A simple iterative grid- and density-based Clustering Algorithm

Uwe Stöhr

Soloof EOOD, Sofia, Bulgaria
Email: research @ soloof.com
Project Page: https://codeberg.org/Soloof/Iteridense

Abstract

We introduce ITERIDENSE, an iterative clustering algorithm combining grid-based and density-based methods. It provides two possibilities to perform the clustering and for both it provides a clear path on how to change the algorithm's input parameters to achieve suitable results. We show that ITERIDENSE is applicable for datasets with any dimensionality. ITERIDENSE provides shorter computation times than pure density-based algorithms and that it performs clustering at least as good as the DBSCAN algorithm. We provide a reference implementation of ITERIDENSE as well as a stand-alone program with a graphical user interface.

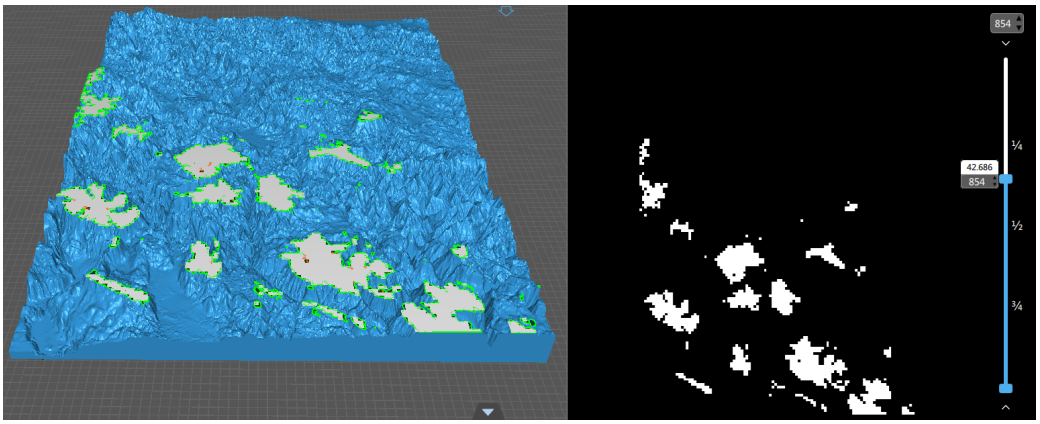
1 Introduction

Density-based clustering algorithms have been proven useful for many practical applications. There exists a wide variety of algorithms, some optimized for particular use cases [1]. However, these algorithms have a major drawback – one needs to evaluate neighboring points for every data point in the dataset to determine if it is part of a cluster. This is the case for most density-based clustering algorithms like PreDeCon [2] or HDBSCAN [3]. Other algorithms that combine grid-based with density-based methods like DENCLUE [4] or CLIQUE [5] don't have this drawback but make assumptions about the shape of the probability distribution of the data (what is the probability to find a data point inside the range of available data).

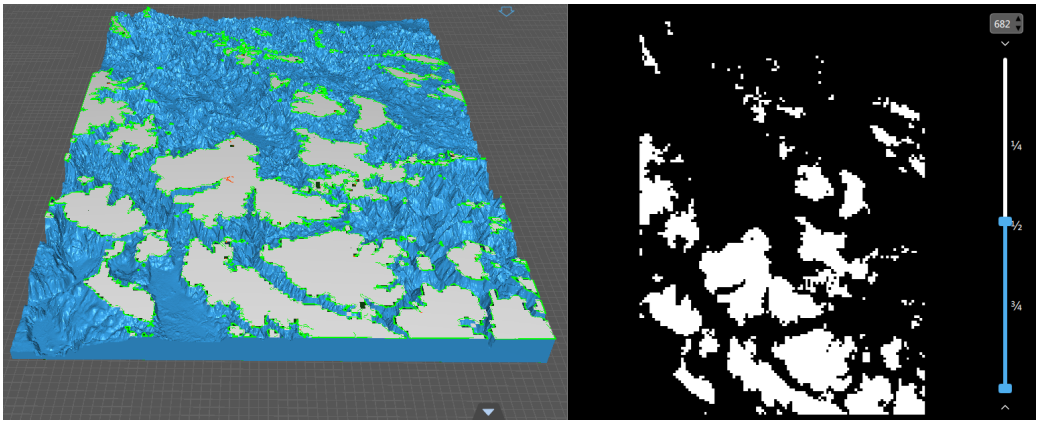
Another issue of many clustering algorithms is that as user there is no clear path on how to change the algorithm's input parameters to achieve a suitable clustering result. Taking for example the algorithm DBSCAN [6], it requires to specify the parameter ϵ, the maximum distance to another core point of a cluster. For many use cases there is no clear path on how to change ϵ to get a suitable result as we will also discuss in this paper.

The algorithm presented in this paper uses both, density-based and grid-based methods. Its main idea is to work with different probability-density functions of the dataset and analyzing them in a grid. This approach makes it possible that even for small datasets in 2 dimensions the computation can be 10 times faster than a pure density-based algorithm like DBSCAN. Our algorithm follows an iterative approach to calculate the probability-density function and makes no assumptions about the shape of that function. Therefore it provides a clear path on how to set a start value of the algorithm's main parameter and how to change it to achieve a certain result.

The algorithm clusters so that no point in a dataset can belong to more than one cluster. And not all points in the dataset must be assigned to clusters.



(a) Cut at 2/3 of its maximal height.



(b) Cut at half of its maximal height.

Figure 1: Relief of a random geographic area cut at different heights.

2 Derivation of the Algorithm

The basic idea of the algorithm is how mountain peak areas are separated from each other in a geographical relief. Take for example the relief shown at the left in Fig. 1. To identify areas around a peak one cuts the relief at a desired height. The cut-off areas are then the mountain peak areas. For example in Fig. 1 (a) the relief was cut at about 2/3 of the maximal height leading to more than a dozen peak areas. In Fig. 1 (b) the relief was cut at about half the maximal size leading to larger areas. By decreasing the cutting height, the number of areas will become fewer unless at a zero height the whole relief area is part of a single mountain area.

Translating the relief height to the density of data points ρ, the cut is made at a certain ρ through the probability-density function of the dataset. See also the section *Background and Motivation* of [1] for a similar visualization than in Fig. 1.

Detecting clusters by making a cut through the probability-distribution is state-of-the-art [1]. The novelty is how to generate the distribution and how to cut it. Instead of calculating a single probability-density function, we calculate different probability-density functions in an iterative process with increasing resolutions. Resolution means hereby into how many cells every dimension of the dataset is divided to calculate the probability-density function. For every resolution a clustering of the cells is performed and one gets two results: the amount of clusters and the density of every cluster. Depending on the results, the algorithm is stopped, or it continues and calculates another probability-density function at a higher resolution.

We call our algorithm ITERIDENSE (ITERative grID- and dENSity-basEd clustering). Based on our approach the key features of ITERIDENSE are:

- The clustering works without the need to test neighbors of every data point, making the clustering more computation-efficient. The clusters are derived by counting the numbers of data points within a certain data area.
- One has two choices to stop the algorithm. Either one specifies ρ, the minimal density every found cluster must have to stop the algorithm, or **MinClusters**, the number of how many clusters should at least be detected (in that case the specification of ρ is not necessary). The possibility to specify **MinClusters** is a big advantage compared to pure density-based algorithms that by design cannot have this feature.
- For ITERIDENSE ρ is defined being normalized so that the whole dataset has $\rho = 1$. Because of the iteration with increasing resolutions our algorithm provides a clear path on how to set and change ρ: Start with a low $\rho > 1$ and increase ρ until you get a suitable result. The algorithm stops at a resolution that fulfills the specification of ρ. We will demonstrate this feature with an example in Sec. 4.2.

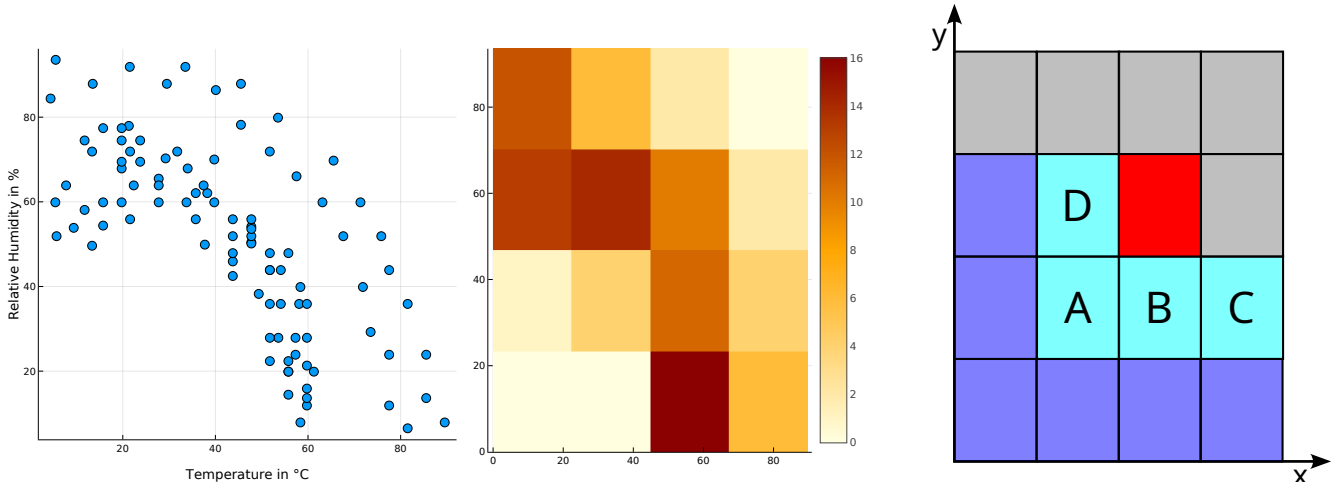
3 Description of the Iteridense Clustering Algorithm

3.1 Basic Algorithm

There are two possible input parameters to the algorithm, either to specify how many clusters should at least be detected (**MinClusters**) or the minimal data point density ρ to form a cluster. ρ is treated as a dimensionless number since the dimension of the data point space can be anything, depending on the source of the data points.

Fig. 2 (a) shows as example a 2-dimensional dataset which represents the relative humidity and temperature of air inside a thermal cabinet after a certain chemical process. It looks like there might be a cluster of points inside an area in form of a quarter circle. To find that potential cluster ITERIDENSE works on this dataset the following way:

1) The space of the dataset is divided into a grid of 4×4 cells. In our example the data range in dimension 1 "temperature" is 4 – 89 °C and the range in dimension 2 "humidity" is 6 – 93 %. These ranges define the space of the dataset. As we divide into 4×4 cells, cell 1 covers the space of temperature range 4 – 25.25 °C and humidity range 6 – 27.75 %.
   A division into 4×4 cells is defined as a resolution of 4. A division into 5×5 cells would be a resolution of 5 and so on.
2) The number of data points in the cells are counted. The result is a count map as shown in Fig. 2 (b).
3) Now the actual clustering is performed. The evaluating scheme is depicted in Fig. 2 (c). Every cell is evaluated one after another first in x- then in y-direction. The most basic definition of a cluster is that a cluster consists of at least one cell that has at least 2 data points. Therefore, if a cell has more than 1 data point it could either be a sole cluster or part of a cluster. To decide this, its neighboring cells are evaluated. Thereby only those neighbors are evaluated that have already been evaluated (blue and light-blue in Fig. 2 (c)) because for them it is already known to which cluster they belong to.
   If none of the neighbor cells A – D are in a cluster, the current cell will start a new cluster. If any neighbor is in a cluster, the current cell will become part of that cluster. If neighbor cells are in different clusters, these clusters are merged and the current cell becomes part of that merged cluster. For example cell A and cell C could be in different clusters and the current cell unites both clusters.



(a) Data to be clustered.

(b) Count map generated in algorithm step 3.

(c) Clustering scheme of the count map.

Figure 2: Clustering process of ITERIDENSE. (a) Data to be clustered; (b) Count map generated in algorithm step 3 for resolution 4: 16 cells with the info how many data points are in.; (c) Clustering scheme of the count map. The red cell is the currently evaluated cell, the blue cells have already be evaluated, cells A – D are neighboring cells determining the clustering for the red cell.

4) The density of every cluster ρ_{cluster} is calculated as the number of points in the cluster divided by the number of cells in the cluster.

$$\rho_{\text{cluster}} = \frac{\text{num points in cluster}}{\text{num cells of cluster}} \tag{1}$$

To make ρ_{cluster} independent of the resolution, it has to be the normalized.
For dimension > 3:

$$\rho_{\text{cluster}} = \frac{\text{num points in cluster}}{\text{num cells of cluster}} \cdot \frac{\text{total num of points}}{\text{total num of cells}} \cdot \frac{\text{dimension}}{2\,\text{resolution}^{\text{dimension}-2}} \tag{2}$$

For dimension < 3:

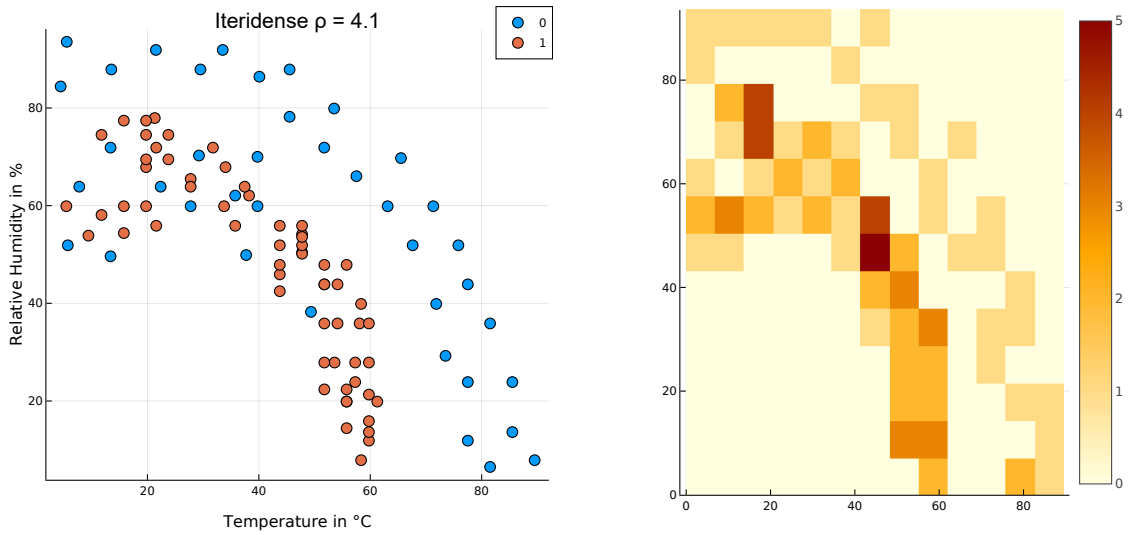
$$\rho_{\text{cluster}} = \frac{\text{num points in cluster}}{\text{num cells of cluster}} \cdot \frac{\text{total num of points}}{\text{total num of cells}} \tag{3}$$

The normalization is derived in Sec. 3.3. For less than 2 dimensions ρ_{cluster} expresses by what factor the cluster is more dense than the whole dataset.
The final density ρ_{final} is the minimum of all ρ_{cluster}.

5) All clusters are evaluated. Optionally clusters with ρ_{cluster} lower than a specified value will be deleted. If they contain fewer data points than a specified value, they are deleted as well. See the next section for a description of these optional settings. The result of this algorithm step is a set of clusters that are subsequently numbered.

6) The resolution is incremented by one and the steps 1 – 5 are repeated until either $\rho_{\text{final}} > \rho$ or until as many clusters were detected as specified as **MinClusters**. To assure that the steps are not repeated forever, the loop is stopped if the resolution reaches the total number of points in the dataset. The resolution reached at the end of this step is the final resolution.

7) All data points are assigned according to the found clusters. Points in cluster "0" are hereby not part of a cluster.



(a) Resulting data point assignment.

(b) Final count map (resolution 13).

Figure 3: Result of ITERIDENSE for the data shown in Fig. 2 (a) for $\rho = 4.1$.

8) Due to the grid generated in step 1, single points might appear in the corner of a cell and are thus not detected as part of a cluster. Therefore steps 1 – 5 are repeated with the final resolution plus 1.
9) Points that were before step 8 not part of a cluster but now are, are finally assigned to that cluster. This will only be done if the number of clusters did not change in step 8 and if no data point belongs now to another cluster than before step 8.

Fig. 3 shows the result for $\rho = 4.1$. In that case the final resolution is 13 and there is one cluster with $\rho_{\text{cluster}} = 4.3$. Note that this case is just an example for the algorithm. A suitable clustering result for this dataset would probably be one with 2 clusters. We will discuss later what "suitable" means.

3.2 Optional Settings

The algorithm can optionally be modified this way:

1) In step 3 don't take cells into account that are diagonally connected to the current cell (**NoDiagonals**). In Fig. 2 (b) cells A and C would then not be evaluated.
2) Specification of the start resolution for step 1 (**StartResolution**). The default and minimum is 2, the maximum is the total number of points in the dataset.
3) Specification of a resolution at which the loop step 1 – 5 is stopped (**StopResolution**). It is by default set to $\min(64, N)$, whereas N is the number of data points.
4) Specification of the minimal number of data points in a cluster (**MinClusterSize**). Clusters with less points will be erased. The default is 3, the minimum is 2, the maximum is the total number of points in the dataset minus 1.
5) Specification of the minimal ρ_{cluster} of a cluster (**MinClusterDensity**). Clusters with lower ρ_{cluster} will be erased. The minimum and default is 1.0.

Option 1 can be useful for a low ρ (and thus low resolutions)[1] while for higher ρ and also high dimensions it might lead to bad results. Therefore this option should only be used if really desired.

1. For example with **NoDiagonals** in Fig. 3 with $\rho = 2.2$ the cluster would contain more points.

Option 2 can speed up the computation, see Sec. 5 for details.

Option 3 prevents undesired many loops. For example if ρ was set to a high value and no cluster will be found.

Option 4 is useful to exclude unsuitably small clusters. It is recommended to set **MinClusterSize** $\geq D+1$ where D is the dimensionality of the dataset.

Option 5 has only an effect if **MinClusters** is used. It helps to sort out clusters with a density too low to be sensible for the use case.

A reference implementation of ITERIDENSE in the programming language JULIA is online available [7]. Its outputs are the assignments of the points to clusters, the size of clusters, density of clusters, final resolution, number of clusters, the count map as tensor in the final resolution (the actual probability-density function) and a tensor like the count map but with information about what cluster a grid cell belongs to. There is also a stand-alone program available with a graphical user interface (GUI) that uses the reference implementation [8].

3.3 Density Normalization

To be able to use the cluster density in a useful way, it has to be normalized. The point is that without it, the density will increase with the increase of the resolution. We define ρ_{cluster} as:

$$\rho_{\text{cluster}} = \frac{P}{C} \tag{4}$$

whereas C is the number of cells of the cluster and P the number of points in the cluster.

When the resolution doubles, ρ_{cluster} will increase because the cluster will keep its number of points but will have more cells. The ITERIDENSE algorithm requires ρ_{cluster} to be independent of the resolution. One approach would be to normalize ρ_{cluster} with the density of the whole dataset

$$\rho_{\text{cluster norm}} = \frac{\rho_{\text{cluster}}}{\rho_{\text{dataset}}} \tag{5}$$

We have

$$\rho_{\text{dataset}} = \frac{N}{R^D} \tag{6}$$

whereas N is total number of points, R the resolution and D the dimension.

This density is independent of the resolution but not on the dimension. In practice it is often a tricky question if more or less dimensions are applicable for the effect one wants to evaluate. By taking another dimension into account, the density would increase, even if the new dimension does not influence the cluster at all.

Take for example this case: dimension $D = 1$, resolution $R = 8$, $C = R/4$ (C depends on R), $P = 8$, $N = 16$.
We get $\rho_{\text{cluster}} = \frac{P}{C} = 4$, $\rho_{\text{dataset 1D}} = \frac{N}{R} = 2$, $\rho_{\text{cluster norm 1D}} = \frac{PR}{CN} = 2$, so the cluster is 2 times more dense than the dataset.

Now another dimension is added. Here we have to make assumptions how the number of data points change by this addition:

- We assume that for the 1D case we have 2 equal clusters that contain together all data points, so $N = 2P$.

- We assume that every dimension adds 2 more of these clusters (like a hypercube gets 2 more facets with every dimension).

With this we have for 2D:

$$\rho_{\text{cluster}} = \frac{P}{C},\ \rho_{\text{dataset 2D}} = \frac{4P}{R^2},\ \rho_{\text{cluster norm 2D}} = \frac{P \cdot R^2}{C \cdot 4P} = 8,$$

so the cluster is 8 times more dense than the dataset and $\rho_{\text{cluster norm}}$ is 4 times the one for the 1D case. This is a problem because the idea of ITERIDENSE is that you can increase ρ to find clusters. The addition of a dimension would lead to the fact that one has to increase ρ drastically to find the same cluster. This makes the usage of ρ hard.

The solution is to preserve $\rho_{\text{cluster norm}}$ of the 1D case to all greater dimensions using the assumptions we made for adding a dimension. That means we have to multiply $\rho_{\text{cluster norm}}$ with a correction factor γ that depends on the dimension. In our example γ would be 1/4.

ρ_{cluster} is independent of D, therefore γ only depends on ρ_{dataset}. With our assumptions we have

$$\rho_{\text{dataset D}} = \frac{2D \cdot P}{R^D} \tag{7}$$

and

$$\gamma_{1D} = \frac{\rho_{\text{dataset D}}}{\rho_{\text{dataset 1D}}} = \frac{R \cdot 2D \cdot P}{2 \cdot PR^D} = \frac{D}{R^{D-1}} \tag{8}$$

and

$$\rho_{\text{cluster norm 1D}} = \rho_{\text{cluster norm}} \cdot \gamma_{1D} = \frac{P}{C} \cdot \frac{R^D}{2D \cdot P} \cdot \frac{D}{R^{D-1}} = \frac{R}{2C} \tag{9}$$

Since for $D = 1$ $C \propto R$, $\rho_{\text{cluster norm}}$ is independent of the dimension and the resolution.

This normalization will preserve the density from the 1D case. However, for most practical use cases, one starts with 2D. Therefore we normalize according to the 2D case:

$$\gamma_{2D} = \frac{\rho_{\text{dataset D}}}{\rho_{\text{dataset 2D}}} = \frac{R^2 \cdot 2D \cdot P}{4 \cdot PR^D} = \frac{D}{2R^{D-2}} \tag{10}$$

The final normalization is for $D > 2$:

$$\rho_{\text{cluster norm}} = \frac{\rho_{\text{cluster}}}{\rho_{\text{dataset}}} \cdot \gamma_{2D} = \frac{P}{C} \cdot \frac{R^D}{N} \cdot \frac{D}{2R^{D-2}} \tag{11}$$

For $D < 3$ we omit the factor γ_{2D}:

$$\rho_{\text{cluster norm}} = \frac{\rho_{\text{cluster}}}{\rho_{\text{dataset}}} \tag{12}$$

This normalization cannot cover all cases but keeps $\rho_{\text{cluster norm}}$ stable enough for practical usage. To get a feeling about the stability we take the case $D = 2$, $R = 8$ and clusters with each $P = 8$ and $C = 2$, thus $\rho_{\text{cluster}} = 4$. We look at these 4 cases:

- 2 clusters, one at the upper right corner, the other one of the lower left corner of the data grid.
  - Now a new dimension is added which only adds a single new cluster of the same size. Then we have $\rho_{\text{dataset 2D}} = \frac{2P}{R^2}$ and thus $\rho_{\text{cluster norm 2D}} = 16$. And for $D = 3$ we have $\rho_{\text{data set 3D}} = \frac{3P}{R^3}$ and thus $\rho_{\text{cluster norm 3D}} = 16$. So no change for $\rho_{\text{cluster norm}}$.
  - Now a new dimension is added which adds two new clusters of the same size. Then we have for $D = 3$ $\rho_{\text{dataset 3D}} = \frac{4P}{R^3}$ and thus $\rho_{\text{cluster norm 3D}} = 12$. So $\rho_{\text{cluster norm}}$ increased by a factor 0.75. Without γ_{2D} it would have increased by a factor 4.
- 4 clusters at every corner of the data grid.
  - Now a new dimension is added which only adds a single new cluster of the same size. Then we have $\rho_{\text{dataset 2D}} = \frac{4P}{R^2}$ and thus $\rho_{\text{cluster norm 2D}} = 8$. And for $D = 3$ we have $\rho_{\text{dataset 3D}} = \frac{5P}{R^3}$ and thus $\rho_{\text{cluster norm 3D}} = 9.6$. So an increase of $\rho_{\text{cluster norm}}$ by a factor of 1.2. Without γ_{2D} it would have increased by a factor 6.4.
  - Now a new dimension is added which adds two new clusters of the same size. Then we have for $D = 3$ $\rho_{\text{dataset 3D}} = \frac{6P}{R^3}$ and thus $\rho_{\text{cluster norm 3D}} = 8$. So no change for $\rho_{\text{cluster norm}}$.

4 Clustering Results

To show the clustering results artificial data was generated using the library *sklearn.datasets* from the scikit-learn project [9]. Real data were taken from the *Rdatasets* database [10]. The clustering results figures were created using the package *Plots* of the Julia programming language [11] and the GUI reference implementation for Iteridense that uses the component *TAChart* of the Lazarus Component Library [12].

4.1 Effect of the Density Parameter

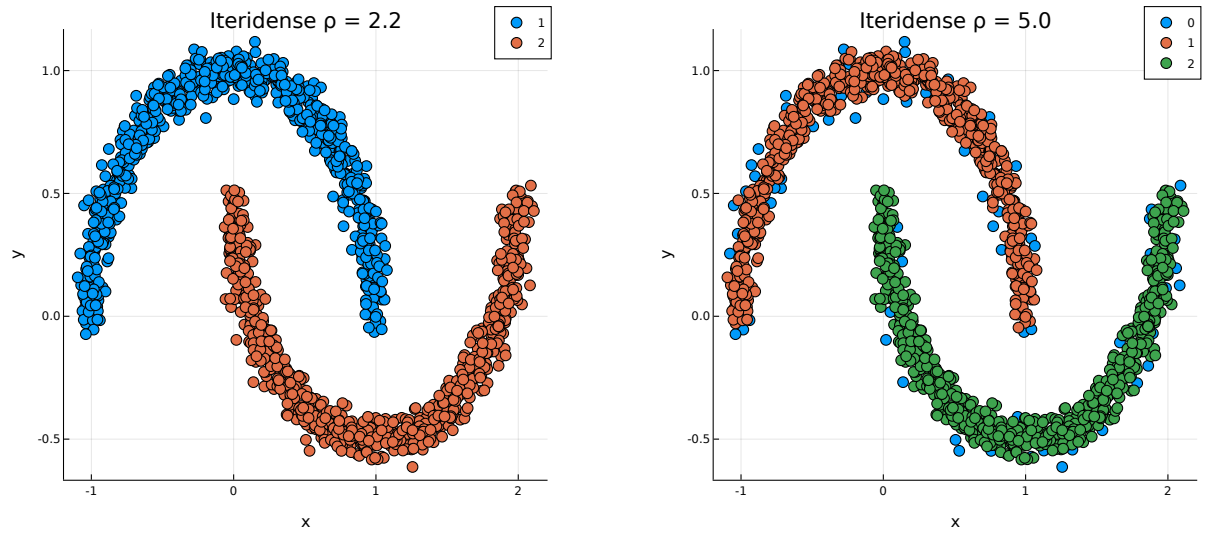
The data shown in Fig. 4 and Fig. 5 was generated using the *make_moons* call to *sklearn.datasets*. Fig. 4 (a) shows the result for $\rho = 2.2$, Fig. 4 (b) shows the result for $\rho = 5.0$. As derived in Sec. 2, the greater the density, means (translated to the relief example) the area of the cluster gets smaller. Therefore less points are assigned to the clusters for $\rho = 5.0$.

By increasing ρ one can for example define points as outliers: One could define that all data points that are not part of a cluster at density $\rho = 4.0$ are outliers. If the data points are the result of a measurement, one can repeat the measurement of an outlier point to verify if the result is still an outlier, check the measurement setup etc.

Fig. 5 shows the result for $\rho = 6.0$. The density is now so high that the moon-like clusters break down into many small clusters if the option **NoDiagonals** is used for the clustering.

4.2 Clustering Performance

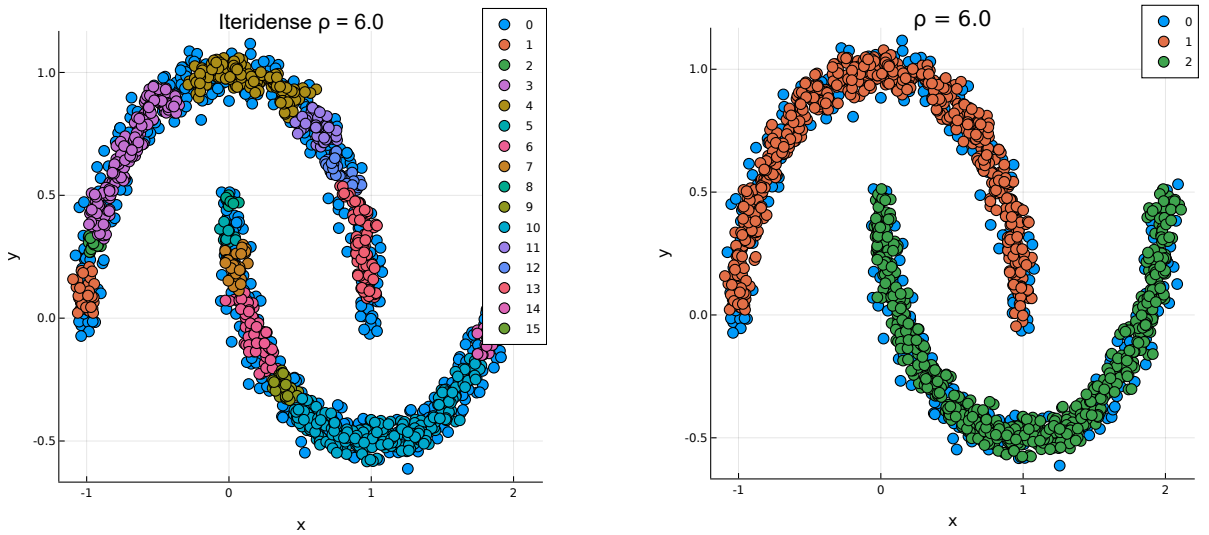
An advantage compared to other clustering algorithm is that Iteridense treats all data points the same way. There is no separation between core points, border points or the like. Another major feature of Iteridense is that it provides two ways to achieve results and a clear path for the user on how to change the input parameters to get a suitable result:



(a) $\rho = 2.2$, final resolution: 12

(b) $\rho = 5.0$, final resolution: 43

Figure 4: Result of ITERIDENSE for intersected moon-like clusters using different ρ.



(a) $\rho = 6.0$, **MinClusterSize** = 10 with option **NoDiagonals**, final resolution: 60

(b) $\rho = 6.0$, final resolution: 52

Figure 5: Effect of a high ρ and the option to evaluate neighbor cells.

- Either start with a low ρ and increase it gradually to get a suitable result. If the result is not suitable, look at the resulting ρ_{cluster} and set for the next run ρ above their minimum.
- Or specify with **MinClusters** the desired number of clusters. If the result is not suitable, increase gradually either **MinClusterSize** or **MinClusterDensity**.

The second path is computationally the fastest, as discussed in the next section. However, it can only be taken if there is a physical or technical reason for the number of clusters. For the path to specify ρ Fig. 6 (a) – Fig. 7 (a) shows the results (with **MinClusterSize** = 6): Until $\rho \leq 6.8$ only one cluster is detected and starting at $\rho = 7.3$ there are 3 clusters. Increasing ρ leads to more and more clusters. This can be used to identify regions with higher density inside a "base" cluster. In the example there are 3 base clusters

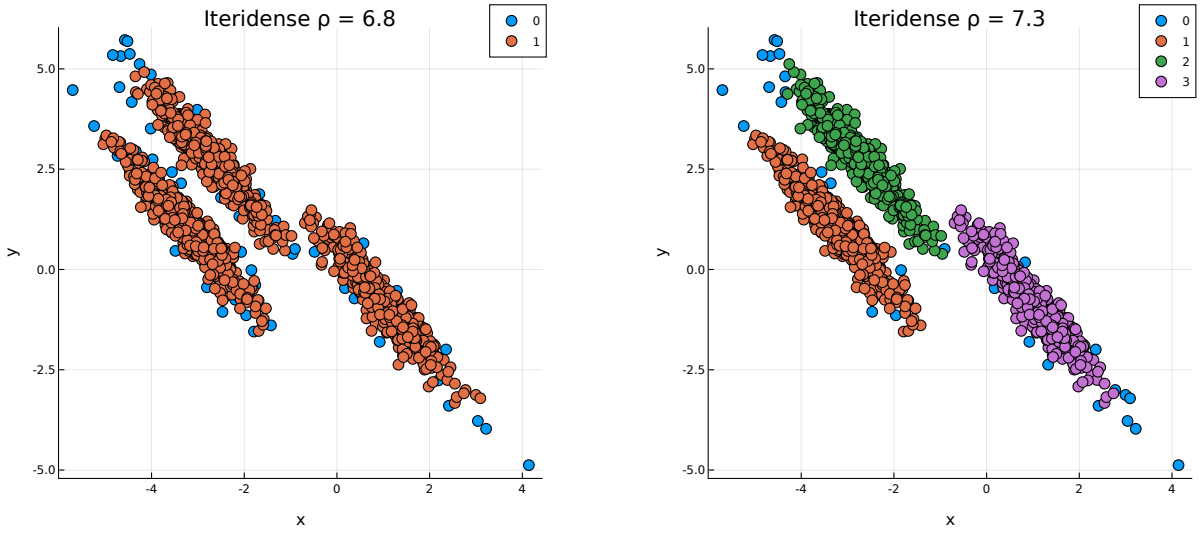
and every one has more dense regions that are unveiled with greater ρ.

For comparison, the algorithm DBSCAN does not provide a clear path on how to change its input parameters. An example is the data shown in Fig. 7 (b). This data was generated using the *make_blobs* call to *sklearn.datasets* with a subsequent transformation. Like in the previous example, there are 1500 data points. $\epsilon = 0.1$ (the maximum distance to another core point of a cluster) seems to be a sensible start value for DBSCAN. The result is Fig. 8 (b). As the result is not a useful one might increase ϵ in small steps and gets with **MinPts** (minimal points to form a dense region) of 6 as results Fig. 8 – Fig. 9. For this dataset a human would expect 3 base clusters but DBSCAN does not find exactly 3 clusters. One has to try different ϵ to get this result. For data in 2 or 3 dimensions one can plot the data to get a feeling for ϵ and for example increase **MinPts**. However, for data in higher dimensions this is hardly possible.

What is "suitable" depends on the application. For example the data shown in Fig. 10 consist also of 1500 data points. It was generated using the *make_blobs* call to *sklearn.datasets* with a subsequent transformation. One might see in the data 2 clusters and therefore set **MinClusters** to 2 and **MinClusterSize** to 20. As result one gets Fig. 10 (a). There cluster 1 has $\rho_{\text{cluster}} = 2.6$ and is a merge of a high- and a low-density region. This might not be suitable because one had the 2 dense regions in mind to form each a cluster. There are now two ways to change the result:

- Either set **MinClusters** to 3 to get two clusters with a high density and one with a lower density, see Fig. 10 (b).
- Or increase ρ and also increase **MinClusterSize** e.g. to 50 unless one gets only 2 high-density clusters, see Fig. 10 (c). Increasing **MinClusterSize** is hereby necessary to avoid small clusters in the low-density area.

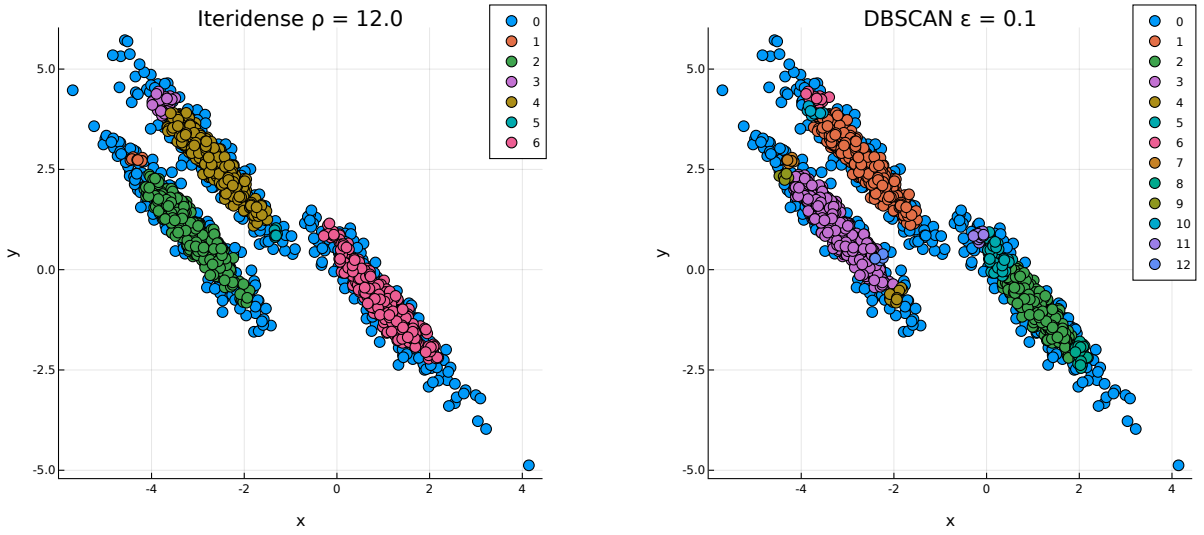
The change of **MinClusterSize** might not be obvious when one cannot plot for example high-dimensional data. It is therefore a useful feature of the ITERIDENSE algorithm that for the case ρ_{cluster} is too low for a suitable result, one can increase **MinClusterSize** together with ρ.



(a) $\rho = 6.8$, final resolution: 30

(b) $\rho = 7.3$, final resolution: 35

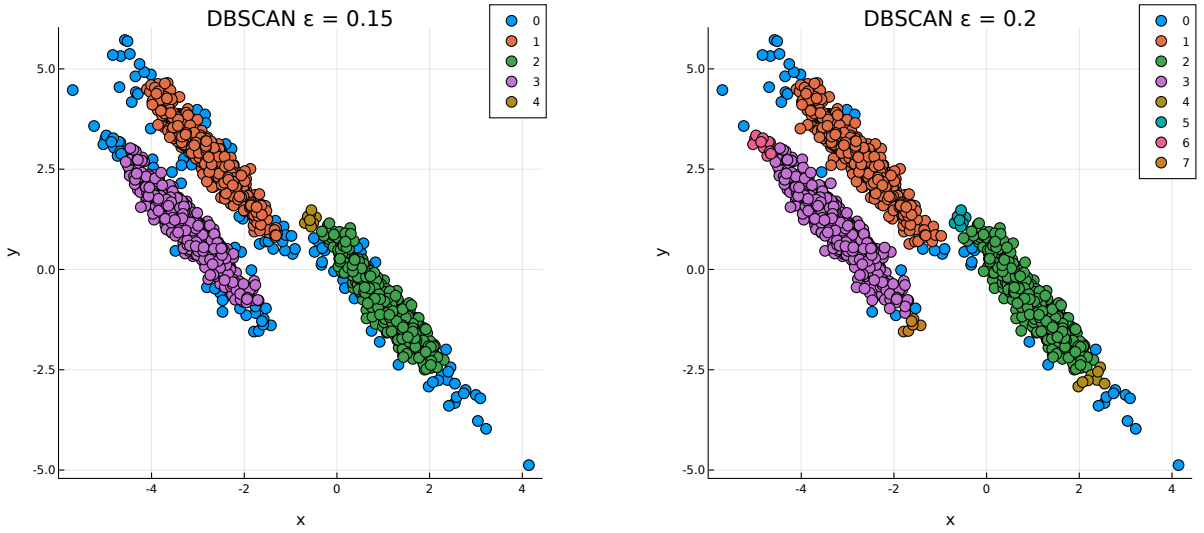
Figure 6: Result of ITERIDENSE for $\rho \geq 6.8$ at anisotropic clusters.



(a) $\rho = 12.0$, final resolution: 68

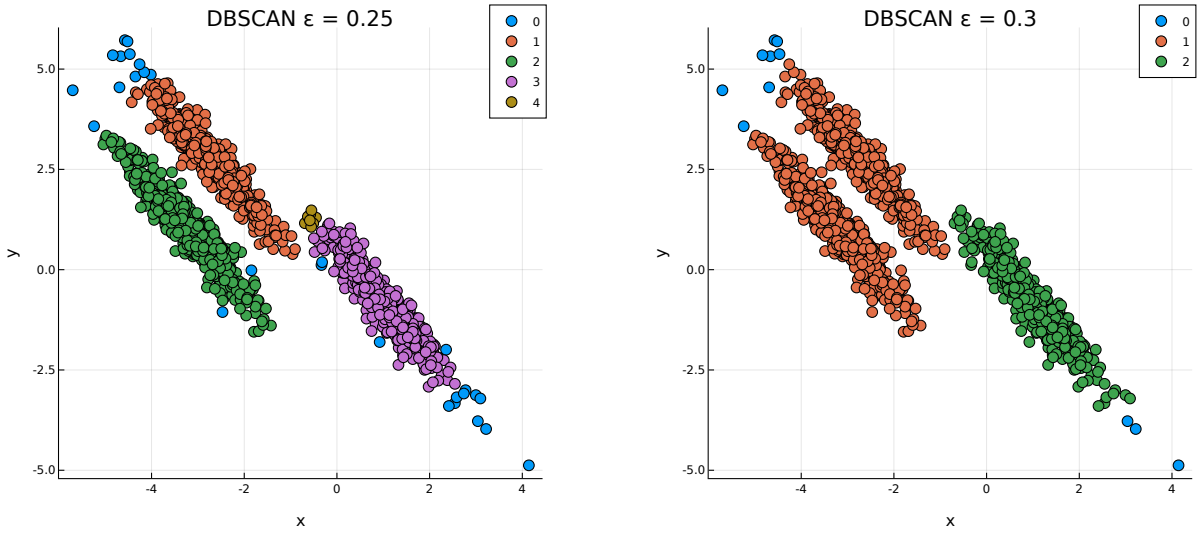
(b) $\epsilon = 0.1$

Figure 7: Result of ITERIDENSE for $\rho = 12.0$ and DBSCAN for $\epsilon = 0.1$ at anisotropic clusters.



(a) $\epsilon = 0.15$ **(b)** $\epsilon = 0.2$

Figure 8: Result of DBSCAN at anisotropic clusters with $\epsilon \leq 0.2$.



(a) $\epsilon = 0.25$ **(b)** $\epsilon = 0.3$

Figure 9: Result of DBSCAN at anisotropic clusters with $\epsilon > 0.2$.

(a) MinClusters = 2 **(b) MinClusters** = 3 **(c)** $\rho = 4.0$, **MinClusterSize** = 50

Figure 10: Result of Iteridense for clusters with different densities.

5 Computational and Memory Complexity

5.1 Computational Complexity

Iteridense creates two tensors, the count tensor to store how many data points are in a cell and the cluster tensor that stores to what cluster a cell belongs to. Both tensors have the rank D whereas D are the dimensions of the data (or features). Each tensor has R^D entries (cells) whereas R denote the resolution. Therefore Iteridense is memory-limited. This is an important point as this limits the practical usability.

To estimate the computational complexity $\mathcal{O}$ we look at the different steps:

Counting Because a computation is performed for every data point in every dimension we have $\mathcal{O}(DN)$.

Clustering Every cell has B neighbor cells: $B = 3^D - 1$ (for the case that option **NoDiagonals** is used $B = 2D$). For the clustering we only evaluate $B/2$ of the neighbors. Therefore checking a neighbors has $\mathcal{O}\left(0.5BR^D\right) \approx \mathcal{O}\left(0.5\left(3R\right)^D\right)$.

Evaluation To evaluate how many clusters were found and how many cells they have, all tensor cells are evaluated, therefore this step has $\mathcal{O}\left(R^D\right)$.

Assignment To assign the cluster number to every data point the complexity is $\mathcal{O}(DN)$.

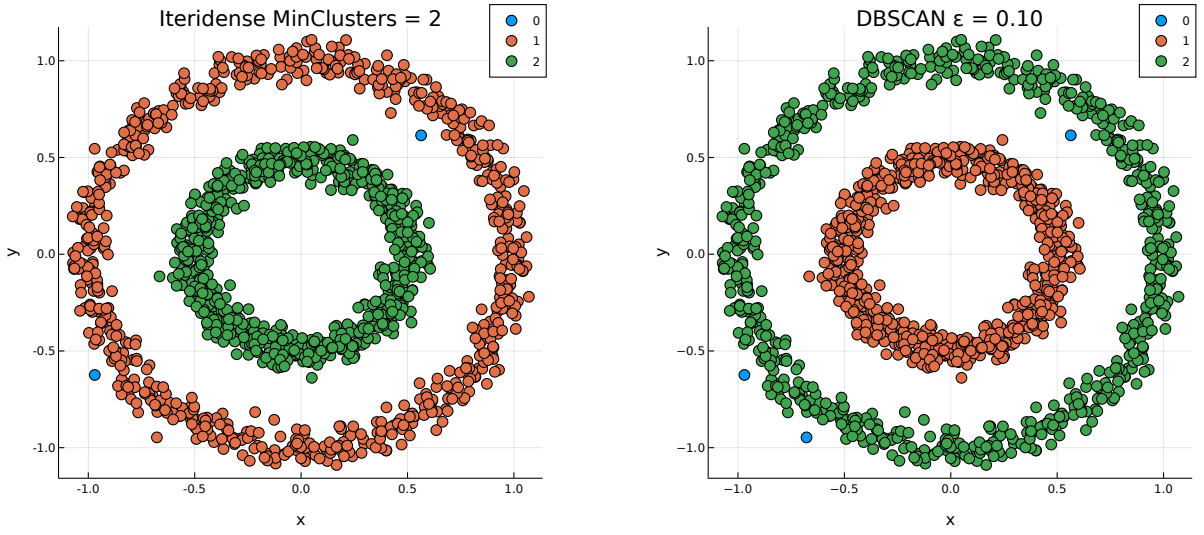
Iteridense clusters iteratively with increasing $R = R_{\text{start}} \dots R_{\text{final}} + 1$. Note that the assignment is only performed once for the final R. In total these are the number of computes:

$$DN + \sum_{R=R_{\text{start}}}^{R_{\text{final}}+1} \left(DN + R^D \left(\frac{3^D}{2} + 1 \right) \right) \tag{13}$$

$$\approx DN + \left(R_{\text{final}} + 1 - R_{\text{start}} + 1\right) DN + \frac{3^D}{2} \sum_{R=R_{\text{start}}}^{R_{\text{final}}+1} R^D \tag{14}$$

as approximation we can write

$$\sum_{R=R_{\text{start}}}^{R_{\text{final}}+1} R^D \approx \frac{\left(R_{\text{final}} + 1\right)^{D+1} - R_{\text{start}}^{D+1}}{D+1}$$



(a) MinClusters = 2, final resolution: 16 **(b)** $\epsilon = 0.1$

Figure 11: Result of ITERIDENSE and DBSCAN for intersected circle-like clusters.

For $\mathcal{O}$ we get

$$\mathcal{O}\left(\left(R_{\text{final}} - R_{\text{start}} + 3\right) DN + \frac{3^D}{2(D+1)}\left(\left(R_{\text{final}}+1\right)^{D+1} - R_{\text{start}}^{D+1}\right)\right) \tag{15}$$

The worst case is $R_{\text{start}} = 2$ as this is the lowest possible resolution. This leads to

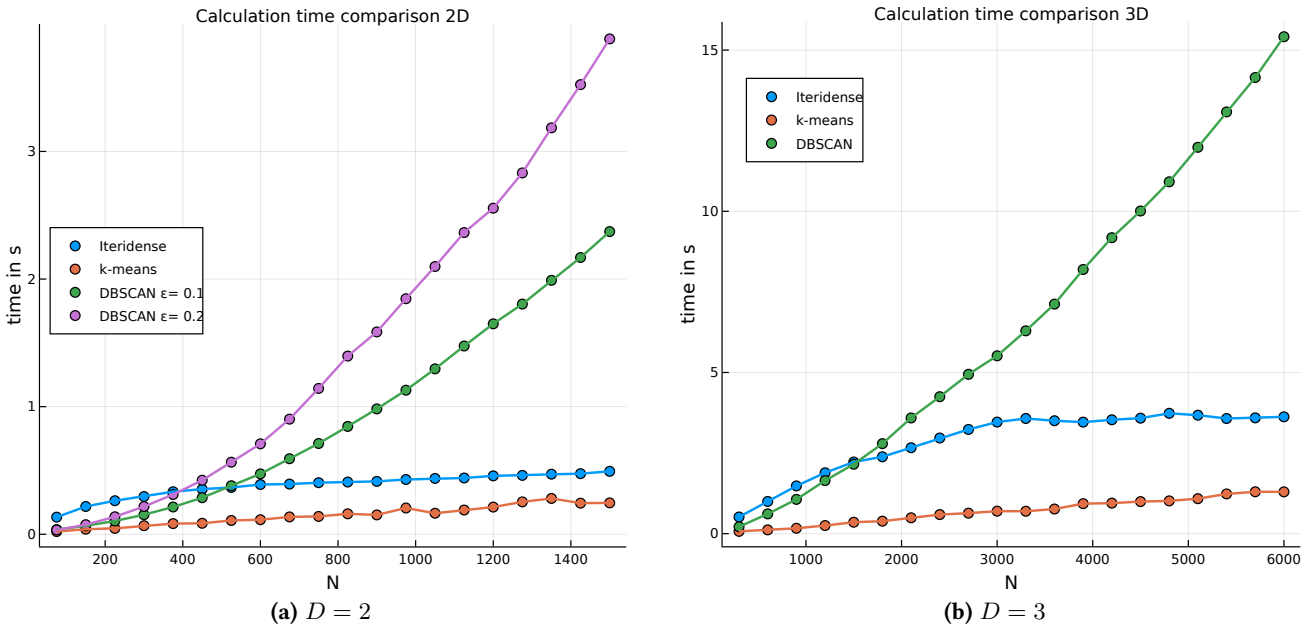
$$\mathcal{O}_{\text{worst case}} \approx \mathcal{O}\left(R_{\text{final}} DN + \frac{3^D}{2(D+1)}\left(R_{\text{final}}+1\right)^{D+1}\right) \tag{16}$$

With this result the following can be seen:

- ITERIDENSE scales roughly with $\mathcal{O}\left(DN + R^{D+1}\right)$ – a linear increase plus an offset.
- Setting R_{start} (the parameter **StartResolution**) has an impact, but a minor one.
- The worst case for ITERIDENSE is that ρ is set so high that $R_{\text{final}} = N$. To prevent that this happens, the parameter **StopResolution** sets the limit for R_{final}.

To prove that our derivation of $\mathcal{O}$ is correct, we took a dataset with $N = 1500$ and clustered subsets of that dataset. Every subset has a different N, which allows to create a plot with the computation time over N. The used dataset is shown in Fig. 11. It was generated using the *make_circles* call to *sklearn.datasets*.

Fig. 11 (a) shows the ITERIDENSE result for the full dataset for **MinClusters** $= 2$. Thereby $R_{\text{final}} = 16$ and this value was used for **StopResolution** when clustering the subsets. By setting **MinClusters** $= 20$ it was assured that for every clustering $R_{\text{final}} = 16$ was reached.
For comparison, Fig. 11 (b) shows the result using the DBSCAN algorithm for $\epsilon = 0.1$ and **MinPts** $= 3$. We chose for the subset clustering also $\epsilon = 0.2$ to see the influence of ϵ on the computation time.
The computation of the k-mean and DBSCAN algorithms were performed using their implementation in the package *Clustering* of the JULIA programming language, [13]. The times were measured using the feature *@elapsed* of the JULIA programming language. To get reliable values, the clustering was for every algorithm performed 2000 times in a row and the overall time was taken.



(a) $D = 2$ **(b)** $D = 3$

Figure 12: Computation time comparisons.

The result is shown in Fig. 12 (a). Iteridense shows the same linear increase as k-means proving that formula (16) is basically correct. The slope of the increase is relatively low. For DBSCAN in the used implementation, one can see that is has $\mathcal{O}\left(DN^2\right)$ and how the choice of ϵ influences the computation time.

The N at which Iteridense is faster than DBSCAN is at about $N = 520$. By repeating the test in 3D, there should be an increase of this value because of the greater offset in $\mathcal{O}$ for Iteridense. This test was performed by adding z-values to the dataset so that the clusters form cylinders. DBSCAN finds the 2 clusters for the range $0.2 \leq \epsilon \leq 0.26$ and we chose the lowest possible $\epsilon = 0.2$. Eventually, we extended the dataset by multiplying its points to get in total $N = 6000$ points. The result is shown in Fig. 12 (b). The N at which Iteridense is faster than DBSCAN is now at about $N = 1500$.

Fig. 13 shows $\mathcal{O}$ of Iteridense in comparison to $\mathcal{O}\left(DN^2\right)$ for $R_{\text{final}} = 10$. (R_{final} around 10 is a common result when clustering in more than 3 dimensions.) The $\mathcal{O}$ of Iteridense is in all areas lower than $\mathcal{O}\left(DN^2\right)$ except of low N. Therefore Iteridense extends the parameter range at which clustering could be performed, dramatically, compared to algorithms with $\mathcal{O}\left(DN^2\right)$.

Fig. 14 shows the cluster result for the full 3D dataset for **MinClusters** $= 2$. Iteridense does not find the outer cylinder to be completely a cluster. DBSCAN can find it because it allows to change the search radius. However, it took several cluster attempts to find an ϵ that works. In 3D the dataset can be visualized to find a suitable ϵ but this is impossible in higher dimensions. In effect DBSCAN provides for this task more flexibility and a more suitable result for the cost that one needs to try. In comparison, for Iteridense one can set **MinClusters** $= 2$ and then fine-tune with ρ. This dataset is a good example that every algorithm has its strengths and weaknesses.

5.2 Memory Complexity

The main limitation of Iteridense is its basic memory complexity of $\mathcal{O}\left(R^D\right)$. Therefore on consumer hardware the practical limit would be with $R = 10$ around $D \approx 11$.[2] However, for many datasets several

2. Assuming Iteridense uses the data type Float32 (size 4 byte) for the tensors.

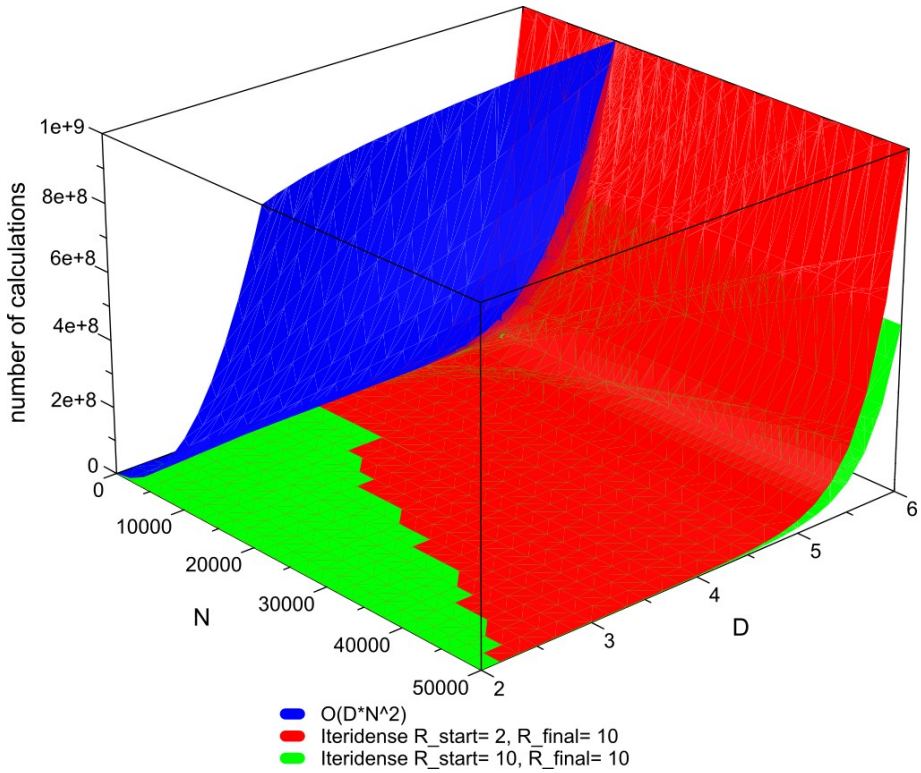
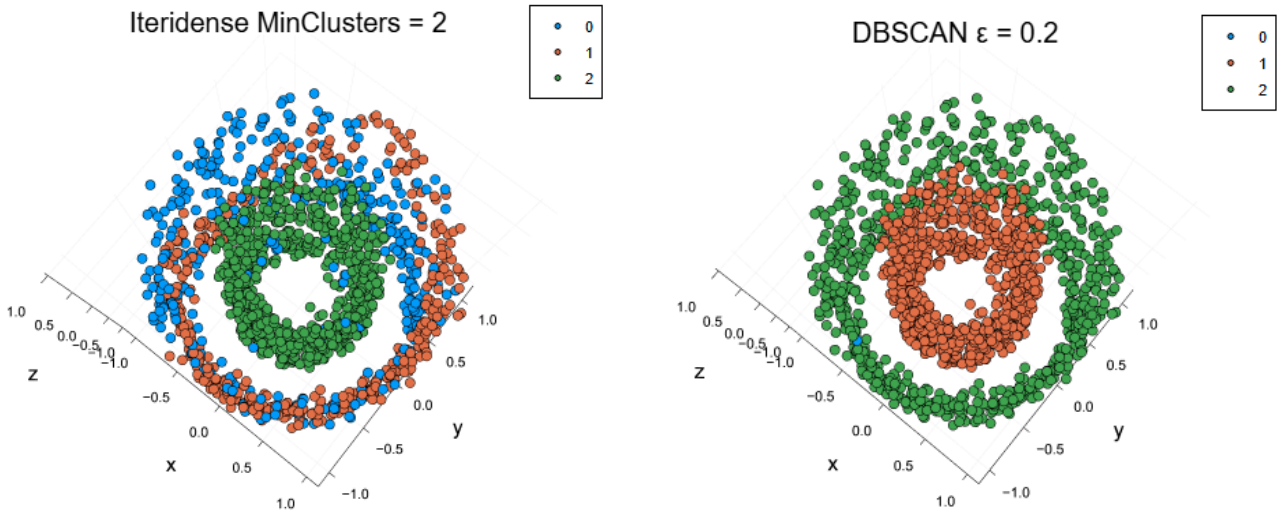


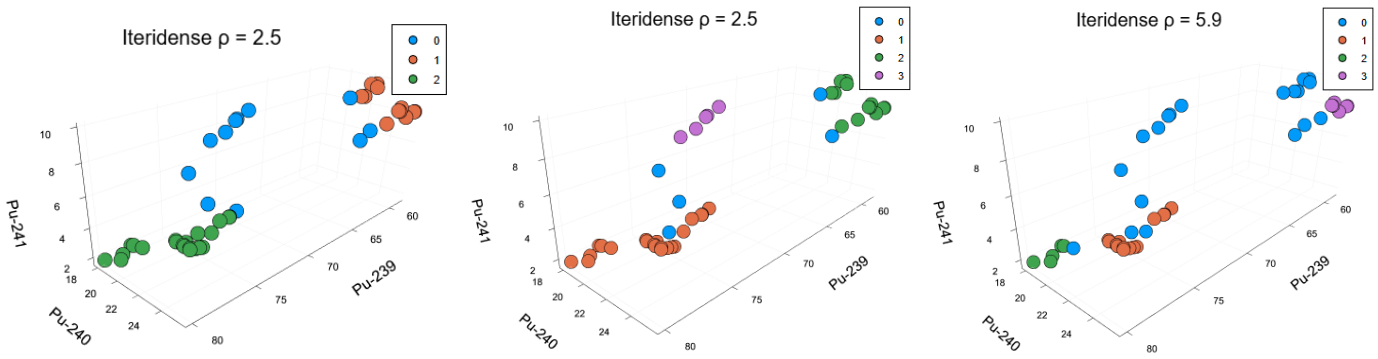
Figure 13: Computational complexity of ITERIDENSE in comparison to $\mathcal{O}\left(DN^2\right)$.



(a) MinClusters = 2, final resolution: 13

(b) $\epsilon = 0.2$

Figure 14: Result of ITERIDENSE and DBSCAN for intersected cylinder-like clusters.



(a) Clustering with only the shown dimensions taken into account. final resolution: 6

(b) Clustering with all dimensions taken into account. final resolution: 5

(c) like (b) but $\rho = 5.9$, final resolution: 7

Figure 15: Result of ITERIDENSE for the *pluton* dataset.

dimensions are categorical data. Therefore it is not necessary to create the count tensor with R^D cells. For example if one dimension is the gender, there are only 3 cells necessary to store the information of this dimension, each one for the gender and one cell to separate both.

For this case the ITERIDENSE algorithm has the option **OmitEmptyCells**. If used, the tensor will have

$$R^{D-D_c} \cdot \bar{K}^{D_c} \tag{17}$$

cells, whereas D_c are the number of dimensions with categorical data and $\bar{K}$ is the mean number of categories in the categorical dimensions of the dataset.

This increases the applicability of Iteridense significantly. Because a dataset with e. g. $D_c = \frac{D}{2}$ and $\bar{K} = 3$ can be clustered on consumer hardware for $R = 10$ if it has up to $D = 14$.

The determination of the necessary number of cells per dimension has $\mathcal{O}\left(DN\log(N)\right)$, therefore the total $\mathcal{O}$ is greater than without the option **OmitEmptyCells**.

Table 1 lists $\mathcal{O}$ of different clustering algorithms.

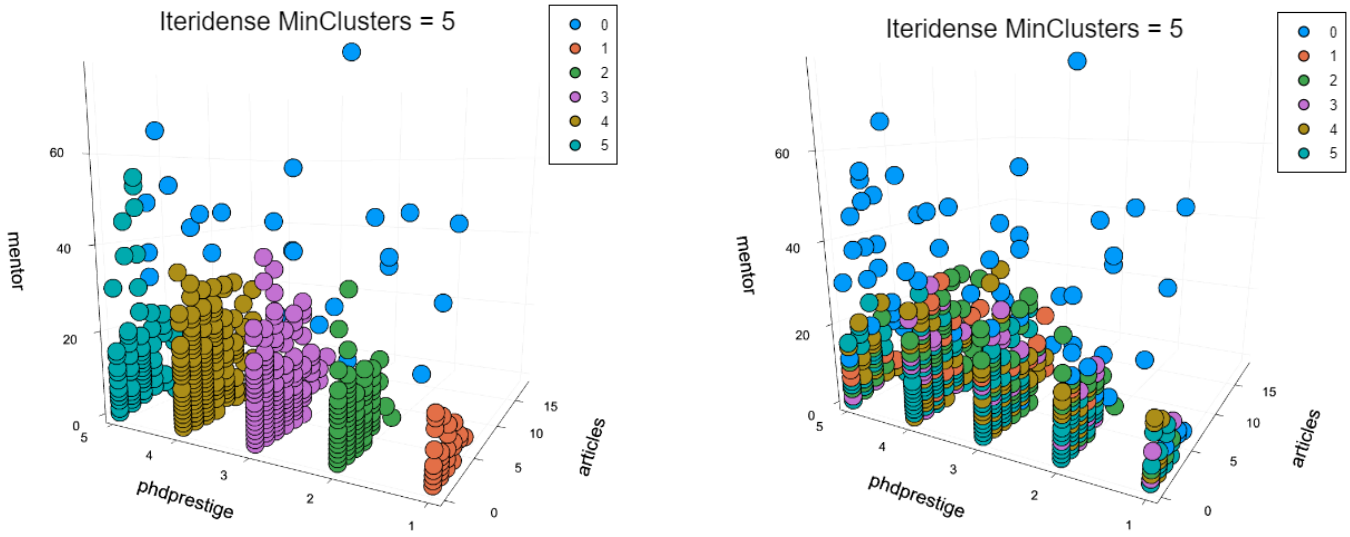
Table 1: Computational and memory complexity for different cluster algorithms.

Algorithm	Computational Complexity	note
ITERIDENSE	$\mathcal{O}\left(DN + R^{D+1}\right)$ to $\mathcal{O}\left(DN\left(1+\log(N)\right) + R^{D-D_c}\bar{K}^{D_c}\right)$	actually formula (15)
DBSCAN	$\mathcal{O}(DN\log N)$ to $\mathcal{O}\left(DN^2\right)$ [14]	$\mathcal{O}\left(DN^2\right)$ if brute force
DENCLUE	$\mathcal{O}(DN\log N)$ to $\mathcal{O}\left(DN^3\right)$ [15]	$\mathcal{O}\left(N^3\right)$ worst case for N iterations
k-means	$\mathcal{O}(DN)$ [16]	

5.3 Results in higher Dimensions

Data with higher dimensions are a main use case for clustering algorithms as no human could do the clustering according to plots. Fig. 15 is an example and also demonstrate the clustering performance of ITERIDENSE. The data are the publicly available *pluton* dataset[17] containing concentrations of the different Plutonium isotopes in 45 ore samples. It has 4 dimensions.

By plotting 3 dimensions of the dataset and performing ITERIDENSE only on the plotted dimensions, the result would be Fig. 15 (a). Taking all dimensions into account, the result is Fig. 15 (b). The points of high



(a) Evaluation of the 3 shown dimensions. final resolution: 9

(b) Evaluation of all 6 dimension. final resolution: 5

Figure 16: Result of Iteridense on the *PhDPublications* dataset.

Pu-241 concentrations are then identified as a cluster. Cluster 1 is so large because its density in all 4 dimensions matters. If one wants for some reason cluster 1 to be split into 2 clusters, one follows the Iteridense path and increases ρ above the lowest ρ_{cluster} and ends up with Fig. 15 (c).

Fig. 16 shows a use case in which Iteridense shows its strengths. The data is the publicly available *PhDPublications* dataset [18]. It has 6 dimensions and we selected 3 for the visualization. Since there are 5 classes in the set, **MinCluster**s was set to 5. By evaluating only the 3 shown dimensions, one gets Fig. 16 (a). This shows that Iteridense's grid-based approach combined with its density analysis makes it possible to deal directly with classes inside datasets. Pure density-based algorithms cannot directly cluster the dataset in the same way.

When evaluating all available dimensions the clusters run across the clusters, Fig. 16 (b). To find in this case clusters inside the classes, one has to extract the classes to different datasets and then run Iteridense on every class. This is requires more efforts but leads to sensible results when going the way to specify ρ.

6 Discussion

As shown, the Iteridense algorithm is applicable for general purposes. It is more computation-efficient than pure density-based algorithms that evaluate neighboring data points. But it shares with density-based algorithms the drawback that clusters overlapping each other cannot be detected. For example it will perform as poor as DBSCAN on Fisher's Iris dataset, [19].

Iteridense shows some similarities to the grid-based DENCLUE algorithm. However, the assignments of the data points to the clusters is different because DENCLUE assumes a Gaussian distribution function as shape of the density function. Another big difference to DENCLUE is that there is not only a single probability-density function created but iteratively several ones with increasing resolution. This increases the computation efforts for the benefit that one does not need to make assumptions about the density in the dataset. DENCLUE requires at least 2 input variables. ξ is similar to ρ in Iteridense. The parameter σ defines the width of the Gaussian that is used to assign the data points to the clusters. Its value has to be guessed and therefore introduces for

some practical applications a trial and error process. There are approaches to improve the initial setting of the DENCLUE parameters, see [15], but in general this will remain as a practical challenge.

Compared to the grid-based algorithm CLIQUE, Iteridense does not require to specify the size of the cell (CLIQUE's parameter ξ) since the cell width is iteratively approached. There is also no need to specify the number of data points in a cell to treat the cell as being part of a cell (CLIQUE's parameter τ). The Iteridense algorithm treats every cell with at least 2 data points as part of a cluster. This is possible because at the end of every loop (steps 1 – 5) the clusters are evaluated and if their ρ_{cluster} is too low, the cluster is deleted. This is an advantage to CLIQUE because especially for high-dimensional data it is hard to estimate how many data points might end up in a cell.

Iteridense is a simple algorithm: The user does not need to estimate in advance a cell size, how many data points will be in a cell, the mean distance between data points in a cluster or the like. One can either specify **MinClusters** or ρ , sets **MinClusterSize** and gets in many cases directly a suitable result. If necessary, Iteridense's clear path on how to change the input parameters guides the user to more suitable or different results.

7 Conclusions

This paper introduced Iteridense, an iterative clustering algorithm combining grid-based and density-based methods. As result no point in a dataset can be in more than one cluster. In contrary to k-Means and Gaussian mixture clustering, points can also remain as not being part of any cluster.

Iteridense is simple to use because it does not require to estimate settings in advance and because it provides two ways to run the clustering. For both ways there is a clear path on how to change the algorithm's input parameters to achieve suitable results. Iteridense provides different options to affect the clustering result.

The Iteridense algorithm has a $\mathcal{O}\left(DN + R^{D+1}\right)$ and therefore extends the application range compared to algorithms having $\mathcal{O}\left(DN^2\right)$. Compared to pure grid-based algorithms it has the advantage that the user does not have to make assumptions on how the grid should be defined or about the shape of the probability-distribution.

It was demonstrated that for common example datasets Iteridense performs clustering as good to the DBSCAN algorithm with less computational effort.

We provide online a reference implementation together with an example worksheet, [7]. We also provide a stand-alone program with a graphical user interface that can be used to cluster any data that is available as a CSV file, [8].

References

[1] Hans-Peter Kriegel et al. "Density-based clustering". In: *WIREs Data Mining and Knowledge Discovery* 1.3 (Apr. 2011), pp. 231–240. ISSN: 1942-4795. DOI: 10.1002/widm.30.

[2] Christian Böhm et al. "Density Connected Clustering with Local Subspace Preferences". In: *Fourth IEEE International Conference on Data Mining (ICDM'04)*. IEEE, pp. 27–34. DOI: 10.1109/icdm.2004.10087.

[3] Ricardo Campello, Davoud Moulavi, and Jörg Sander. "Density-Based Clustering Based on Hierarchical Density Estimates". In: *Advances in Knowledge Discovery and Data Mining*. Springer Berlin Heidelberg, 2013, pp. 160–172. ISBN: 9783642374562. DOI: 10.1007/978-3-642-37456-2_14.

[4] Alexander Hinneburg and Daniel Keim. “An efficient approach to clustering in large multimedia databases with noise”. In: *KDD’98: Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*. Vol. 98. Bibliothek der Universität Konstanz Konstanz, Germany, 1998, pp. 58–65. URL: https://cdn.aaai.org/KDD/1998/KDD98-009.pdf.

[5] Rakesh Agrawal et al. “Automatic subspace clustering of high dimensional data for data mining applications”. In: *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. SIGMOD/PODS98. ACM, June 1998. DOI: 10.1145/276304.276314.

[6] Martin Ester et al. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *Second International Conference on Knowledge Discovery and Data Mining (KDD’96). Proceedings of a conference held August 2-4*. Ed. by D. W. Pfitzner and J. K. Salmon. Jan. 1996, pp. 226–331. URL: https://www.dbs.ifi.lmu.de/Publikationen/Papers/KDD-96.final.frame.pdf.

[7] Uwe Stöhr. *Iteridense Clustering*. Internet. Aug. 2025. URL: https://codeberg.org/Soloof/Iteridense.

[8] Uwe Stöhr. *Iteridense-package*. Internet. Aug. 2025. URL: https://github.com/donovaly/Iteridense-package.

[9] scikit-learn Team. *Utilities to load popular datasets and artificial data generators*. Internet. scikit-learn.org. URL: https://scikit-learn.org/stable/api/sklearn.datasets.html.

[10] Vincent Arel-Bundock. *Rdatasets: A repository of datasets available in R*. 2025. URL: https://vincentarelbundock.github.io/Rdatasets/articles/data.html.

[11] The JuliaPlots Organization. *Plots - powerful convenience for visualization in Julia*. Internet. URL: https://juliaplots.org/.

[12] Lazarus Team. *TAChart component*. Internet. URL: https://en.wikipedia.org/wiki/TAChart.

[13] Clustering.jl Team. *DBSCAN*. Internet. URL: https://juliastats.org/Clustering.jl/stable/dbscan.html.

[14] Youguang Chen, William Ruys, and George Biros. “KNN-DBSCAN: a DBSCAN in high dimensions”. In: *ACM Transactions on Parallel Computing* 12.1 (Feb. 2025), pp. 1–27. ISSN: 2329-4957. DOI: 10.1145/3701624.

[15] Omer Ajmal et al. “Enhanced Parameter Estimation of DENsity CLUstEring (DENCLUE) Using Differential Evolution”. In: *Mathematics* 12.17 (Sept. 2024), p. 2790. ISSN: 2227-7390. DOI: 10.3390/math12172790.

[16] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. “The k-means Algorithm: A Comprehensive Survey and Performance Evaluation”. In: *Electronics* 9.8 (Aug. 2020), p. 1295. ISSN: 2079-9292. DOI: 10.3390/electronics9081295.

[17] Rdatasets Team. *Isotopic Composition Plutonium Batches*. Ed. by Vincent Arel-Bundock. Internet. URL: https://vincentarelbundock.github.io/Rdatasets/doc/cluster/pluton.html.

[18] Rdatasets Team. *Publications of PhD Candidates*. Ed. by Vincent Arel-Bundock. Internet. URL: https://vincentarelbundock.github.io/Rdatasets/doc/vcdExtra/PhdPubs.html.

[19] R. A. Fisher. “The use of multiple measurements in taxonomic problems”. In: *Annals of Eugenics* 7.2 (Sept. 1936), pp. 179–188. ISSN: 2050-1439. DOI: 10.1111/j.1469-1809.1936.tb02137.x.