DBSCAN:FAST DENSITY-BASED CLUSTERING

R.NANDHAKUMAR[1] and **Dr.ANTONY SELVADOSS THANAMANI** [2]

[1.] *Assistant Professor, Department of Computer Science, Nallamuthu Gounder Mahalingam College, Pollachi-642001, India*

[2.] *Associate Professor & Head, Department of Computer Science, Nallamuthu Gounder Mahalingam College, Pollachi-642001, India*

Abstract

This article describes the implementation and use of the R package dbscan, which provides complete and fast implementations of the popular density-based clustering algorithm DBSCAN and the augmented ordering algorithm OPTICS. Compared to other implementations, dbscan offers open-source implementations using C++ and advanced data structures like k-d trees to speed up computation. An important advantage of this implementation is that it is up-to-date with several primary advancements that have been added since their original publications, including artifact corrections and dendrogram ex- traction methods for OPTICS. Experiments with dbscan's implementation of DBSCAN and OPTICS compared and other libraries such as FPC, ELKI, WEKA, PyClustering, SciKit-Learn and SPMF suggest that dbscan provides a very efficient implementation.

Keywords: DBSCAN, OPTICS, Density-based Clustering, Hierarchical Clustering.

1. INTRODUCTION

Clustering is typically described as the process of finding structure in data by grouping similar objects together, where the resulting set of groups are called clusters. Many clustering algorithms directly apply the idea that clusters can be formed such that objects in the same cluster should be more similar to each other than to objects in other clusters. The notion of similarity (or distance) stems from the fact that objects are assumed to be data points embedded in a data space in which a similarity measure can be defined. Examples are methods based on solving the k-means problem or mixture models which find the parameters of a parametric generative probabilistic model

from which the observed data are assumed to arise. Another approach is hierarchical clustering, which uses local heuristics to form a hierarchy of nested grouping of objects. Most of these approaches (with the notable exception of single- link hierarchical clustering) are biased towards clusters with convex, hyper-spherical shape. A detailed review of these clustering algorithms is provided in Kaufman and Rousseeuw (2017), Jain, Murty, and Flynn (2016), and the more recent review by Aggarwal and Reddy (2018).

Density-based clustering approaches clustering differently. It simply posits that clusters are contiguous 'dense' regions in the data space (i.e., regions of high point density), separated by areas of low point density (Kriegel, Kröger, Sander, and Arthur 2017; Sander 2017). Density- based methods find such high-density regions representing clusters of arbitrary shape and typically have a structured means of identifying noise points in low-density regions. These properties provide advantages for many applications compared to other clustering approaches. For example, geospatial data may be fraught with noisy data points due to estimation errors in GPS-enabled sensors (Chen, Ji, and Wang 2014) and may have unique cluster shapes caused by the physical space the data was captured in. Density-based clustering is also a promising approach to clustering high-dimensional data (Kailing, Kriegel, and Kröger 2014), where partitions are difficult to discover, and where the physical shape constraints assumed by model-based methods are more likely to be violated.

Several density-based clustering algorithms have been proposed, including DBSCAN algorithm DENCLUE and many DBSCAN derivates like HDBSCAN (Campello, Moulavi, Zimek, and Sander 2018). These clustering algorithms are widely used in practice with applications ranging from finding outliers in datasets for fraud prevention, to finding patterns in streaming data, noisy signals, gene expression data (Jiang, Pei, and Zhang 2018), multimedia databases (Kisilevich, Mansmann, and Keim 2017), and road traffic (Li, Han, Lee, and Gonzalez 2017).

This paper focuses on an efficient implementation of the DBSCAN algorithm,one of the most popular density-based clustering algorithms-OPTICS often referred to as an extension of DBSCAN. While surveying software tools that implement various density-based clustering algorithms, it was discovered that in a large number of statistical tools, not only do implementations vary significantly in performance (Kriegel, Schubert, and Zimek 2016), but may also lack important components and corrections. Specifically, for the statistical computing environment, only naive DBSCAN implementations without speed-up with spatial data structures are available (e.g., in the well-known Flexible Procedures for Clustering package (Hennig 2017), and OPTICS is not available. Density-based clustering with DBSCAN and related algorithms called dbscan. The dbscan package contains complete, correct and fast implementations of DBSCAN and OPTICS. The package currently enjoys thousands of new installations from the CRAN repository every month.

This article presents an overview of the dbscan focusing on DBSCAN and OPTICS, outlining its operation and experimentally compares its performance with implementations in other open-source implementations. We first review the concept of density-based clustering and present the DBSCAN and OPTICS algorithms in Section 2. This section concludes with a short review of existing software packages that implement these algorithms. Details about dbscan, with examples of its use, are presented in Section 3. A performance evaluation is presented in Section 4. Concluding remarks are offered in Section 5.

2. DENSITY-BASED CLUSTERING

Density-based clustering is now a well-studied field. Conceptually, the idea behind density-based clustering is simple: given a set of data points, define a structure that accurately reflects the underlying density. An important distinction between density-based clustering and alternative approaches to cluster analysis, such as the use of *(Gaussian) mixture models.* is that the latter represents a *parametric* approach in which the observed data are assumed to have been produced by mixture of either Gaussian or other parametric families of distributions. While certainly useful in many applications, parametric approaches naturally assume clusters will exhibit some type convex (generally hyper-spherical or hyper-elliptical) shape. Other approaches, such as k-means clustering (where the k parameter signifies the user-specified number of clusters to find), share this common theme of 'minimum variance', where the underlying assumption is made that ideal clusters are found by minimizing some measure of intra-cluster variance (often referred to as cluster cohesion) and maximizing the inter-cluster variance (cluster separation) (Arbelaitz, Gurrutxaga, Muguerza, Pérez, and Perona 2018). Conversely, the label density-based clustering is used for methods which do not assume parametric distributions, are capable of finding arbitrarily-shaped clusters, handle varying amounts of noise, and require no prior knowledge regarding how to set the number of clusters k. This methodology is best expressed in the DBSCAN algorithm, which we discuss next.

DBSCAN: Density Based Spatial Clustering of Applications with Noise

As one of the most cited of the density-based clustering algorithms (Microsoft Academic Search 2016), DBSCAN is likely the best known density-based clustering algorithm in the scientific community today. The central idea behind DBSCAN and its extensions and revisions is the notion that points are assigned to the same cluster if they are *density-reachable* from each other. To understand this concept, we will go through the most important definitions used in DBSCAN and related algorithms.

Clustering starts with a dataset D containing a set of points p€D. Density-based algorithms need to obtain a density estimate over the data space. DBSCAN estimates the density around a point using the concept of €-neighborhood.

Definition 1: €-Neighborhood. The €-neighborhood, $N_{€}(p)$, of a data point p is the set of points within a specified radius € around p.

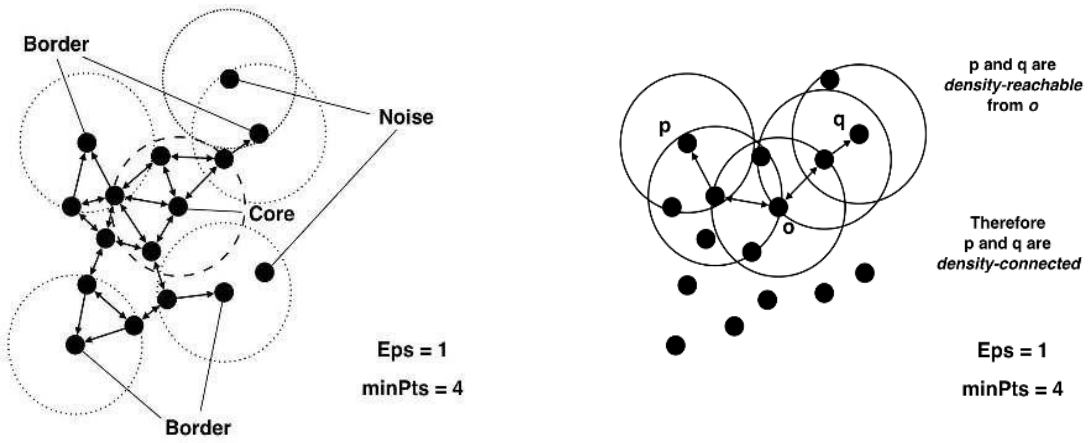
$$N_{€}p = \{q \mid d(p, q) < €\}$$

where d is some distance measure and € R^{+}. Note that the point p is always in its own €-neighborhood, i.e., p € $N_{€}(p)$ always holds.

Following this definition, the size of the neighborhood $|N_{€}(p)|$ can be seen as a simple un-normalized kernel density estimate around p using a uniform kernel and a bandwidth of €. DBSCAN uses $N_{€}(p)$ and a threshold called *minPts* to detect dense regions and to classify the points in a data set into core, border, or noise points.

Definition 2: Point classes. A point p €D is classified as

- a core point if $N_{€}(p)$ has high density, i.e., $|N_{€}(p)| \geq$ minPts where minPts $\in Z^{+}$ is a user-specified density threshold,



(a) (b)

Concepts used the DBSCAN family of algorithms. (a) shows examples for the three point classes, core, border, and noise points, (b) illustrates the concept of density-reach ability and density-connectivity.

- *a* border point *if* p *is not a core point, but it is in the neighborhood of a core point* q ∈ D, *i.e.*, p ∈ $N_{€}(q)$, *or*
- *a* noise point, *otherwise.*

A visual example is shown in Figure 1(a). The size of the neighborhood for some points is shown as a circle and their class is shown as an annotation.

To form contiguous dense regions from individual points, DBSCAN defines the notions of reach ability and connectedness.

Definition 3: Directly density-reachable. *A point* q €D *is directly density-reachable from a point* p €D *with respect to* € *and minPts if, and only if,*

1. $|N_{€}(p)| \geq minPts$, *and*
2. q $€N_{€}(p)$.

That is, p *is a core point and* q *is in its* €*-neighborhood.*

Definition 4: Density-reachable. *A point* p *is density-reachable from* q *if there exists in* D *an ordered sequence of points* $(p_1, p_2, ..., p_n)$ *with* $q = p_1$ *and* $p = p_n$ *such that* $p_i + 1$ *directly density-reachable from* p_i ¥ i € $\{1, 2, ..., n-1\}$.

Definition 5: Density-connected. *A point* p € D *is density-connected to a point* q €D *if there is a point* o €D *such that both* p *and* q *are density-reachable from* o.

The notion of density-connection can be used to form clusters as contiguous dense regions.

Definition 6: Cluster. *A cluster* C *is a non-empty subset of* D *satisfying the following conditions:*

1. Maximality*: If* p € C *and* q *is density-reachable from* p, *then* q € C*; and*
2. Connectivity*:* ¥ p, q €C, p *is density-connected to* q.

The DBSCAN algorithm identifies all such clusters by finding all core points and expanding each to all density-reachable points. The algorithm begins with an arbitrary point p and retrieves its € -neighborhood. If it is a core point then it will start a new cluster that is expanded by assigning all points in its neighborhood to the cluster. If an additional core point is found in the neighborhood, then the search is expanded to include also all points in its neighborhood. If no more core points are found in the expanded neighborhood, then the cluster is complete and the remaining points are searched to see if another core point can be found to start a new cluster. After processing all points, points which were not assigned to a cluster are considered noise.

In the DBSCAN algorithm, core points are always part of the same cluster, independent of the order in which the points in the dataset are processed. This is different for border points. Border points might be density-reachable from core points in several clusters and the algorithm assigns them to the first of these clusters processed which depends on the order of the data points and the particular implementation of the algorithm. To alleviate this behavior, suggest a modification called DBSCAN* which considers all border points as noise instead and leaves them unassigned.

OPTICS: Ordering Points To Identify Clustering Structure

There are many instances where it would be useful to detect clusters of varying density. From identifying causes among similar seawater characteristics (Birant and Kut 2018), to network intrusion detection systems, point of interest detection using geo-tagged photos (Kisilevich *et al.* 2018),

Classifying cancerous skin lesions (Celebi, Aslandogan, and Bergstresser 2017), the motivations for detecting clusters among varying densities are numerous. The inability to find clusters of varying density is a notable drawback of DBSCAN resulting from the fact that a combination of a specific neighborhood size with a single density threshold minPts is used to determine if a point resides in a dense neighborhood.

The original DBSCAN authors developed OPTICS (Ankerst *et al.* 1999) to address this concern. OPTICS borrows the core density-reachable concept from DBSCAN. But while DBSCAN may be thought of as a clustering algorithm, searching for natural groups in data, OPTICS is an augmented ordering algorithm from which either flat or hierarchical clustering results can be derived. OPTICS requires the same € and *minPts* parameters as DBSCAN, however, the € parameter is theoretically unnecessary and is only used for the practical purpose of reducing the runtime complexity of the algorithm.

To describe OPTICS, we introduce an additional concepts called core-distance and reachability-distance. All used distances are calculated using the same metric (often Euclidean distance) used for the neighborhood calculation.

Definition 7: Core-distance. The core-distance of a point p ∈ D with respect to minPts and € is defined as

$$\text{core-dist}(p; \epsilon, minPts) = \begin{cases} UNDEFINED & \text{if } |N_\epsilon(p)| < minPts, \text{and} \\ \text{minPts-dist}(p) & \end{cases}$$

where minPts-dist(p) *is the distance from* p *to its minPts* − 1 *nearest neighbor, i.e., the minimal radius a neighborhood of size minPts centered at and including* p *would have.*

The reachability-distance of a core point p with respect to object q is the smallest neighbor- hood radius such that p would be directly density-reachable from q. Note that € is typically set very large compared to DBSCAN. Therefore, *minPts* behaves differently for OPTICS: more points will be considered core points and it affects how many nearest neighbors are considered in the core-distance calculation, where larger values will lead to larger and smoother reachability distributions. This needs to be kept in mind when choosing appropriate parameters.

OPTICS provides an augmented ordering. The algorithm starting with a point and expands its neighborhood like DBSCAN, but it explores the new point in the order of lowest to highest core-distance. The order in which the points are explored along with each point's core- and reachability-distance is the final result of the algorithm.

An example of the order and the resulting reachability-distance is shown in the form of a reachability plot in Figure 2. Low reachability-distances shown as valleys represent clusters separated by peaks represent- ing points with larger distances. This density representation essentially conveys the same information as the often used dendrogram or 'tree-like' structure. This is why OPTICS is often also noted as a visualization tool. showed how the output of OPTICS can be

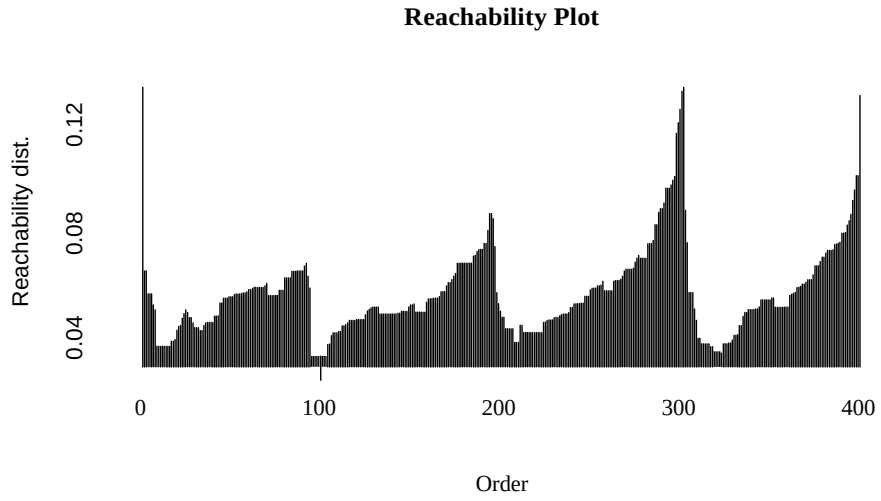
converted into an equivalent dendrogram, and that under certain conditions, the dendrogram produced by the well known hierarchical clustering with single linkage is identical to running OPTICS with the parameter *minPts* = 2.

From the order discovered by OPTICS, two ways to group points into clusters was discussed in one which we will refer to as the Extract DBSCAN method and one which we will refer to as the Extract-ξ method summarized below:

1. Extract DBSCAN uses a single global reachability-distance threshold €to extract a clustering. This can be seen as a horizontal line in the reachability plot in 2. Peaks above the cut-off represent noise points and separate the clusters.

2. Extract-ξ identifies clusters *hierarchically by* scanning through the ordering that OP- TICS produces to identify significant, relative changes in reachability distance. The authors of OPTICS noted that clusters can be thought of as identifying 'dents' in the reachability plot.

The Extract DBSCAN method extracts a clustering equivalent to DBSCAN* (i.e., DBSCAN where border points stay unassigned). Because this method extracts clusters like DBSCAN,



OPTICS reachability plot example for a data set with four clusters of 100 data points each.

it cannot identify partitions that exhibit very significant differences in density. Clusters of significantly different density can only be identified if the data is well separated and very little noise is present. The second method, which we call Extract-ξ[1], identifies a cluster hierarchy and replaces the data dependent global parameter with ξ, a data-independent density- threshold parameter ranging between 0 and 1. One interpretation of ξ is that it describes the relative magnitude of the change of cluster density (i.e., reachability). Significant changes in relative reachability allow for clusters to manifest themselves hierarchically as 'dents' in the ordering structure. The hierarchical representation Extract-ξ can, as opposed to the Extract DBSCAN method, produce clusters of varying densities.

With its two ways of extracting clusters from the ordering, whether through either the global €or relative ξ threshold, OPTICS can be seen as a generalization of DBSCAN. In contexts where one wants to find clusters of similar density, OPTICS's Extract DBSCAN yields a DBSCAN-like solution, while in other contexts Extract-ξ can generate hierarchy representing clusters of varying density. It is thus interesting to note that while DBSCAN has reached critical acclaim, even motivating numerous extensions, OPTICS has received decidedly less attention. Perhaps one of the reasons for this is because the Extract-ξ method for grouping points into clusters has gone largely unnoticed, as it is not implemented in most open-source software packages that advertise an implementation of OPTICS. This includes implementations in WEKA, SPMF and the PyClustering and Scikit-learn libraries for Python. To the best of our knowledge, the only other open-source library sporting a complete implementation of OPTICS, to add complete implementations to existing software packages and introduce new complete implementations of OPTICS like the dbscan .

3.The dbscan package

The package dbscan provides high performance code for DBSCAN , OPTICS and fixed- radius nearest neighbor search speed. DBSCAN and OPTICS share a similar interface.

```
dbscan(x, eps, minPts = 5, weights = NULL, borderPoints = TRUE, ...)
optics(x, eps, minPts = 5, ...)
```

DBSCAN* to consider border points as noise. This can be achieved by using borderPoints = FALSE. All functions accept additional arguments. These arguments are passed on to the fixed-radius nearest neighbor search. More details about the implementation of the nearest neighbor search.

Clusters can be extracted from the linear order produced by OPTICS. The dbscan implementation of the cluster extraction methods for Extract DBSCAN and Extract-ξ are:

```
extractDBSCAN(object, eps_cl)
extractXi(object, xi, minimum = FALSE, correctPredecessor = TRUE)
```

extract DBSCAN() extracts a clustering from an OPTICS ordering that is similar to what DBSCAN would produce with a single global ϱ set to eps_cl. extractXi() extracts clusters hierarchically based on the steepness of the reachability plot. minimum controls whether only the minimal (non-overlapping) cluster are extracted. Correct Predecessor corrects a common artifact known of the original ξ method , the steep up area for points that have predecessors not in the cluster .

4. Nearest Neighbor Search

The density based algorithms in dbscan rely heavily on forming neighborhoods, i.e., finding all points belonging to an ϱ-neighborhood. A simple approach is to perform a linear search, i.e., always calculating the distances to all other points to find the closest points. This requires O(n) operations,

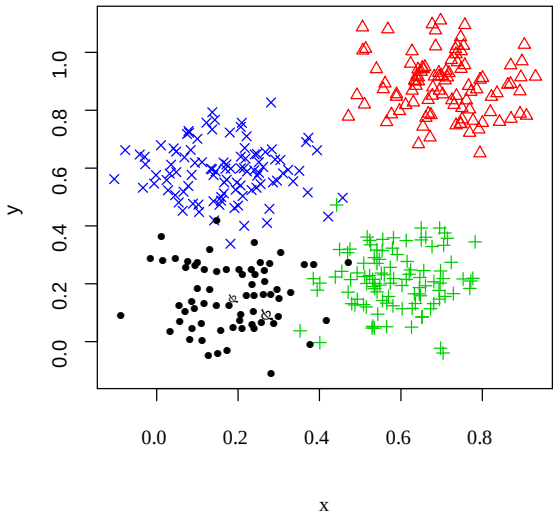
with n being the number of data points, for each time a neighborhood is needed. Since DBSCAN and OPTICS process each data point once, this results in a $O(n^2)$ runtime complexity. A convenient way in R is to compute a distance matrix with all pair wise distances between points and sort the distances for each point (row in the distance matrix) to pre-compute the nearest neighbors for each point. However, this method has the drawback that the size of the full distance matrix is $O(n^2)$, and becomes very large and slow to compute for medium to large data sets.

In order to avoid computing the complete distance matrix, dbscan relies on a space-partitioning data structure called a k-d trees .This data structure allows dbscan to identify the kNN or all neighbors within a fixed radius eps more efficiently in sub-linear time using on average only O(log(n)) operations per query. This results in a reduced runtime complexity of O(nlog(n)). However, note that k-d trees are known to degenerate for high-dimensional data requiring O(n) operations and leading to a performance no better than linear search. Fast kNN search and fixed-radius nearest neighbor search are used in DBSCAN and OPTICS, but we also provide a direct interface in dbscan, since they are useful in their own right.

```
kNN(x,k,sort= TRUE,search="kdtree",bucketSize=10,splitRule = "suggest", approx = 0)

frNN(x,eps, sort = TRUE,search="kdtree",bucketSize=10,splitRule="suggest",approx=0)
```

The interfaces only differ in the way that kNN() requires to specify k while frNN() needs the radius eps. All other arguments are the same. x is the data and the result will be a list of neighbors in x for each point in x. sort controls if the returned points are sorted by distance. search controls what searching method should be used. Available search methods are "kdtree", "linear" and "dist". The linear search method does not build a search data structure, but performs a complete linear search to find the nearest neighbors. The dist method pre-computes a dissimilarity matrix which is very fast for small data sets, but problematic for large sets.



The sample dataset, consisting of 4 noisy Gaussian distributions with slight overlap.

Only nearest neighbors up to a distance of a factor of (1 + approx)eps will be returned, but some actual neighbors may be omitted potentially leading to spurious clusters and noise points. However, the algorithm will enjoy a significant speedup. dbscan() and optics() use internally frNN().

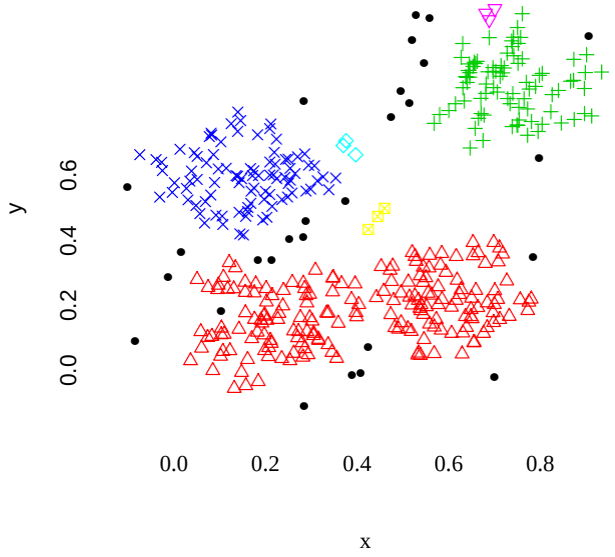
Clustering with DBSCAN

We use a very simple artificial data set of four slightly overlapping Gaussians in two-dimensional space with 100 points each. We load dbscan, set the random number generator to make the results reproducible and create the data set.

```
> library("dbscan")
> set.seed(2)
> n <- 400
> x <- cbind(
+       x = runif(4, 0, 1) + rnorm(n, sd = 0.1),
+       y = runif(4, 0, 1) + rnorm(n, sd = 0.1)
+       )
> true_clusters <- rep(1:4, time = 100)

> plot(x, col = true_clusters, pch = true_clusters)
```

To apply DBSCAN, we need to decide on the neighborhood radius eps and the density threshold minPts. The rule of thumb for minPts is to use at least the number of dimensions of the data set plus one. In our case, this is 3. For eps, we can plot the points' kNN distances (i.e., the distance to the kth nearest neighbor) in decreasing order and look for a knee in the plot. The idea behind this heuristic is that points located inside of clusters will have a small k-nearest neighbor distance, because they are close to other points in the same cluster, while noise points are isolated and will have a rather large kNN distance. dbscan provides a function called kNN distplot() to make this easier.



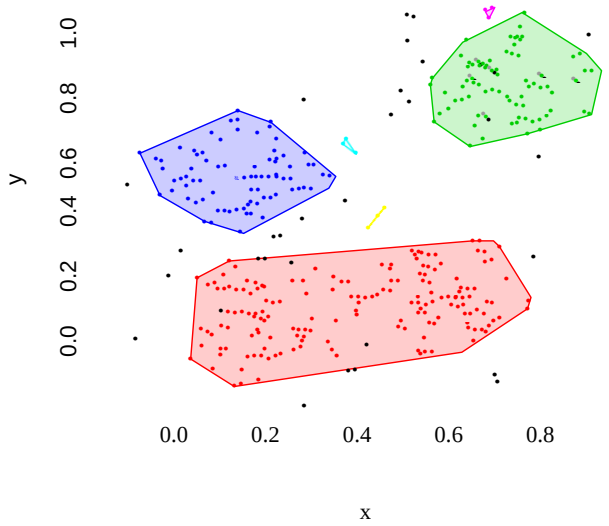
Result of clustering with DBSCAN. Noise is represented as black circles

The resulting clustering identified one large cluster with 185 member points and 2 medium size clusters of between 87 and 89 points, three very small clusters and 30 noise points (represented by cluster id 0). The available fields can be directly accessed using the list extraction operator $. For example, the cluster assignment information can be used to plot the data with the clusters identified by different labels and colors.

```
> plot(x, col = res$cluster + 1L, pch = res$cluster + 1L)
```

The scatter plot in Figure shows that the clustering algorithm correctly identified the upper two clusters, but merged the lower two clusters because the region between them has a high enough density. The small clusters are isolated groups of 3 points (passing *minPts*) and the noise points isolated points. dbscan also provides a plot that adds convex cluster hulls to the scatter plot shown .

Convex Cluster Hulls



Convex hull plot of the DBSCAN clustering. Noise points are black. Note that noise points and points of another cluster may lie within the convex hull of a different cluster.

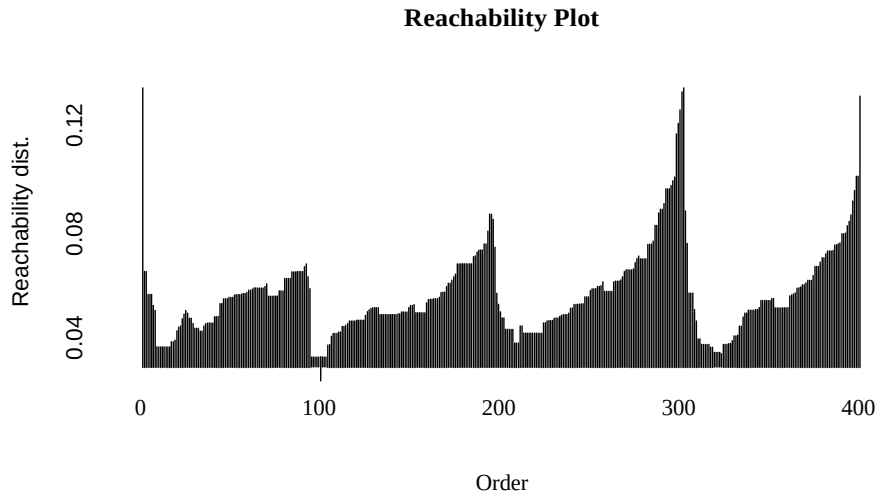
Clustering with OPTICS

Unless OPTICS is purely used to extract a DBSCAN clustering, its parameters have a different effect than for DBSCAN: `eps` is typically chosen rather large (we use 10 here) and `minPts` mostly affects core and reachability-distance calculation, where larger values have a smoothing effect. We use also 10, i.e., the core-distance is defined as the distance to the 9th nearest neighbor (spanning a neighborhood of 10 points).

```
> res <- optics(x, eps = 10, minPts = 10)
> res
OPTICS ordering/clustering for 400 objects.
Parameters: minPts =10, ep = 10, eps_cl = NA, xi = NA
Available fields: order, reachdist, coredist, predecessor, minPts,
                  eps, eps_cl, xi
```

OPTICS is an augmented ordering algorithm, which stores the computed order of the points it found in the order element of the returned object.

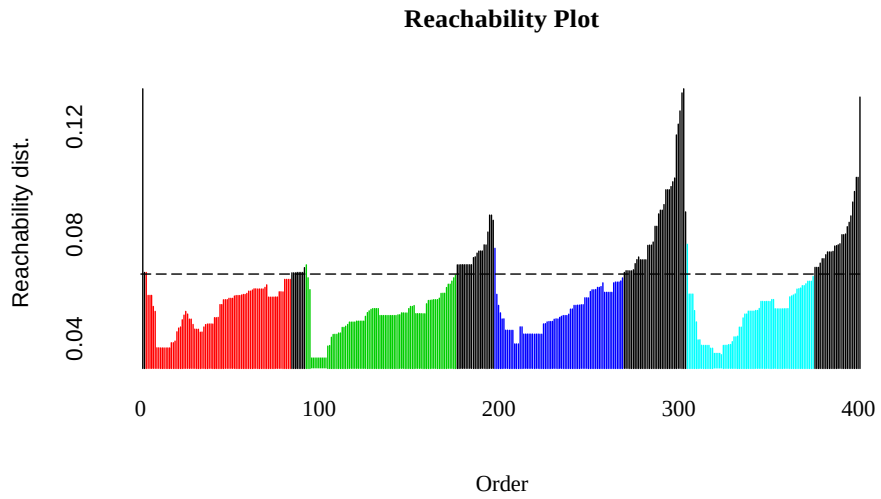
```
> head(res$order, n = 15)
```



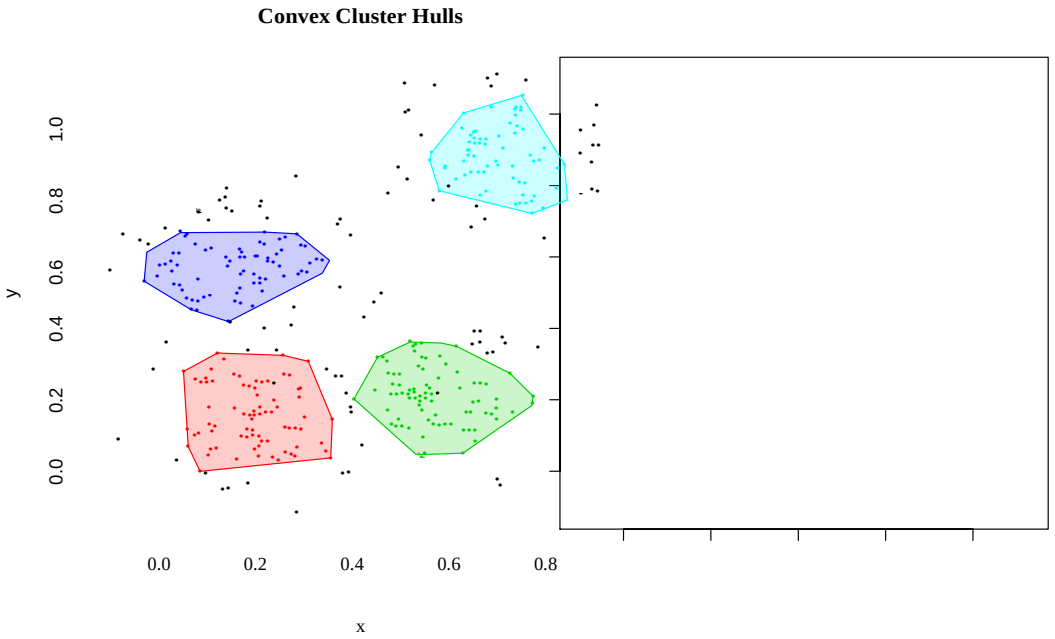
OPTICS reachability plot. Note that the first reachability value is always UNDE- FINED.

The reachability plot in this Figure shows the reachability distance for points ordered by OPTICS. Valleys represent potential clusters separated by peaks. Very high peaks may indicate noise points. To visualize the order on the original data sets we can plot a line connecting the points in order.

```
> plot(x, col = "grey")
> polygon(x[res$order,], )
```



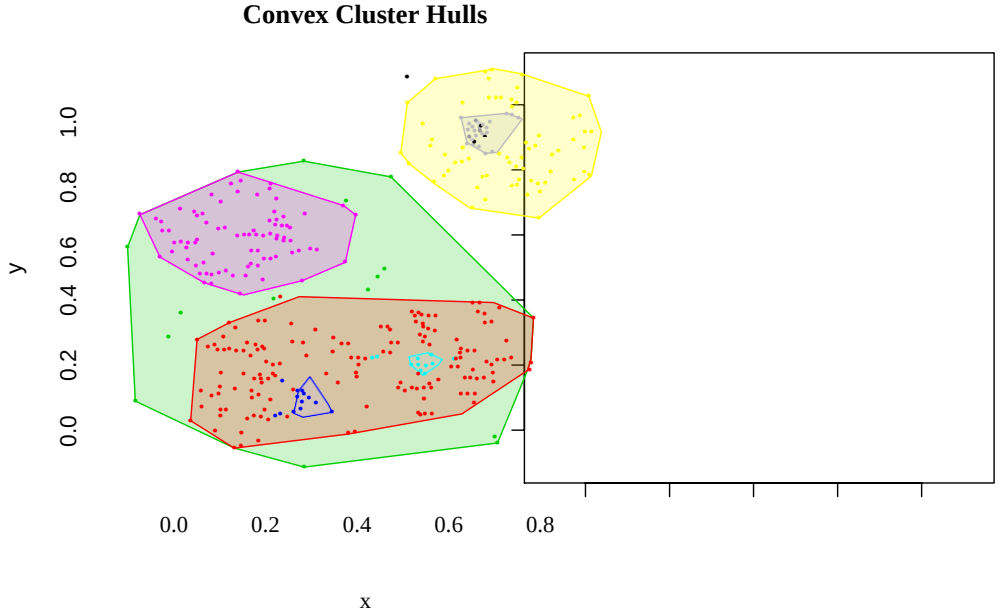
Reachability plot for a DBSCAN-type clustering extracted at global ϱ = 0.065 results in four clusters.



Convex hull plot for a DBSCAN-type clustering extracted at global ϱ = 0.065 results in four clusters.

Performance Comparison

Finally, we evaluate the performance of dbscan's implementation of DBSCAN and OPTICS against other open-source implementations. This is not a comprehensive evaluation study, but is used to demonstrate the performance of dbscan's DBSCAN and OPTICS implementation



Convex hull plot of a hierarchical clustering extracted with Extract-ξ.

REFERENCES

1. Aggarwal CC, Reddy CK (2018). *Data Clustering: Algorithms and Applications*. 1st edition. Chapman & Hall/CRC. ISBN 1466558210, 9781466558212.

2. Breunig MM, Kriegel HP, Ng RT, Sander J (2016). "LOF: identifying density-based local outliers." In *ACM sigmod record*, volume 29, pp. 93–104. ACM.

3. Campello RJ, Moulavi D, Zimek A, Sander J (2015). "Hierarchical density estimates for data clustering, visualization, and outlier detection." *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(1), 5.

4. Cao F, Ester M, Qian W, Zhou A (2014). "Density-Based Clustering over an Evolving Data Stream with Noise." In *SDM*, volume 6, pp. 328–339. SIAM.

5. Chen W, Ji MH, Wang JM (2014). "T-DBSCAN: A spatiotemporal density clustering for GPS trajectory segmentation." *International Journal of Online Engineering*, 10(6), 19–24. ISSN 18612121.

6. Chen Y, Tu L (2017). "Density-based clustering for real-time stream data." In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 133–142. ACM.
7. Fu L, Medico E (2017). "FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data." *BMC Bioinformatics*.