quickshift++ further improves clustering performance by first estimating the data modes and providing guarantees that data are assigned to their appropriate cluster mode (Jiang, Jang and Kpotufe, 2018).

The DBSCAN method essentially relies on two parameters, namely the neighborhood radius and a minimum number of data points in every neighborhood. The original implementation uses an R* tree data structure to perform range queries, and the performance of DBSCAN relies crucially on the efficient implementation of such a tree structure. Other trees can be used as well, such as kd-tree or ball tree, but as can be seen in (Kriegel, Schubert and Zimek, 2017) the implementation of such structures is nontrivial. As pointed out by Gan and Tao (2015), DBSCAN has a worst-case time complexity of $O(n^2)$ and a time complexity of $\Omega(n^{4/3})$ if the number of features is greater than three. This problem often is referred to as the curse of dimensionality (Indyk and Motwani, 1998; Har-Peled, Indyk and Motwani, 2012).

Many variants of DBSCAN have been proposed to mitigate the aforementioned issues. OPTICS computes an augmented cluster ordering over a range of parameters and thus provides a versatile basis for both automatic and interactive cluster analysis (Ankerst, Breunig, Kriegel and Sander, 1999; Hahsler, Piekenbrock and Doran, 2019). Gunawan (2013) proposes a DBSCAN variant for two-dimensional data which has a theoretical time complexity of $O(n \log n)$. To fix the limitation left by (Gunawan, 2013), Gan and Tao (2015) introduce the concept of ρ -approximate DBSCAN with an expected linear time complexity $O(n)$ for a small sacrifice in accuracy. However, Schubert, Sander, Ester, Kriegel and Xu (2017) claim that the original DBSCAN can be competitive with ρ -approximate DBSCAN for reasonable parameters choices and effective index structures. There are also parallel implementations like PDSDBSCAN (Patwary, Palsetia, Agrawal, Liao, Manne and Choudhary, 2012). APSCAN (Chen, Liu, Qiu and Lai,

2011) is a parameter-free clustering method that uses affinity propagation (Frey and Dueck, 2007) to infer the local density of a dataset. In such a way, APSCAN can cluster datasets with varying density and preserve the associated nonlinear data structure. HDBSCAN (Campello, Moulavi and Sander, 2013) and HDBSCAN* (Campello, Moulavi, Zimek and Sander, 2015), for simplicity just referred to as “HDBSCAN” here, extend DBSCAN to hierarchical clustering algorithms and then extract a flat clustering based on the stability of clusters. Again, this allows for variable-density clusters and, possibly, a more robust parameter selection. DBSCAN++ (Jang and Jiang, 2019) is a modification of DBSCAN based on the observation that density estimates for merely a subset of $m \ll n$ data points need to be computed, resulting in a speedup of DBSCAN while achieving similar clustering results as DBSCAN. To select the m data points, uniform and greedy k -center-based sampling approaches are proposed.

CLASSIX shares features all of the aforementioned methods. CLASSIX begins with the greedy aggregation of the data into groups, each of which is identified with a so-called starting point. Similar to the neighborhood centers in DBSCAN, the starting points can be interpreted as a reduced-density estimator of the data. However, in CLASSIX the groups are also re-used to cluster the data, not just as estimators. CLASSIX’s two-stage approach also shares similarities with hierarchical clustering methods like HDBSCAN. In the merging phase of CLASSIX, a distance-based criterion can be used, e.g., based on the Euclidean distance. However, as opposed to the distance-based methods discussed above, the merging phase of CLASSIX is non-iterative and does not require a specification of the number of clusters. Instead, the number of clusters is determined by the algorithm based on a `radius` parameter and the minimum number of data points a cluster, `minPts`. While we use a similar notation for these parameters as DBSCAN, their meaning in CLASSIX is slightly different. The groups formed in CLASSIX are not necessarily spherical as the scanned neighborhoods in DBSCAN. Instead, they are formed from a starting point by traversing the data in the sorted order and only including points that have not been aggregated before. The `minPts` criterion is applied on the cluster level in CLASSIX, while in DBSCAN it is used to distinguish between noise, core, and boundary points.

3. The CLASSIX algorithm

In this section we introduce the CLASSIX algorithm, starting with a discussion of the initial preparation of the data (section 3.1), followed by a discussion of the aggregation (section 3.2) and merging (section 3.3) phases. The short sections 3.4 and 3.5 discuss the treatment of outliers and out-of-sample data, respectively.

3.1. Data preparation

Let us assume that we are given n raw data points $p_1, \dots, p_n \in \mathbb{R}^d$ which we would like to cluster. Throughout this work all vectors are column vectors and we assume that $d \ll n$. The dimensionality d is also referred to as the number of features. In the following, we will denote the processed data points as x_1, \dots, x_n .

As a first step, we will center all data points by taking off the empirical mean value of each feature:

$$x_i := p_i - \text{mean}(\{p_j\}).$$

This operation has a complexity of $O(dn)$, i.e. linear in n , and it can be performed *in-place*, meaning that the data points p_i can be overwritten with the mean-centered values x_i if memory consumption is an issue.

The second step consists of computing the first principal component $v_1 \in \mathbb{R}^d$, i.e., the vector along which the data $\{x_i\}$ exhibits largest empirical variance. This vector can be computed by a thin singular value decomposition of the tall-skinny data matrix $X := [x_1, \dots, x_n]^T \in \mathbb{R}^{n \times d}$,

$$X = U\Sigma V^T, \tag{1}$$

where $U \in \mathbb{R}^{n \times d}$ and $V \in \mathbb{R}^{d \times d}$ have orthonormal columns and $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_d) \in \mathbb{R}^{d \times d}$ is a diagonal matrix such that $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_d \geq 0$. The principal components are given as the columns of $V = [v_1, \dots, v_d]$ and we require only the first column v_1 . The score of a point x_i along v_1 is

$$\alpha_i := x_i^T v_1 = (e_i^T X) v_1 = (e_i^T U \Sigma V^T) v_1 = e_i^T u_1 \sigma_1,$$

where e_i denotes the i -th canonical unit vector in \mathbb{R}^n . In other words, the scores α_i of all points can be read off from the first column of $U = [u_1, \dots, u_d]$ times σ_1 . The computation of the scores using a thin SVD requires $O(nd^2)$ operations and is therefore linear in n (Golub and Loan, 2013, chapter 8.6).

The third (and crucial) step is to order all data points x_i by their α_i scores; that is,

$$(x_i) := \text{sort}(\{x_i\})$$

so that $\alpha_1 \leq \alpha_2 \leq \dots \leq \alpha_n$ with each $\alpha_i = x_i^T v_1$. Note that v_1 is not uniquely determined even when $\sigma_1 > \sigma_2$ as also $-v_1$ is a principal component, but it does not matter if the order of the α_i is reversed from largest to smallest. The sorting generally requires $O(n \log n)$ operations on average, and at most $O(n)$ memory, provided that an efficient algorithm such as Quicksort (Hoare, 1962), IntroSort (Musser, 1997) or TimSort (Auger, Jugé, Nicaud and Pivoteau, 2018) is used. It is important to highlight again that the sorting is performed on *scalar* values α_i , irrespective of the data dimensionality d . This will result in a significant advantage in performance over tree-based query structures such as R* or ball tree, which work with the d -dimensional data points and are nontrivial to implement efficiently; see, e.g., the discussion in Kriegel et al. (2017).

Finally, we estimate the *median extend* of the data as the smallest number $\text{mext} > 0$ such that there are $\lceil n/2 \rceil$ data points with a score α_i in the interval $[-\text{mext}, \text{mext}]$. The use of this parameter is merely to make any distance computations independent of the data scale, ensuring that approximately half of the data points x_i have a score α_i that lie within a ball of radius mext about the origin. As the scalars α_i are already sorted, mext can be computed in $O(n)$ operations. An alternative and better estimate is $\text{mext} := \text{median}(\{\|x_i\|\})$, but this is slightly more expensive to compute as it involves n Euclidean norm computations for a cost of $O(nd)$ operations, and another sorting of scalar norm values costing $O(n \log n)$.

We note that throughout this work we will use the Euclidean norm $\|\cdot\|$ for any distance computations. If a weighted Euclidean needs to be used (giving different weight to different features), this weighting can simply be applied to the original data points $\{p_i\}$ before performing the follow-up steps as described above.

3.2. Aggregation phase

The aim of CLASSIX's aggregation phase is to group nearby data points together in a computationally inexpensive manner, where “nearby” is simply defined in terms of the Euclidean distance $\text{dist}(x_i, x_j) = \|x_i - x_j\|$. We consider two data points x_i and x_j close if

$$\text{dist}(x_i, x_j) \leq \text{radius} \cdot \text{mext} =: R.$$

Here, radius is a user-chosen tolerance and mext is the median extend estimated from the data, as explained in the previous section.

The greedy aggregation now proceeds as follows: starting with the first data point x_1 (i.e., the one with the smallest α_i value), we assign the data points x_j for $j = 2, 3, \dots$ to the same group G_1 if $\text{dist}(x_1, x_j) \leq R$. If done in a naive way, this would require computing all $n - 1$ distances between the x_j and x_1 . However, we can utilize the Cauchy–Schwarz inequality

$$|\alpha_i - \alpha_j| = |v_1^T x_i - v_1^T x_j| \leq \|x_i - x_j\| = \text{dist}(x_i, x_j) \quad (2)$$

to avoid doing this: as we have sorted the x_i such that $\alpha_1 \leq \alpha_2 \leq \dots \leq \alpha_n$, we know that

$$\text{if } \alpha_j - \alpha_i > R \text{ for some } j > i \text{ then } \text{dist}(x_i, x_k) > R \text{ for all } k \geq j.$$

Hence, we can terminate adding data points to the first group G_1 as soon as we encounter a data point x_j such that $\alpha_j - \alpha_1 > R$. The point x_1 is also referred to as the *starting point* of group G_1 .

Once the first group G_1 is completed, the assignment process continues with the first data point x_i that is not part of G_1 . This point x_i will be the starting point of the next group G_2 . (We emphasize that the starting points do not need to be searched for; they are always the first data point that was skipped when forming the previous group.) As before, we add unassigned data points x_j , $j > i$, to the group G_2 until we encounter a point such that $\alpha_j - \alpha_i > R$. And this process continues until all points are assigned to a group.

A pseudocode of the greedy aggregation algorithm is given in Algorithm 1. Once completed, all n data points have been partitioned into groups G_1, \dots, G_ℓ with their corresponding starting points $x_{s(1)}, \dots, x_{s(\ell)}$.

3.3. Merging phase

The aim of this phase is to merge the groups G_1, \dots, G_ℓ into clusters C_1, \dots, C_k , $k \leq \ell$. See also Figure 1 for illustration. We discuss two approaches for doing this, one based solely on the distance of the starting points $x_{s(1)}, \dots, x_{s(\ell)}$, and the other based on the density of the points in the groups and a volume of intersection.

Algorithm 1: Greedy aggregation of data points into groups

1. Given data points x_1, x_2, \dots, x_n sorted such that $\alpha_1 \leq \alpha_2 \leq \dots \leq \alpha_n$.
Label all of them as “unassigned”.
2. Let x_i be the first unassigned point x_i (referred to as a “starting point”).
If there are no unassigned points left, terminate.
3. FOR $j = i + 1 : n$
4. Compute $d_{ij} := \text{dist}(x_i, x_j)$
5. IF $d_{ij} \leq R$ and x_j is unassigned, assign x_j to the same group as x_i
6. ELSE IF $\alpha_j - \alpha_i > R$, go to Step 2.

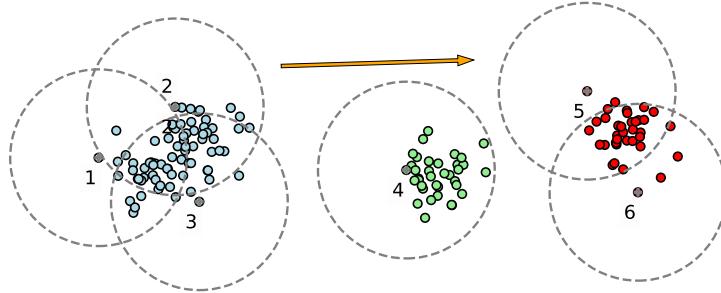


Figure 1: Illustration of a point cloud aggregated into $\ell = 6$ groups (dashed circles) and merged into $k = 3$ clusters (labelled by colour). The data points are ordered along the direction of the first principal component indicated by the arrow, and the six starting points of the groups (which are also data points) follow that same order.

3.3.1. Distance-based merging

We first highlight that the points x_i assigned to each group G_j are still sorted by their first principal coordinates α_i and, as a consequence, so are the starting points; i.e., $\alpha_{s(1)} \leq \dots \leq \alpha_{s(\ell)}$.

In the distance-based merging approach we will merge two groups G_i and G_j if the distance of their starting points satisfies

$$\text{dist}(x_{s(i)}, x_{s(j)}) \leq \text{scale} \cdot R, \quad (3)$$

where scale is a tuning parameter in the interval $[1, 2]$, which can be set to $\text{scale} = 1.5$ in most cases. Note that two distinct starting points $x_{s(i)}$ and $x_{s(j)}$, $i < j$, will never have a distance that is less than R , as otherwise they would have ended up in the same group G_i during aggregation. Also, if two starting points have a distance greater than $2R$, their surrounding R -balls have no overlap. Let us denote the R -ball surrounding a point x as

$$\mathcal{B}(x, R) := \{y \in \mathbb{R}^d : \text{dist}(x, y) \leq R\}.$$

The above requirement that $\text{scale} \in [1, 2]$ ensures that there is a nonempty intersection of the balls $\mathcal{B}(x_{s(i)}, R)$ and $\mathcal{B}(x_{s(j)}, R)$.

To determine which pairs of starting points satisfy the merging criterion (3), we can again use the termination trick based on the corresponding principal coordinates. Given a starting point $x_{s(i)}$, we inspect the starting points that succeed it, namely $x_{s(i+1)}, x_{s(i+2)}, \dots$ in that order. If any of these starting points is a distance not more than $\text{scale} \cdot R$ away from $x_{s(i)}$, it should be in the same cluster as $x_{s(i)}$. As soon as we encounter a starting point $x_{s(i+k)}$ with the property $\alpha_{s(i+k)} - \alpha_{s(i)} > \text{scale} \cdot R$ we can terminate the search.

Once all starting points $x_{s(i)}$ have been traversed, we effectively have a list of pairs of starting points that satisfy the merging criterion (3). These pairs can be thought of as the edges of a directed graph. We then simply determine clusters of starting points by computing the connected components of this graph (or its undirected version) using, e.g., depth-first search (Hopcroft and Tarjan, 1973) or a disjoint-set data structure (Galler and Fisher, 1964). The complexity

of this procedure is $O(\max(\ell, E))$, where E is the number of edges in the graph. All data points x_i which are not starting points are naturally assigned to the same cluster as the starting point of the group they belong to.

3.3.2. Density-based merging

In density-based merging we will join two starting points $x_{s(i)}$ and $x_{s(j)}$ (and the points in their associated groups G_i and G_j , respectively) into one cluster if the density of the points that lie in the intersection $\mathcal{B}(x_{s(i)}, R) \cap \mathcal{B}(x_{s(j)}, R)$ is at least as big as the overall density of the two groups. Using $|\cdot|$ to denote cardinality of a set and $\text{vol}(\cdot)$ to denote the volume of a set, this can be formalized as follows: $x_{s(i)}$ and $x_{s(j)}$ belong to the same cluster if

$$\frac{|\mathcal{B}(x_{s(i)}, R) \cup \mathcal{B}(x_{s(j)}, R)|}{\text{vol}(\mathcal{B}(x_{s(i)}, R) \cup \mathcal{B}(x_{s(j)}, R))} \leq \frac{|\mathcal{B}(x_{s(i)}, R) \cap \mathcal{B}(x_{s(j)}, R)|}{\text{vol}(\mathcal{B}(x_{s(i)}, R) \cap \mathcal{B}(x_{s(j)}, R))}. \quad (4)$$

The volume of the intersection of two R -balls in \mathbb{R}^d is

$$\text{vol}(\mathcal{B}(x, R) \cap \mathcal{B}(y, R)) = \frac{\pi^{d/2} \cdot R^d \cdot I_{1-\text{dist}(x,y)^2/(4R^2)}(d/2 + 1, 1/2)}{\Gamma(d/2 + 1)},$$

with the incomplete beta function defined as

$$I_s(a, b) = \frac{\Gamma(a + b)}{\Gamma(a)\Gamma(b)} \int_0^s t^{a-1} (1-t)^{b-1} dt, \quad s \in [0, 1];$$

see Li (2011). The volume of an R -ball is given as $\text{vol}(\mathcal{B}(x, R)) = \text{vol}(\mathcal{B}(x, R) \cap \mathcal{B}(x, R))$, from which the volume of the union of two R -balls is easily obtained. Two R -balls can only have a nonempty overlap if their centers are not more than $2R$ apart. Hence we can again limit the number of pairs G_i and G_j , $i < j$, for which to check the merging criterion (4) by using the principal coordinate-based criterion: if $\alpha_{s(j)} - \alpha_{s(i)} > 2R$, the groups G_i and G_j do not need to be inspected for intersection.

As in the distance-based case, the set of all pairs of starting points that satisfy (4) can be thought of as the edges of a graph, and we determine clusters of starting points by computing the connected components of this graph. All data points x_i which are not starting points are naturally assigned to the same cluster as the starting point of the group they belong to.

Remark: As is well known, the volume of an R -ball tends to zero as the dimension d increases,

$$\text{vol}(\mathcal{B}(x, R)) \sim \frac{1}{\sqrt{d\pi}} \left(\frac{2\pi e}{d} \right)^{d/2} R^d.$$

Similarly, the volume of any intersection of two balls tends to zero, and even the relative volume of the intersection of two balls (i.e., the ratio of the intersection volume and the volume of a ball) tends to zero unless both balls have the same centers. As a consequence, density-based clustering is only suitable for low feature dimensions.

3.4. Treatment of outliers

After all groups have been merged into k clusters C_1, \dots, C_k , we optionally apply a removal process for outliers. In CLASSIX, outliers are simply defined as the points in clusters that have a small number of elements, namely clusters such that $|C_j| < \text{minPts}$. There are two options for processing outliers.

- **Reassign outliers to larger clusters:** All points in a cluster C_j with less than minPts points will be assigned to larger nearby clusters. This is done on the group level using the starting points. Every group G_i that was merged into C_j has an associated starting point $x_{s(i)}$. We simply find a starting point $x_{s(i')}$ nearest to $x_{s(i)}$ that belongs to a cluster $C_{j'}$ with at least minPts points, and reassign G_i to cluster $C_{j'}$. This procedure will succeed to eliminate all outliers if there is at least one cluster with minPts or more points. Otherwise, all clusters will remain outliers.
- **Return outliers separately:** All points in clusters with less than minPts points will be labelled as outliers and returned as such to the user.

We illustrate these two options applied to an example with synthetic data in Figure 2.

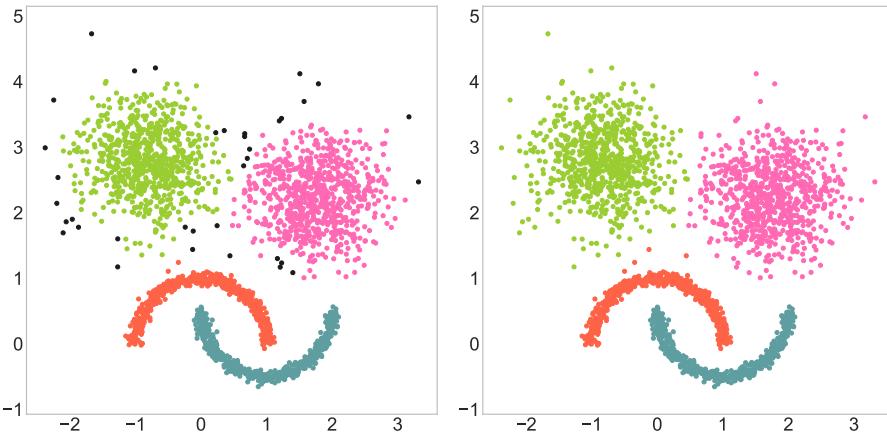


Figure 2: Demonstration of the outlier treatment. We apply CLASSIX to synthetic data with $n = 2500$ points using the parameters $R = 0.1$ and $\text{minPts} = 5$. Left: Without outlier reassignment, 27 clusters are identified, four of which are genuine ground-truth clusters (shown in colour) and each of the other 23 clusters have less than minPts points (shown as the black dots). Right: The reassignment of the 23 outlier clusters results in four clusters of good quality.

3.5. Out-of-sample data

The reassignment strategy for outliers described in section 3.4 can also be used to classify out-of-sample data based on a previously trained model. That is, CLASSIX will allocate a new data point to its closest group (measured by distance to starting points) and then assign it to the associated cluster. This is illustrated in Figure 3. In this example, CLASSIX is first run on a train set (90% of the overall data points) to obtain the clusters, and then tested on the remaining 10% of the data (left vs right plots). In this example, both the train and test data are clustered with 100% accuracy when compared to the provided ground truth (top vs bottom plots).

4. Theoretical considerations

4.1. Efficiency

The efficiency of the aggregation phase crucially depends on the number of pairwise distance calculations that are performed in Step 4 of Algorithm 1. In the best case, each point x_i is involved in exactly one such computation, resulting in $O(n)$ distance computations overall. In the worst case, the distances between each point x_i and all points x_{i+1}, \dots, x_n that succeed it are computed. This amounts to

$$(n-1) + (n-2) + \dots + 2 = \frac{(n-1)n}{2} - 1 = O(n^2)$$

distance computations, an unacceptably high number in most practical situations. The hope is that through the sorting of the data points x_i by their principal coordinates α_i , the early termination criterion in Step 6 of Algorithm 1 is triggered often enough so that, on average, the number of distance computations a data point is involved in remains very small.

An unassigned data point x_j is involved in more than one distance computation only if there is a starting point x_i , $i < j$, such that $\alpha_j - \alpha_i \leq R$ but $\text{dist}(x_i, x_j) > R$. So, it is natural to ask the following question:

How likely is it that $|\alpha_i - \alpha_j| \leq R$, yet $\text{dist}(x_i, x_j) > R$?

We first note that, using the singular value decomposition (1) of the data matrix X , we can derive an upper bound on $\text{dist}(x_i, x_j)$ that complements the lower bound (2). Using that $x_i^T = e_i^T X = e_i^T U \Sigma V^T$ and denoting the elements of

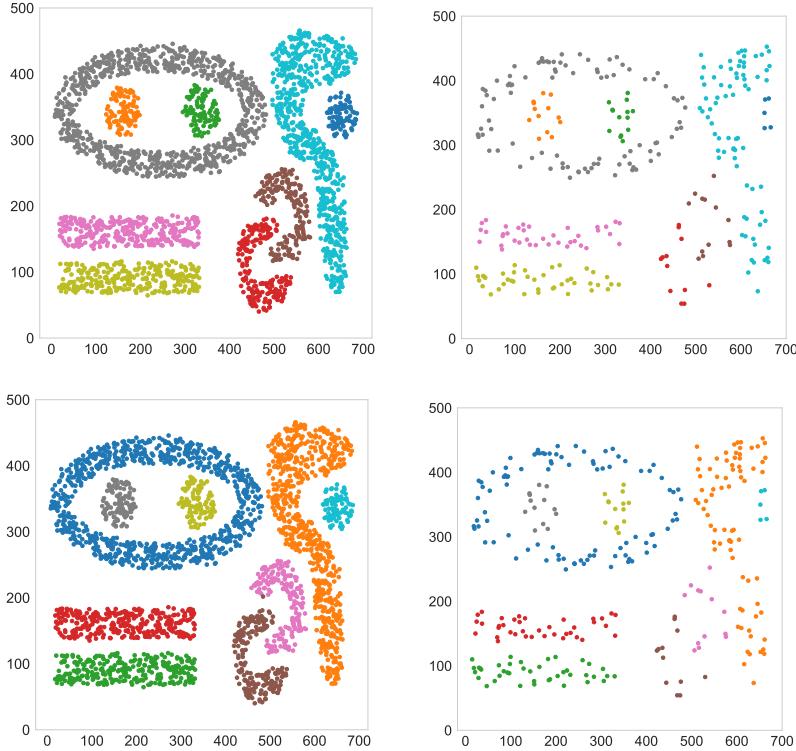


Figure 3: Illustration of CLASSIX clustering with out-of-sample data. Top: Ground truth labels of the train and test data (left and right, respectively). Bottom: CLASSIX clustering of the train and test data (left and right, respectively). The train vs test split is 90% vs 10%.

U by $u_{i,j}$, we have

$$\begin{aligned} \|x_i - x_j\|^2 &= |\alpha_i - \alpha_j|^2 + \left\| \begin{bmatrix} (u_{i2} - u_{j2}), \dots, (u_{id} - u_{jd}) \end{bmatrix} \begin{bmatrix} \sigma_2 & & \\ & \ddots & \\ & & \sigma_d \end{bmatrix} \right\|^2 \\ &\leq |\alpha_i - \alpha_j|^2 + \|u_i - u_j\|^2 \cdot \left\| \begin{bmatrix} \sigma_2 & & \\ & \ddots & \\ & & \sigma_d \end{bmatrix} \right\|^2 \\ &\leq |\alpha_i - \alpha_j|^2 + 2\sigma_2^2. \end{aligned}$$

Therefore,

$$|\alpha_i - \alpha_j|^2 \leq \text{dist}(x_i, x_j) \leq |\alpha_i - \alpha_j|^2 + 2\sigma_2^2 \quad (5)$$

and the gap in these inequalities is determined by σ_2 , the second singular value of X . Indeed, if $\sigma_2 = 0$, then all data points x_i lie on a straight line passing through the origin and their distances correspond exactly to the difference in their principal coordinates. This is a best-case scenario for Algorithm 1, as points x_j will just be added in their sorted order to a group until the distance from the starting point x_i exceeds R , in which case that x_j becomes the next starting point and a new group is formed. As no point needs to be revisited, the number of pairwise distance computations is smallest possible. If, on the other hand, σ_2 is relatively large compared to σ_1 , the gap in the inequalities becomes large as well and the α_i, α_j may not be a good proxy for the point distance $\text{dist}(x_i, x_j)$.

4.2. Parameter dependency

It is clear that there is no natural order of data points in two or higher dimensions, and it is expected that some data points will be involved in more than one distance computation. In order to get a qualitative understanding of how the number of distance computations depends on the various parameters (dimension d , singular values of the data matrix, group radius R , etc.), we consider the following model. Let $\{x_i\}_{i=1}^n$ be a large sample of points whose d components are normally distributed with zero mean and standard deviation $[1, s, \dots, s]$, $s < 1$, respectively. These points describe an elongated ‘‘Gaussian blob’’ in \mathbb{R}^d , with the elongation controlled by s . In the large data limit ($n \rightarrow \infty$) the singular values of the data matrix $X = [x_1, \dots, x_n]^T$ approach $\sqrt{n}, s\sqrt{n}, \dots, s\sqrt{n}$ and the principal components approach the canonical unit vectors e_1, e_2, \dots, e_d . As a consequence, the principal coordinates $\alpha_i = e_1^T x_i$ follow a standard normal distribution, and hence for any $c \in \mathbb{R}$ the probability that $|\alpha_i - c| \leq R$ is given as

$$P_1 = P_1(c, R) = \frac{1}{\sqrt{2\pi}} \int_{c-R}^{c+R} e^{-r^2/2} dr.$$

On the other hand, the probability that $\|x_i - [c, 0, \dots, 0]^T\| \leq R$ is given by

$$P_2 = P_2(c, R, s, d) = \frac{1}{\sqrt{2\pi}} \int_{c-R}^{c+R} e^{-r^2/2} \cdot F\left(\frac{R^2 - (r - c)^2}{s^2}; d - 1\right) dr,$$

where F denotes the χ^2 cumulative distribution function. In this model we can think of the point $[c, 0, \dots, 0]^T$ as a starting point, and our aim is to identify all data points x_i within a radius R of this starting point.

Since $\|x_i - [c, 0, \dots, 0]^T\| \leq R$ implies that $|\alpha_i - c| \leq R$, we have $P_1 \geq P_2$. Hence, the quotient P_2/P_1 can be interpreted as a conditional probability of a point x_i satisfying $\|x_i - [c, 0, \dots, 0]^T\| \leq R$ given that $|e_1^T x_i - c| \leq R$, i.e.

$$P = P\left(\|x_i - [c, 0, \dots, 0]^T\| \leq R \mid |e_1^T x_i - c| \leq R\right) = P_2/P_1.$$

Ideally, we would like this quotient P be close to 1. As all the points x_i are sorted by their first principal coordinates α_i , all points within a radius of R from $[c, 0, \dots, 0]^T$ (belonging to the same group) are very likely to have consecutive indices if $P \approx 1$. On the other hand, if P is very small, then the sorting has very little effect of indexing nearby points close to each other.

It is now easy to study the dependence of the quotient $P = P_2/P_1$ on the various parameters. First note that P_1 does not depend on s nor d , and hence the only effect these two parameters have on P is via the factor $F\left(\frac{R^2 - (r - c)^2}{s^2}; d - 1\right)$ in the integrand of P_2 . This term corresponds to the probability that the sum of squares of $d - 1$ independent Gaussian random variables with mean zero and standard deviation s is less or equal to $R^2 - (r - c)^2$. Hence, P_2 and therefore P are monotonically decreasing as s or d are increasing. This is consistent with intuition: as s increases, the elongated point cloud $\{x_i\}$ becomes more spherical and hence it gets more difficult to find a direction in which to enumerate the points naturally. And this problem gets more pronounced in higher dimensions d .

We now show that P converges to 1 as R increases. First note that for an arbitrarily small $\epsilon > 0$ there exists a radius $R_1 > 1$ such that $P_1(c, R_1 - 1) > 1 - \epsilon$. Further, there is a $R_2 > 1$ such that

$$F\left(\frac{R_2^2 - (r - c)^2}{s^2}; d - 1\right) > 1 - \epsilon \quad \text{for all } r \in [c - R_2 + 1, c + R_2 - 1].$$

To see this, note that the cumulative distribution function F monotonically increases from 0 to 1 as its first argument increases from 0 to ∞ . Hence there exists a value T for which $F(t, d - 1) > 1 - \epsilon$ for all $t \geq T$. Now we just need to find R_2 such that

$$\frac{R_2^2 - (r - c)^2}{s^2} \geq T \quad \text{for all } r \in [c - R_2 + 1, c + R_2 - 1].$$

The left-hand side is a quadratic function with roots at $r = c - R_2$ and $r = c + R_2$ and symmetric with respect to the maximum at $r = c$. Hence choosing R_2 such that

$$\frac{R_2^2 - ([c + R_2 - 1] - c)^2}{s^2} = T, \quad \text{i.e.,} \quad R_2 = \left(\frac{Ts^2 + 1}{2}\right)^{1/2},$$

or any value R_2 larger than that, will be sufficient. Now, setting $R = \max\{R_1, R_2\}$, we have

$$P_2 \geq \frac{1}{\sqrt{2\pi}} \int_{c-R+1}^{c+R-1} e^{-r^2/2} \cdot F\left(\frac{R^2 - (r-c)^2}{s^2}; d-1\right) dr \geq (1-\epsilon)^2.$$

Hence, both P_1 and P_2 come arbitrarily close to 1 as R increases, and so does their quotient P_2/P_1 .

We illustrate our findings in Figure 4. The model just analyzed only gives quantitative insights as it does not take into account that our aggregation algorithm (i) uses starting points that are not necessarily at the center of a group and (ii) removes points once they have been assigned to a group. Nevertheless, we can read off from Figure 4 that aggregation generally gets harder in higher dimensions and when R gets smaller.

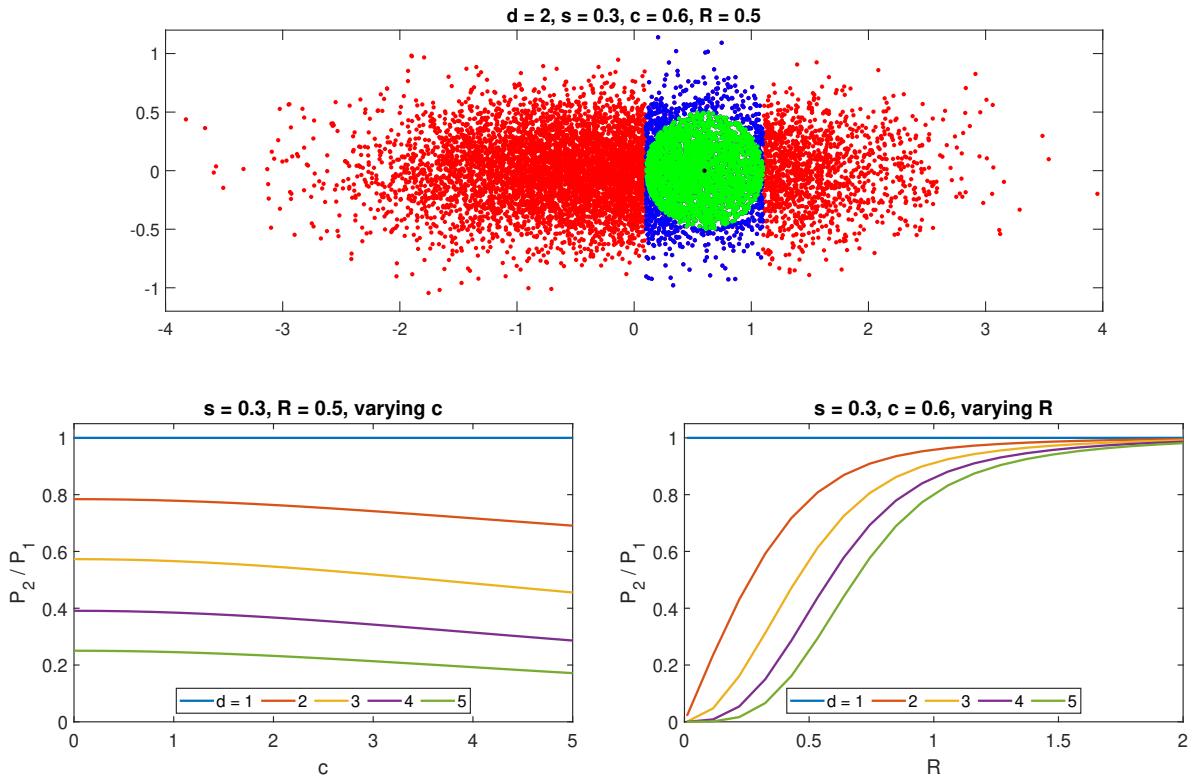


Figure 4: Illustration of the model analyzed in section 4.2. Top: A Gaussian blob of $n = 10^4$ points (red dots) in $d = 2$ dimensions, with the horizontal components having a standard deviation of 1 and the vertical components having a standard deviation of $s = 0.3$. The radius for the aggregation is chosen as $R = 0.5$. The blue dots are all points whose first component (first principal coordinate) is within a distance of R from $c = 0.6$, while the blue points are within a distance of R from the group center $c = [0.6, 0]^T$ (shown as the black dot). Bottom left: The quotient P_2/P_1 corresponding to the ratio of green and blue points as a function of c and d . A ratio closer to 1 means that sorting-based aggregation is more efficient, with fewer points being revisited. Bottom right: P_2/P_1 as a function of R and d .

5. Experiments

In this section we conduct extensive experiments on various datasets to evaluate the performance of CLASSIX and compare it to other clustering algorithms. The quality of a clustering will be evaluated using metrics like the adjusted Rand index (ARI) and the adjusted mutual information (AMI); see (Hubert and Arabie, 1985) and (Vinh, Epps and Bailey, 2010), respectively. For a fairest possible comparison of timings we select widely-used clustering

algorithms for which reliable and tuned implementations are publicly available. In particular, we use Cython compiled implementations of k-means++, DBSCAN, HDBSCAN, and Quickshift++. For the HDBSCAN and Quickshift++ implementations we refer to (McInnes and Healy, 2017) and (Jiang et al., 2018), respectively, while the other methods are implemented in the scikit-learn library (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, Vanderplas, Passos, Cournapeau, Brucher, Perrot and Duchesnay, 2011). All reported runtimes are averaged timings of ten consecutive runs. For the reproducibility of our results, all datasets are chosen from publicly available sources or generated with open-source software. All computations are done on a Dell PowerEdge R740 Server with two Intel Xeon Silver 4114 2.2G processors, 1,536 GB RAM, and 1.8 TB disk space. All algorithms are forced to run in one thread.

5.1. Gaussian blobs

In this first test we study the sensitivity of several clustering algorithms to the data size n and the feature dimensionality d . To this end we generate n data points in d dimensions forming ten isotropic Gaussian blobs of unit standard deviation using the `sklearn.datasets.make_blobs` function in scikit-learn.

5.1.1. Dependence on data size

We compare k-means++, DBSCAN (using the index query structures “ball-tree” and “K-D tree”), HDBSCAN, and CLASSIX. The data are n points forming ten isotropic Gaussian blobs of unit standard deviation with n varying between 5,000 and 50,000 while the feature dimension $d = 10$ is fixed. The method’s hyperparameters are chosen as follows. For CLASSIX, we set `radius = 0.3` and `minPts = 5`. For DBSCAN, we use $\epsilon = 3$ and `minPts = 1`. Quickshift++ is run with $k = 20$ and $\beta = 0.7$. Finally, k-means++ is run with the ground-truth value of $k = 10$ clusters. The results are presented in the top two plots of Figure 6, with the ARI shown on the left and the runtime shown on the right. All the curves are plotted with a 95% confidence band estimated from the ten consecutive runs of each method.

We find that all compared methods obtain almost perfect clustering results in this test as judged from an ARI score close to one; see Figure 5 (top left). In terms of timings, both k-means++ and CLASSIX achieve very good performance, with the overall computation time increasing approximately linearly as n increases; see Figure 5 (top right). The other methods exhibit a nonlinear, possibly quadratic, increase in runtime.

The slow, seemingly linear, increase in runtime of CLASSIX is owed to the small number of distance computations between data points due to the early termination condition discussed in section 3.2. To confirm this, we show in Figure 5 (bottom) the *average number of distance calculations per data point* (`avg_dist_pp`) as n increases, both with and without the early termination condition. We find that early termination leads to a significant reduction in the number of distance calculations. In fact, for this data `avg_dist_pp` appears to stay bounded with early termination as n increases.

5.1.2. Dependence on data dimension

We now use a fixed number of $n = 10,000$ samples of ten Gaussian blobs and vary the feature dimension from $d = 10$ to 100. The methods’ hyperparameters are the same as in the previous section, except that for DBSCAN we now use $\epsilon = 10$. This change of ϵ is in DBSCAN’s favour, to delay a very sharp decline in ARI for increasing dimensions observed with $\epsilon = 3$.

The results are shown Figure 6. In the plot on the top left we observe an ARI degradation of DBSCAN and Quickshift++ as d increases. The hyperparameters of these two methods appear to be rather sensitive to the feature dimension. The other methods, including HDBSCAN and CLASSIX, perform robustly across all dimensions without requiring hyperparameter adjustment. In terms of timings, we again find that k-means++ and CLASSIX exhibit the best performance. For this data, the runtimes of all methods except HDBSCAN appear to scale linearly in the feature dimension d .

In the bottom of Figure 6 we again show the average number of distance calculation per data point (`avg_dist_pp`) as the feature dimension d increases. We find that, both with and without early termination, `avg_dist_pp` decreases as d increases. This is to be expected as a Gaussian blob with unit standard deviation becomes more “focused” in higher dimensions. Clustering a more focused point cloud with a fixed aggregation radius R is equivalent to clustering a constant point cloud with increasing radius R . In line with the analysis of the Gaussian model in section 4.2, we indeed expect the number of distance calculations to decrease in higher dimensions.

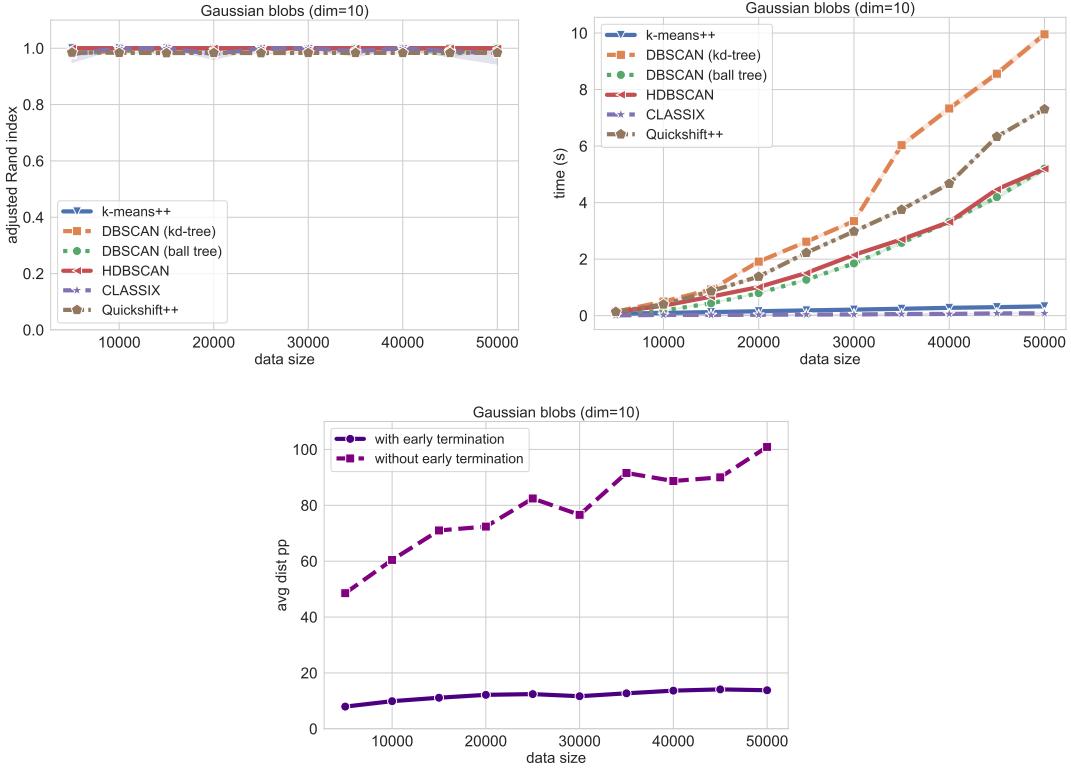


Figure 5: Performance evaluation of various clustering methods on the Gaussian blobs data. Top: We vary the number of data points from $n = 5,000$ to $50,000$ and measure the adjusted Rand index (left) and timing (right) for each method. The feature dimension is fixed at $d = 10$ in all cases. Bottom: Average number of distance calculations per data point in the CLASSIX aggregation phase with and without early termination.

5.1.3. Sensitivity to parameters

We now focus our attention on the sensitivity of CLASSIX with respect to its `radius` parameter. Figure 7 shows the ARI and AMI scores of clustering with `radius` varying between 0.1 to 1.0. The data forms ten Gaussian blobs in $d = 10$ dimensions and the number of data points n is varied from 1000 to 9000 in steps of 1000 (for each plot from the left to right and top to bottom). We show the ARI and AMI scores for both distance-based and density-based CLASSIX clustering. We observe that the method is not very sensitive to the `radius` parameter. As long as this parameter is chosen within the interval 0.2 to 0.6, good to very good clustering results are obtained.

5.2. UCI Machine Learning Repository

We now compare CLASSIX with DBSCAN, HDBSCAN, and Quickshift++ on a number of commonly-used real-world datasets listed in Table 1. All of these datasets except “Phoneme” and “Wine” can be found in the UCI Machine Learning Repository (Dua and Graff, 2017).

We preprocess the data by removing rows with missing values and z-normalise the features (i.e., shift each feature to zero mean and scale it to unit variance). We then perform a hyperparameter search by measuring the ARI and AMI scores while varying the main parameters of each method as shown in Figure 8 and its caption. For each method and dataset we use the respective best hyperparameters and report the achieved ARI and AMI scores in Table 2. The average scores across all datasets shown on the bottom of Table 2 indicate that CLASSIX achieves a good performance compared to the other methods, in particular when density-based merging is used. The only exception is the “Phoneme” dataset where CLASSIX with distance-based merging significantly outperforms density-based merging. This might be explained by the rather high feature dimension of this dataset; see also the remark at the end of section 3.3.2.

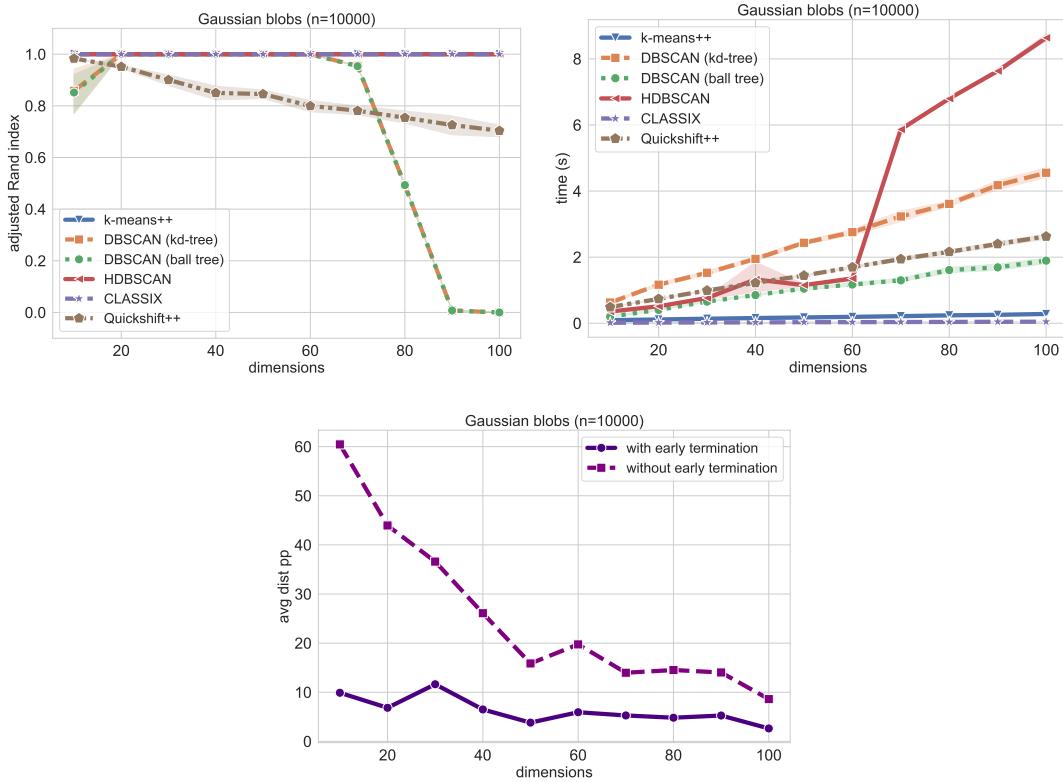


Figure 6: Performance evaluation of various clustering methods on the Gaussian blobs data. Here we keep the number of data points fixed at $n = 10,000$ and vary the dimension from $d = 10$ to 100 . Top: The plots show the adjusted Rand index (left) and timing (right) for each method. Bottom: Average number of distance calculations per data point in the CLASSIX aggregation phase with and without early termination conditions.

5.3. Scikit-learn benchmark

The scikit-learn library contains a benchmark² with six datasets and a number of clustering methods such as BIRCH (Zhang, Ramakrishnan and Livny, 1996), spectral clustering (Shi and Malik, 2000; Yu and Shi, 2003), and Gaussian mixture models (Murphy, 2012, Chap. 11). The library provides preselected parameters for all of these methods and we have left them unchanged. We added HDBSCAN, Quickshift++, and CLASSIX with distance-based and density-based merging to the benchmark and tuned these methods by grid search to find the best hyperparameters in terms of achieved ARI and AMI score.

The clusterings computed by each method are visualized in Figure 9. On all five datasets, CLASSIX produces very good clusters. Among the other methods, Quickshift++ stands out with comparable performance. The figure also shows the required computation time in the bottom right of each plot, and for CLASSIX the average number of distance computations per data point in the bottom left. Most methods can be considered fast, with just a few milliseconds of computation time required, but CLASSIX consistently performs fastest here. This is explained by the small number of distance computations that are required, on average never exceeding 5.47 comparisons per data point in these five tests. The clustering performance measured in terms of ARI and AMI score is shown in Table 3. Both Quickshift++ and CLASSIX stand out as the best performing methods here.

5.4. Shape clusters test

This benchmark involves eight synthetic datasets with varying shapes that are challenging for clustering algorithms. The properties of each dataset are detailed in Table 4. Before clustering this data, we z-normalise each of the features.

²<https://scikit-learn.org/stable/modules/clustering.html>

Fast and explainable clustering based on sorting

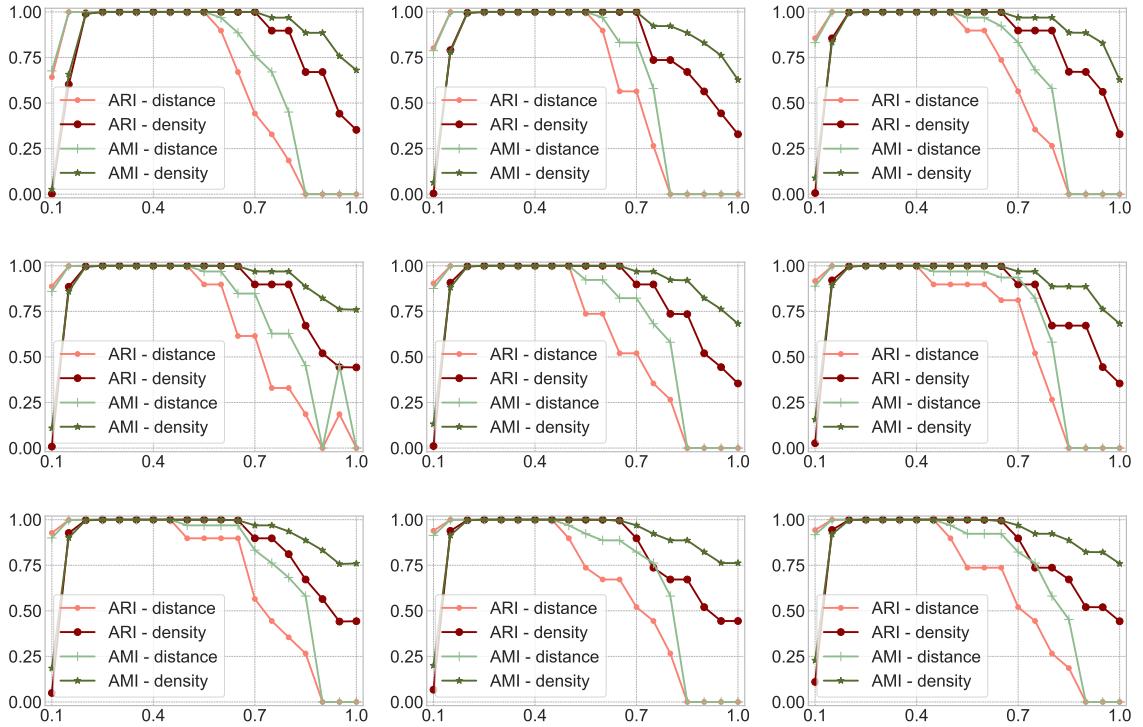


Figure 7: Sensitivity of CLASSIX on the Gaussian blobs data as the radius parameter is varied from 0.1 to 1. We report the ARI and AMI values for distance-based and density-based merging. CLASSIX produces good clusterings over a wide parameter range from 0.2 to 0.6 (approximately).

Table 1

Datasets in the UCI Machine Learning Repository

Dataset	Size	Dimension	Labels	Related references
Banknote	1372	4	2	Dua and Graff (2017)
Dermatology	366	34	6	Dua and Graff (2017); Güvenir, Demiröz and İlter (1998)
Ecoli	336	7	8	Dua and Graff (2017); Nakai and Kanehisa (1991, 1992)
Glass	214	9	6	Dua and Graff (2017)
Iris	150	5	3	Dua and Graff (2017); Fisher (1936); Anderson (1936)
Phoneme	4509	256	5	Hastie, Tibshirani and Friedman (2009)
Seeds	210	7	3	Dua and Graff (2017); Charytanowicz, Niewczas, Kulczycki, Kowalski, Łukasik and Żak (2010)
Wine	178	13	3	Forina, Leardi, C and Lanteri (1998)

We compare k-means++, Mean shift, DBSCAN, HDBSCAN, Quickshift++, CLASSIX (distance-based), and CLASSIX (density-based). For k-means we specify the ground-truth number k , while the hyperparameters of the other methods are again optimized by grid search. The clustering results are visualized in Figure 10, together with the required computation time (bottom right of each plot) and for CLASSIX, the average number of distance computations

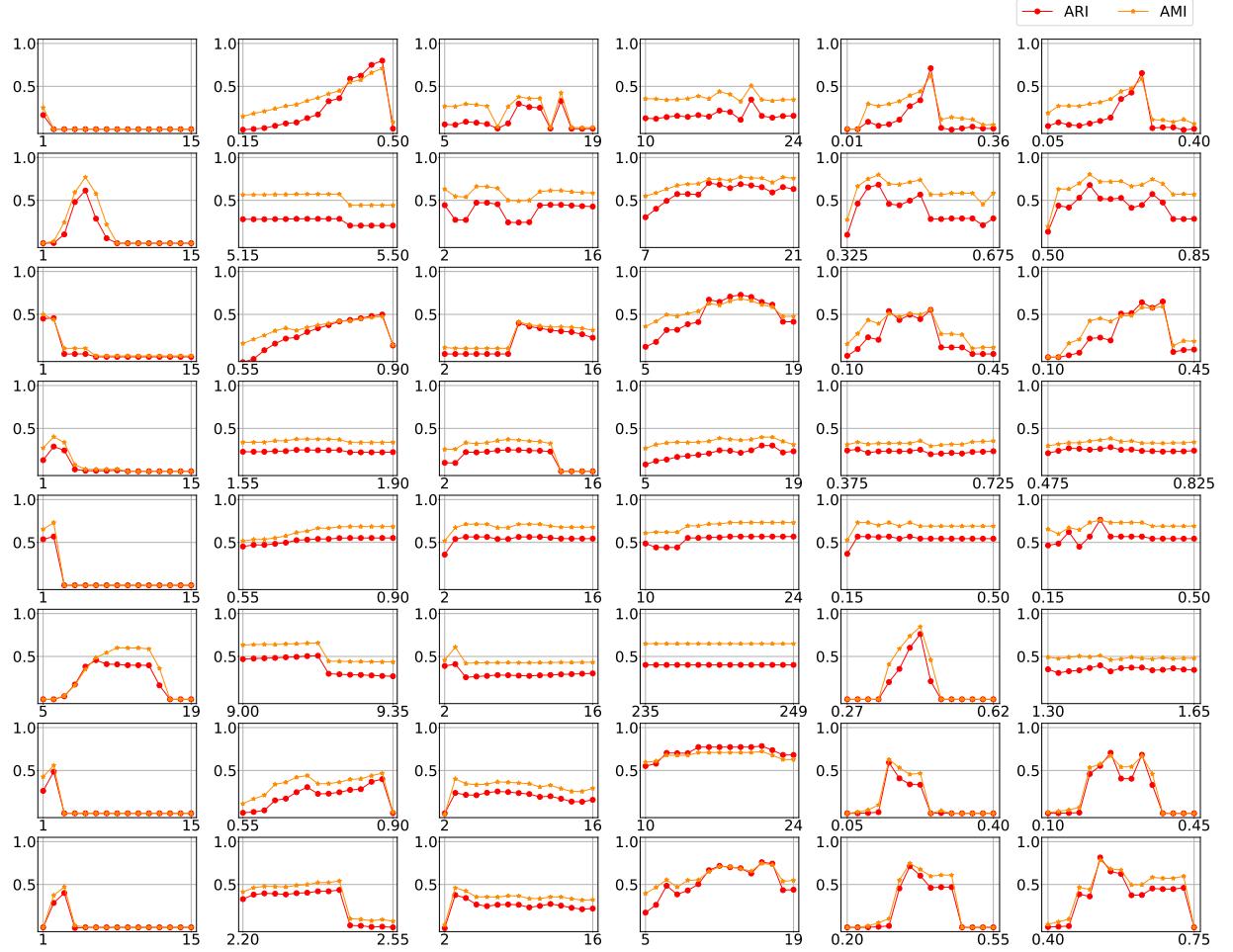


Figure 8: Clustering quality on the UCI Machine Learning Repository as the hyperparameters change. The rows correspond eight datasets in Table 1 while the columns correspond to Mean shift, DBSCAN, HDBSCAN, Quickshift++, CLASSIX with distance-based merging, and CLASSIX with density-based merging, respectively. The searched parameters are bandwidth (Mean shift), ϵ (DBSCAN), minimum cluster size (HDBSCAN), k (Quickshift++), and radius (CLASSIX).

per data point shown in the bottom left of the CLASSIX plots. We find that CLASSIX with density-based merging produces slightly better clusters than distance-based merging, but at the expense of a higher computation time.

The measured AMI and ARI scores are listed in Table 5. Here we see a more mixed picture, with algorithms performing rather differently depending on the dataset. Clearly, in view of Figure 10, ARI and AMI scores do not always provide a good measure of what would be considered a “good clustering” in human perception. Still, on average, CLASSIX with density-based merging appears to perform best across all datasets, closely followed by CLASSIX (distance-based), Quickshift++, DBSCAN, and HDBSCAN (in that order).

5.5. Facial image classification

Most often, image classification is done by supervised learning techniques such as deep convolutional neural networks (Krizhevsky, Sutskever and Hinton, 2012; Simonyan, Vedaldi and Zisserman, 2014) instead of unsupervised learning techniques. Nevertheless, there can be benefit in using unsupervised clustering algorithms in situations where labels are not provided.

We apply CLASSIX to classify images in the Olivetti face dataset. The dataset contains 400 images of 40 distinct faces with variations in lighting, facial expressions (e.g., smiling or not), and other facial details (e.g., wearing glasses or not). Following the example in (Rodriguez and Laio, 2014), we select ten variants of ten faces from the dataset. The images are resized to 100×100 pixels and represented as $d = 100^2$ -dimensional feature vectors containing the

Table 2

Performance on datasets in the UCI Machine Learning Repository. The blue-shaded rows are ARI scores and the white rows are AMI scores. For each row, the best (highest) scores are printed in bold.

Dataset	Mean shift	DBSCAN	HDBSCAN	Quickshift++	CLASSIX (distance)	CLASSIX (density)
Banknote	0.16	0.80	0.33	0.34	0.87	0.85
	0.25	0.71	0.42	0.51	0.78	0.76
Dermatology	0.61	0.28	0.47	0.65	0.68	0.68
	0.77	0.57	0.66	0.77	0.80	0.80
Ecoli	0.46	0.50	0.40	0.73	0.56	0.67
	0.44	0.48	0.41	0.68	0.58	0.62
Glass	0.29	0.25	0.25	0.30	0.23	0.28
	0.40	0.38	0.37	0.40	0.35	0.38
Iris	0.57	0.55	0.56	0.57	0.56	0.83
	0.73	0.68	0.71	0.73	0.68	0.81
Phoneme	0.41	0.51	0.41	0.40	0.76	0.50
	0.60	0.66	0.61	0.65	0.85	0.57
Seeds	0.49	0.40	0.24	0.78	0.70	0.71
	0.56	0.47	0.40	0.72	0.68	0.67
Wine	0.40	0.44	0.38	0.76	0.47	0.80
	0.47	0.54	0.46	0.75	0.61	0.76
Average	0.42	0.47	0.38	0.57	0.60	0.66
	0.53	0.56	0.51	0.65	0.67	0.67

Table 3

ARI (blue-shaded rows) and AMI (white rows) scores on the six datasets in the scikit-learn clustering benchmark. The last two rows “AVG” correspond to the averages across all datasets. For each row, the best (highest) scores are printed in bold.

Data	k-means++	Aggl. clust.	Gaussian mixture	Mean shift	DBSCAN	BIRCH	OPTICS	Spectral clust.	Affinity prop.	HDBSCAN	Quickshift++	CLASSIX (distance)	CLASSIX (density)
I	0.00	1.00	0.00	0.01	1.00	0.01	1.00	1.00	0.00	1.00	1.00	1.00	1.00
	0.00	0.99	0.00	0.01	1.00	0.00	1.00	1.00	0.00	1.00	1.00	1.00	1.00
II	0.49	1.00	0.50	0.55	1.00	0.65	1.00	1.00	0.41	1.00	1.00	1.00	1.00
	0.39	1.00	0.40	0.44	1.00	0.60	1.00	1.00	0.36	1.00	1.00	1.00	0.99
III	0.81	0.93	0.97	0.84	0.55	0.55	0.71	0.94	0.81	0.86	0.94	0.95	0.92
	0.80	0.90	0.94	0.83	0.66	0.66	0.73	0.91	0.80	0.84	0.92	0.92	0.89
IV	0.61	0.42	1.00	0.53	0.97	0.57	0.95	0.96	0.62	0.89	1.00	1.00	1.00
	0.62	0.53	1.00	0.63	0.95	0.63	0.92	0.94	0.62	0.86	1.00	1.00	1.00
V	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
VI	0.00	0.00	0.00	1.00	1.00	0.00	1.00	0.00	0.00	0.00	1.00	1.00	1.00
	0.00	0.00	0.00	1.00	1.00	0.00	1.00	0.00	0.00	0.00	1.00	1.00	1.00
AVG	0.48	0.72	0.58	0.65	0.92	0.46	0.94	0.82	0.47	0.79	0.99	0.99	0.99
	0.47	0.74	0.56	0.65	0.94	0.48	0.94	0.81	0.46	0.78	0.99	0.99	0.98

gray-scale values. We run distance-based CLASSIX with `radius = 0.6` and `minPts = 7` to cluster these faces into groups of similar images, hopefully corresponding to the same person. The results are shown in Figure 11 with different colours corresponding to different clusters. Images shown in gray-scale are outliers, i.e., not assigned to any cluster.

We see from Figure 11 that CLASSIX classifies 77 of the 100 images into 7 clusters. Six of these clusters contain ten images corresponding to the correct classes, i.e., these 60 images (60% of all) are perfectly classified. The seventh cluster mixes 8 images of one face with 9 images of another similarly looking face. We may distinguish two types of

Fast and explainable clustering based on sorting

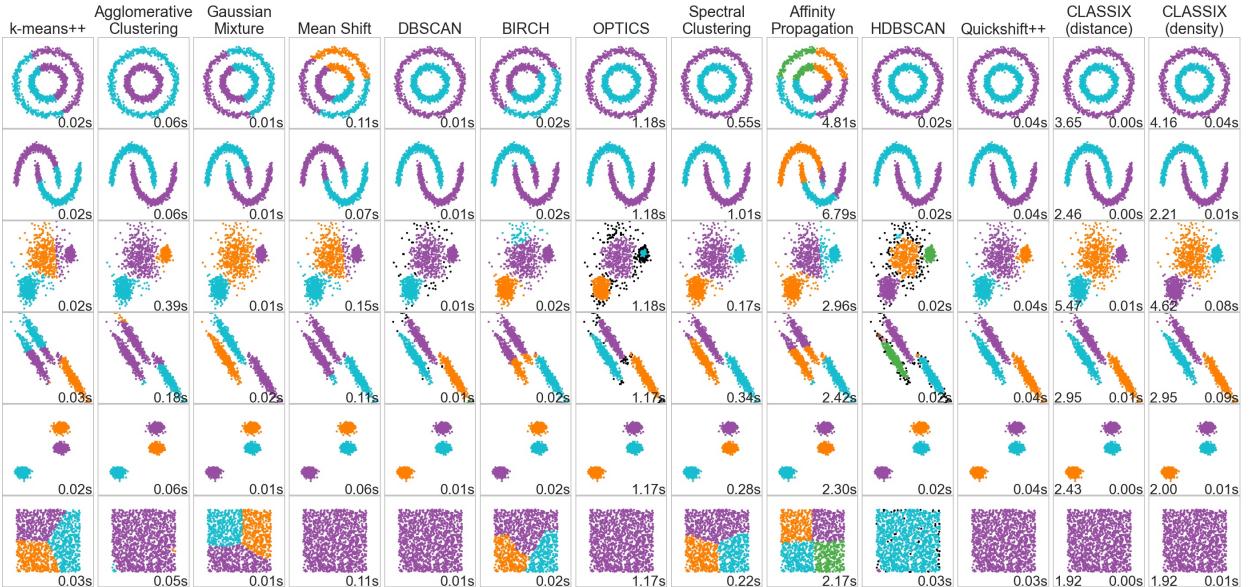


Figure 9: Visualization of clustering results on the scikit-learn benchmark. Each plot shows the required computation time on the bottom right, and for CLASSIX the average number of distance computations per data point on the bottom left.

Table 4

Datasets in the Shape benchmark used for our tests

Id	Dataset	Size	Labels	Related references
(i)	Aggregation	788	7	(Gionis, Mannila and Tsaparas, 2007)
(ii)	Compound	399	6	(Zahn, 1971)
(iii)	D31	3100	31	(Veenman, Reinders and Backer, 2002)
(iv)	Flame	240	2	(Fu and Medico, 2007)
(v)	Jain	373	2	(Jain and Law, 2005)
(vi)	Pathbased	300	3	(Chang and Yeung, 2008)
(vii)	R15	600	15	(Veenman et al., 2002)
(viii)	Spiral	312	3	(Chang and Yeung, 2008)

misclassification as follows: type (A) where images of a unique face are assigned to different clusters, and type (B) where images showing different faces are assigned to the same cluster. Using these types, CLASSIX produces no type (A) misclassifications on this data, but two faces have been type (B) misclassified.

For comparison, the method by Rodriguez and Laio (2014) classifies 41 out of the 100 images into 9 clusters. Of these, 33 images (33% of all) are perfectly classified into 7 clusters. The remaining 8 images correspond to the same face and have been type (B) misclassified into two clusters. There are no type (A) misclassifications.

5.6. Image segmentation

Image segmentation is a technique to partition an image into regions sharing similar characteristics, usually for the purpose of identifying objects. Here we demonstrate an application of CLASSIX to cluster images on a pixel-by-pixel level. To this end we select five images from the COCO 2017 dataset (Lin, Maire, Belongie, Hays, Perona, Ramanan, Dollár and Zitnick, 2014), as shown in Figure 12. Each RGB image of size 100-by-100 pixels is converted into a data array of shape $[100^2, 3]$, and we then cluster these $n = 100^2$ data points of dimension $d = 3$ using DBSCAN,

Fast and explainable clustering based on sorting



Figure 10: Visualisation of clustering results on the Shape benchmark. Each plot shows the required computation time on the bottom right, and for CLASSIX the average number of distance computations per data point on the bottom left.

HDBSCAN, Quickshift++, and CLASSIX with distance-based merging. After the clusters are formed, each pixel feature is replaced by the mean of all points in the cluster it is associated with, resulting in the segmented image.

It is very challenging to choose the parameters of all these methods so that the number of resulting clusters is fairly comparable. Here we have used `radius` = 0.15 and `scale` = 1.05 for CLASSIX and then manually tuned `minPts` and the hyperparameters of the other clustering algorithms to produce a comparable, preferably slightly higher number of clusters. As can be seen in Figure 12, CLASSIX is by far the fastest method in this test, producing reconstructions of high visual quality compared to the other methods despite using the smallest number of clusters. The most challenging image (Zebra) required about 0.33s to segment with an average number of distance calculations per data point of 28.84.

Table 5

Performance of various clustering algorithms on the Shape benchmark. The blue-shaded rows correspond to ARI scores while the white rows are AMI scores. For each row, the best (highest) scores are printed in bold.

Data	k-means++	Mean shift	DBSCAN	HDBSCAN	Quickshift++	CLASSIX (distance)	CLASSIX (density)
Aggregation	0.72	0.86	0.90	0.84	0.97	0.92	0.96
	0.83	0.89	0.94	0.90	0.97	0.96	0.97
Compound	0.57	0.77	0.92	0.81	0.79	0.82	0.85
	0.72	0.79	0.89	0.85	0.79	0.83	0.89
D31	0.96	0.94	0.31	0.60	0.93	0.90	0.83
	0.97	0.95	0.74	0.84	0.93	0.94	0.93
Flame	0.42	0.36	0.84	0.73	1.00	0.87	0.97
	0.38	0.50	0.75	0.66	1.00	0.81	0.94
Jain	0.55	0.27	0.94	0.92	1.00	1.00	1.00
	0.51	0.45	0.84	0.82	1.00	1.00	1.00
Pathbased	0.48	0.52	0.75	0.67	0.47	0.61	0.68
	0.56	0.52	0.74	0.66	0.47	0.70	0.73
R15	0.99	0.99	0.90	0.95	0.98	0.98	0.91
	0.99	0.99	0.92	0.96	0.98	0.99	0.97
Spiral	-0.01	0.12	1.00	1.00	0.88	0.97	1.00
	-0.01	0.43	1.00	1.00	0.88	0.96	1.00
Average	0.59	0.60	0.82	0.81	0.88	0.88	0.90
	0.62	0.69	0.85	0.84	0.88	0.90	0.93

6. Summary and future work

We have introduced a fast clustering algorithm based on the sorting of the data points by their first principal coordinate. CLASSIX’s key component is the fast aggregation of nearby data points into groups, exploiting the sorted order to reduce the number of pairwise distance computations. The simplicity of the aggregation and merging phases allows for clustering results to be explainable, a property that we demonstrate in the Appendix.

We have tested the clustering efficiency in a number of experiments, ranging from low to high-dimensional data. Apart from the analysis of the simple Gaussian model in section 4.2, we currently do not have a good theoretical understanding of why sorting helps even for high-dimensional “real-world” datasets such as those contained in the UCI Machine Learning Repository. While it is known that “big data” matrices often have low numerical rank (Udell and Townsend, 2019), we usually find that the second singular value of the data matrix is still significant in comparison to the first. Hence, bounds like (5) cannot explain why data points can be aggregated with such a small number of pairwise distance computations as we have seen in the experiments. Further work will be needed to understand this inherent “sortability” of big data.

Acknowledgement. We are grateful to Kamil Oster who provided us with the dataset used in the Appendix.

A. Appendix: A demonstration of the explainability feature

As CLASSIX is a direct, noniterative, clustering method it is easy to explain its outputs. In its aggregation phase, data points will be grouped together when their distance from a starting point does not exceed radius . The subsequent merging phase is also fully deterministic and can be explained by distances between starting points (in distance-based merging) or by comparing densities of the groups (in density-based merging). The whole clustering procedure can hence be tracked and summarised in a finite number of steps.

Below we provide a demo of CLASSIX and its explainability feature. In order to run the included Python code, CLASSIX needs to be installed via:



Figure 11: CLASSIX clustering performance on the Olivetti face dataset. Different colours indicate different clusters. The gray-scale images are outliers that have not been assigned to any cluster. The computation time is about 0.33s.

```
pip install ClassixClustering
```

We consider a large dataset relating to sensor readings in a vacuum distillation unit (Oster, Gütterl, Shapiro, Chen and Jobson, 2021), which has been included in the CLASSIX module for convenience. It contains $n = 2,028,780$ data points and is difficult or even impossible to cluster for many density-based algorithms. Here we apply CLASSIX with `radius = 1`, `minPts = 0`, and distance-based group merging (the latter two are default values and could be omitted from the call). CLASSIX completes the task in about 1.2 seconds, returning 4 well separated clusters:

```
import time
import numpy as np
from classix import CLASSIX, load_data
data = load_data('vdu_signals')
st = time.time()
clx = CLASSIX(radius=1, minPts=0, group_merging='distance')
clx.fit_transform(data)
print('Clustering time (s):', time.time() - st)
```

Fast and explainable clustering based on sorting



Figure 12: Image segmentation via clustering. The left-most column shows the original image. For each method and image, the required clustering time is shown on the bottom right. For CLASSIX, the average number of distance calculations per data point is shown on the bottom left.

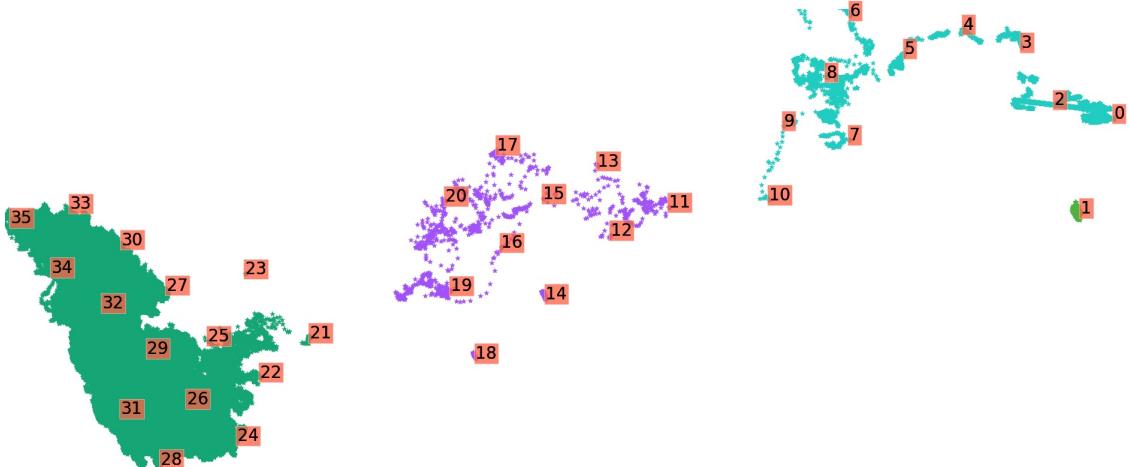
Output:

```
CLASSIX(sorting='pca', radius=1, minPts=0, group_merging='distance')
The 2028780 data points were aggregated into 36 groups.
In total 3920623 comparisons were required (1.93 comparisons per data point).
The 36 groups were merged into 4 clusters with the following sizes:
  * cluster 0 : 2008943
  * cluster 1 : 16920
  * cluster 2 : 1800
  * cluster 3 : 1117
Try the .explain() method to explain the clustering.
Clustering time (s): 1.2285659313201904
```

We can get a more detailed overview of the computed groups and clusters by calling the `explain` method:

```
clx.explain(plot=True)
```

Output:



A clustering of 2028780 data points with 2 features has been performed.

The radius parameter was set to 1.00 and MinPts was set to 0.

As the provided data has been scaled by a factor of 1/2.46,
data points within a radius of $R=1.00 \times 2.46 = 2.46$ were aggregated into groups.
In total 3920623 comparisons were required (1.93 comparisons per data point).
This resulted in 36 groups, each uniquely associated with a starting point.
These 36 groups were subsequently merged into 4 clusters.

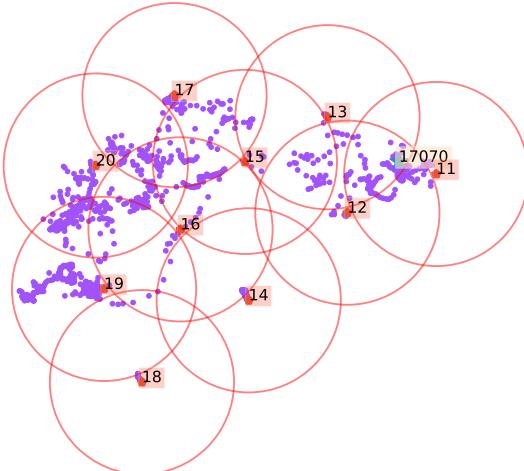
A list of all starting points is shown below.

Group	NrPnts	Cluster	Coordinates
0	10560	1	16.35 3.26
1	1800	2	15.81 1.85
2	2580	1	15.38 3.47
---	lines omitted	---	---
33	123548	0	-0.89 1.92
34	274	0	-1.2 0.96
35	65	0	-1.87 1.7

In order to explain the clustering of individual data points,
use `.explain(ind1)` or `.explain(ind1, ind2)` with indices of the data points.

The above plot of the data is produced using the first two principal coordinates. The indices shown in the red rectangles refer to the starting points. If we want to understand the cluster assignment of a specific data point, we can do so by providing the index of that point:

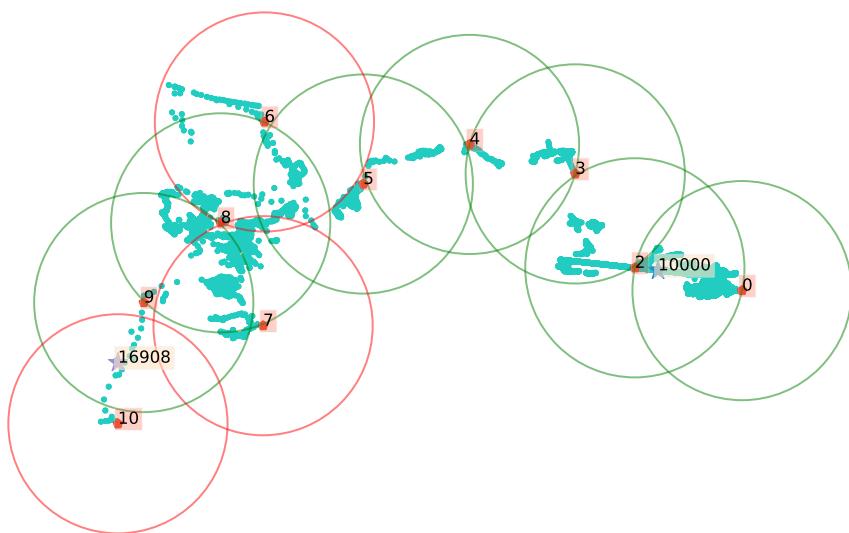
```
clx.explain(17070, plot=True)
```

Output:

The data point 17070 is in group 11, which has been merged into cluster #3.

Similarly, CLASSIX can provide an explanation for why two data points ended up in the same cluster, or why they are in different clusters. In the below example, the shortest path of groups connecting two data points is computed using Dijkstra's algorithm (Dijkstra, 1959):

```
clx.explain(10000, 16908, plot=True)
```

Output:

The data point 10000 is in group 0 and the data point 16908 is in group 9, both of which were merged into cluster #1.

These two groups are connected via groups 0 <-> 2 <-> 3 <-> 4 <-> 5 <-> 8 <-> 9.

References

- Anderson, E., 1936. The species problem in iris. Annals of the Missouri Botanical Garden 23, 457–509.
- Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J., 1999. OPTICS: Ordering points to identify the clustering structure, in: ACM SIGMOD International Conference on Management of Data, pp. 49–60.
- Arthur, D., Vassilvitskii, S., 2006. How slow is the k -means method?, in: Symposium on Computational Geometry, ACM. pp. 144–153.

- Arthur, D., Vassilvitskii, S., 2007. k-means++: The advantages of careful seeding, in: ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics. pp. 1027–1035.
- Auger, N., Jugé, V., Nicaud, C., Pivoteau, C., 2018. On the worst-case complexity of TimSort, in: Annual European Symposium on Algorithms, pp. 4:1–4:13.
- Bensmail, H., DeGennaro, R.P., 2004. Analyzing imputed financial data: a new approach to cluster analysis. FRB Atlanta Working Paper. Federal Reserve Bank of Atlanta.
- Campello, R.J.G.B., Moulavi, D., Sander, J., 2013. Density-based clustering based on hierarchical density estimates, in: Advances in Knowledge Discovery and Data Mining, Springer. pp. 160–172.
- Campello, R.J.G.B., Moulavi, D., Zimek, A., Sander, J., 2015. Hierarchical density estimates for data clustering, visualization, and outlier detection. ACM Transactions on Knowledge Discovery from Data 10, 1–51.
- Chang, H., Yeung, D.Y., 2008. Robust Path-Based Spectral Clustering. Pattern Recognition 41, 191–203.
- Charytanowicz, M., Niewczas, J., Kulczycki, P., Kowalski, P.A., Łukasik, S., Źak, S., 2010. Complete gradient clustering algorithm for features analysis of x-ray images, in: Information Technologies in Biomedicine, Springer. pp. 15–24.
- Chen, X., Liu, W., Qiu, H., Lai, J., 2011. APSCAN: A parameter free algorithm for clustering. Pattern Recognition Letters 32, 973–986.
- Cheng, Y., 1995. Mean shift, mode seeking, and clustering. Transactions on Pattern Analysis and Machine Intelligence 17, 790–799.
- Comaniciu, D., Meer, P., 2002. Mean shift: A robust approach toward feature space analysis. Transactions on Pattern Analysis and Machine Intelligence 24, 603–619.
- de Oliveira, D.P., Garrett, J.H., Soibelman, L., 2011. A density-based spatial clustering approach for defining local indicators of drinking water distribution pipe breakage. Advanced Engineering Informatics 25, 380–389.
- Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271.
- Dua, D., Graff, C., 2017. UCI machine learning repository. URL: <http://archive.ics.uci.edu/ml>.
- Elkan, C., 2003. Using the triangle inequality to accelerate k-means, in: International Conference on International Conference on Machine Learning, AAAI Press. pp. 147–153.
- Erman, J., Arlitt, M., Mahanti, A., 2006. Traffic classification using clustering algorithms, in: SIGCOMM Workshop on Mining Network Data, ACM. pp. 281–286.
- Ester, M., Kriegel, H.P., Sander, J., Xu, X., 1996. A density-based algorithm for discovering clusters in large spatial databases with noise, in: International Conference on Knowledge Discovery and Data Mining, AAAI Press. pp. 226–231.
- Fisher, R.A., 1936. The use of multiple measurements in taxonomic problems. Annals of Eugenics 7, 179–188.
- Forina, M., Leardi, R., C, A., Lanteri, S., 1998. PARVUS: An extendable package of programs for data exploration, classification and correlation. Journal of Chemometrics 4, 191–193.
- Frey, B.J.J., Dueck, D., 2007. Clustering by passing messages between data points. Science 315, 972–976.
- Fu, L., Medico, E., 2007. FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data. BMC Bioinformatics 8, 3–3.
- Galler, B.A., Fisher, M.J., 1964. An improved equivalence algorithm. Communications of the ACM 7, 301—303.
- Gan, J., Tao, Y., 2015. DBSCAN revisited: Mis-claim, un-fixability, and approximation, in: ACM SIGMOD International Conference on Management of Data, pp. 519–530.
- Gionis, A., Mannila, H., Tsaparas, P., 2007. Clustering aggregation. ACM Transactions on Knowledge Discovery from Data 1, 4–es.
- Golub, G.H., Loan, C.F.V., 2013. Matrix Computations. 4 ed., Johns Hopkins University Press Baltimore.
- Gunawan, A., 2013. A faster algorithm for DBSCAN. Master's thesis. Technische University Eindhoven.
- Güvenir, H.A., Demiröz, G., İlter, N., 1998. Learning differential diagnosis of erythema-squamous diseases using voting feature intervals. Artificial Intelligence in Medicine 13, 147–65.
- Hahsler, M., Piekenbrock, M., Doran, D., 2019. dbscan: Fast density-based clustering with R. Journal of Statistical Software 91.
- Har-Peled, S., Indyk, P., Motwani, R., 2012. Approximate nearest neighbor: Towards removing the curse of dimensionality. Theory of Computing 8, 321–350.
- Hastie, T., Tibshirani, R., Friedman, J., 2009. The Elements of Statistical Learning: Data Mining, Inference and Prediction. 2 ed., Springer.
- Hoare, C.A., 1962. Quicksort. The Computer Journal 5, 10–16.
- Hopcroft, J., Tarjan, R., 1973. Algorithm 447: efficient algorithms for graph manipulation. Communications of the ACM 16, 372–378.
- Hotelling, H., 1933. Analysis of a complex of statistical variables into principal components. Journal of Educational Psychology 24, 498–520.
- Hubert, L., Arabie, P., 1985. Comparing partitions. Journal of Classification 2, 193–218.
- Indyk, P., Motwani, R., 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality, in: ACM Symposium on Theory of Computing, pp. 604–613.
- Jain, A.K., 2010. Data clustering: 50 years beyond k-means. Pattern Recognition Letters 31, 651–666.
- Jain, A.K., Law, M.H.C., 2005. Data Clustering: A User's Dilemma, in: Pattern Recognition and Machine Intelligence, Springer. pp. 1–10.
- Jang, J., Jiang, H., 2019. DBSCAN++: Towards fast and scalable density clustering, in: International Conference on Machine Learning, PMLR. pp. 3019–3029.
- Jiang, H., Jang, J., Kpotufe, S., 2018. Quickshift++: Provably good initializations for sample-based mean shift, in: International Conference on Machine Learning, PMLR. pp. 2294–2303.
- Kaufman, L., Rousseeuw, P.J., 1990. Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons, Ltd.
- Kriegel, H.P., Schubert, E., Zimek, A., 2017. The (black) art of runtime evaluation: Are we comparing algorithms or implementations? Knowledge and Information Systems 52, 341–378.
- Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, pp. 1097–1105.
- Li, S., 2011. Concise formulas for the area and volume of a hyperspherical cap. Asian Journal of Mathematics & Statistics 4, 66–70.

- Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L., 2014. Microsoft COCO: Common objects in context, in: European Conference on Computer Vision, Springer. pp. 740–755.
- Lloyd, S.P., 1982. Least squares quantization in PCM. *Transactions on Information Theory* 28, 129–137.
- McInnes, L., Healy, J., 2017. Accelerated hierarchical density based clustering, in: International Conference on Data Mining Workshops, IEEE. pp. 33–42.
- Murphy, K.P., 2012. Machine Learning: A Probabilistic Perspective. MIT Press.
- Musser, D.R., 1997. Introspective sorting and selection algorithms. *Software: Practice and Experience* 27, 983–993.
- Nakai, K., Kanehisa, M., 1991. Expert system for predicting protein localization sites in gram-negative bacteria. *Proteins* 11, 95–110.
- Nakai, K., Kanehisa, M., 1992. A knowledge base for predicting protein localization sites in eukaryotic cells. *Genomics* 14, 897–911.
- Oster, K., Gütterl, S., Shapiro, J.L., Chen, L., Jobson, M., 2021. Pre-treatment of outliers and anomalies in plant data: Methodology and case study of a vacuum distillation unit. [arXiv:2106.14641](https://arxiv.org/abs/2106.14641).
- Patwary, M.M.A., Palsetia, D., Agrawal, A., Liao, W., Manne, F., Choudhary, A., 2012. A new scalable parallel DBSCAN algorithm using the disjoint-set data structure. *International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–11.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12, 2825–2830.
- Rodriguez, A., Laio, A., 2014. Clustering by fast search and find of density peaks. *Science* 344, 1492–1496.
- Schubert, E., Sander, J., Ester, M., Kriegel, H.P., Xu, X., 2017. DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems* 42, 1–21.
- Sculley, D., 2010. Web-scale k-means clustering, in: International Conference on World Wide Web, ACM. pp. 1177–1178.
- Shi, J., Malik, J., 2000. Normalized cuts and image segmentation. *Transactions on Pattern Analysis and Machine Intelligence* 22, 888–905.
- Simonyan, K., Vedaldi, A., Zisserman, A., 2014. Deep inside convolutional networks: Visualising image classification models and saliency maps, in: Workshop at International Conference on Learning Representations, pp. 1–8.
- Song, M., Zhong, H., 2020. Efficient weighted univariate clustering maps outstanding dysregulated genomic zones in human cancers. *Bioinformatics* 36, 5027–5036.
- Udell, M., Townsend, A., 2019. Why are big data matrices approximately low rank? *SIAM Journal on Mathematics of Data Science* 1, 144–160.
- Vedaldi, A., Soatto, S., 2008. Quick shift and kernel methods for mode seeking, in: European Conference on Computer Vision, pp. 705–718.
- Veenman, C., Reinders, M., Backer, E., 2002. A maximum variance cluster algorithm. *Transactions on Pattern Analysis and Machine Intelligence* 24, 1273–1280.
- Vinh, N.X., Epps, J., Bailey, J., 2010. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *The Journal of Machine Learning Research* 11, 2837–2854.
- Yu, S.X., Shi, J., 2003. Multiclass spectral clustering, in: International Conference on Computer Vision, IEEE. pp. 313–319.
- Zahn, C., 1971. Graph-theoretical methods for detecting and describing gestalt clusters. *Transactions on Computers* C-20, 68–86.
- Zhang, T., Ramakrishnan, R., Livny, M., 1996. BIRCH: An efficient data clustering method for very large databases, in: ACM SIGMOD International Conference on Management of Data, pp. 103–114.



Xinye Chen is a third-year Ph.D. student in Numerical Analysis in the Department of Mathematics at the University of Manchester. He is working on topics in graph representation learning and pattern recognition.



Stefan Gütterl is Professor of Applied Mathematics at the University of Manchester. His work focuses on computational mathematics, including numerical algorithms for large-scale linear algebra problems arising with differential equations and in data science. He is Associate Editor for the SIAM Journal on Scientific Computing and Electronic Transactions on Numerical Analysis. He has been awarded the 2021 SIAM James H. Wilkinson Prize in Numerical Analysis and Scientific Computing and holds a Fellowship with the UK's Alan Turing Institute.