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

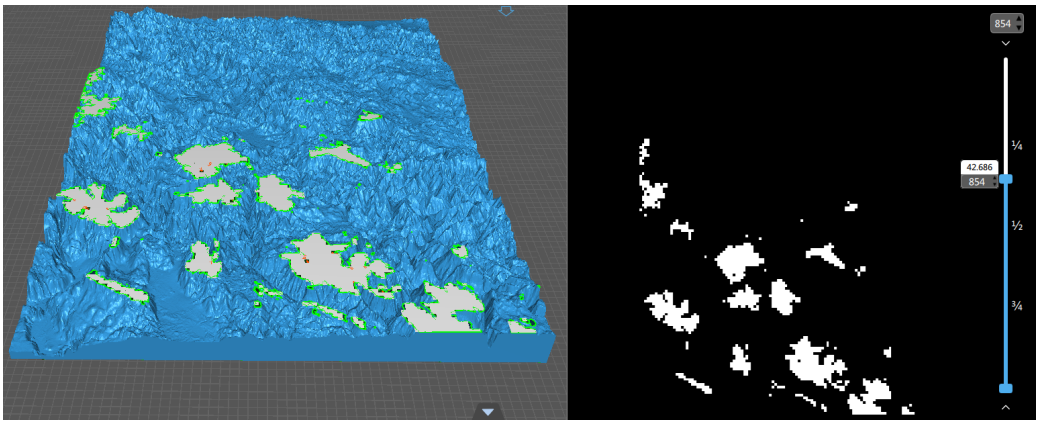
We introduce ITERIDENSE, an iterative clustering algorithm combining grid-based and density-based methods. It provides two possibilities to perform the clustering and for both it provides a clear path on how to change the algorithm's input parameters to achieve suitable results. We show that ITERIDENSE 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 to the DBSCAN algorithm. This work enables users to cluster data in a quick, reliable and objective way based on the the probability-density of the data.

1 Introduction

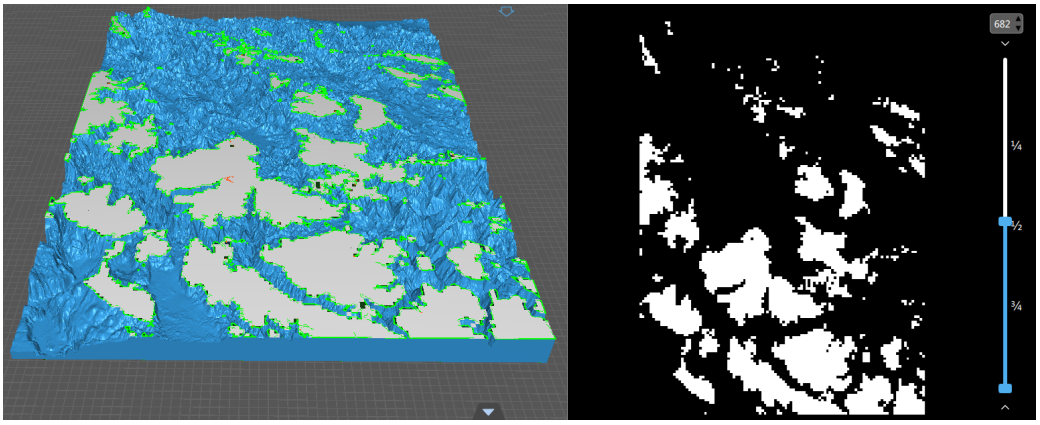
Density-based clustering algorithms have been proven useful for many practical application. There exists a wide variety of algorithms, some optimized for particular use cases [1]. However, these algorithms have a major drawback – one needs to evaluate neighboring points for every data point in the dataset 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.

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 base idea is to work with different probability-density functions of the dataset and analyzing them in a grid. Therefore 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.



(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 height to the density of data points ρ the cut is made along 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 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 there are 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 the choice to specify either 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 to stop the algorithm. When specifying **MinClusters** is possible, one gets a completely objective result (no human bias involved as specifying a 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 increasing ρ until one gets 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.

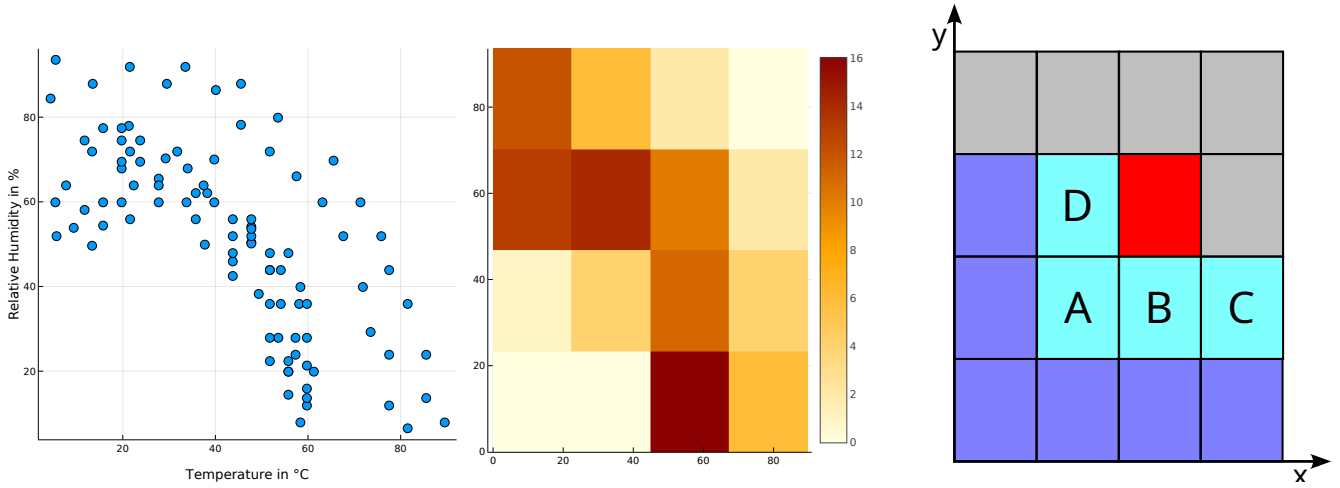
3 Description of the Iteridense Clustering Algorithm

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 to form a cluster ρ. 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 thermo-box 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 of dimension 1 "temperature" is $4-89\,°\mathrm{C}$ and the dimension 2 "humidity" is $6-93\,\%$. This range defines the space of the dataset. 4×4 cells is defined as a resolution of 4. 5×5 cells would be a resolution of 5 and so on.
2) The number of data points in the cell spaces ($[4-89/4\,°\mathrm{C},\ 6-93/4\,\%]$, $[89/4-2\cdot 89/4\,°\mathrm{C};\ 6-2\cdot 93/4\,\%]$, ...) are counted. The result is a count map as shown in Fig. 2 (a).
3) Now the actual clustering is performed. The processing scheme ist depicted in Fig. 2 (b). Every cell is processed one after another first in x- then in y-direction. If a cell has more than 1 data point it could be part of a cluster and then its neighboring cells are evaluated. Thereby only those neighbors are evaluated that have already be evaluated (blue and light-blue in Fig. 2 (b)) 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 of algorithm step 3: 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. To normalize the density, the result is divided by the density of the whole dataset:

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

So the whole dataset has the density 1.0. As exception, if the clustering results in a single cluster containing all points, its ρ_{cluster} is set to 1.0.
The final density ρ_{final} is the minimum of all ρ_{cluster}.

5) All clusters are evaluated. Clusters with $\rho_{\text{cluster}} < 1.0$ will be deleted as they are not sensible. Optionally, if ρ_{cluster} is too low or clusters contain too few data points they are deleted as well. See the next section of a description of these optional parameters. The result of this algorithm step is a set of clusters that are subsequently numbered.

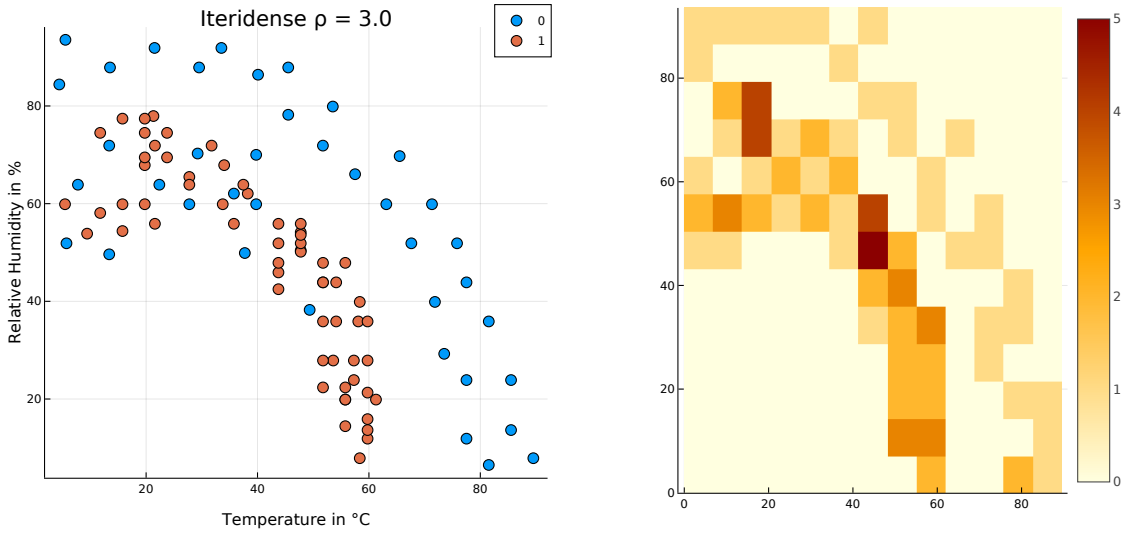
6) The resolution is incremented by one and the steps 1 – 5 are repeated until either $\rho_{\text{final}} > \rho$ or until as many clusters were detected as specified as **MinClusters**. To assure that the steps are not repeated forever, the loop is stopped if the resolution reaches the total number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

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 $\rho = 3.0$. The final resolution is 13, there is one cluster with $\rho_{\text{cluster}} = 4.32$.



(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 $\rho = 3.0$.

3.2 Options

The algorithm can optionally be modified this way:

1) 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.
2) 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.
3) 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.
4) 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.
5) Specification of a resolution at which the loop step 1–5 is stopped (**StopResolution**).

Option 1 can speed up the computation significantly.

Option 2 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.

Option 3 is useful to exclude unsuitably small clusters. It is recommended to set **MinClusterSize** $\geq D + 1$ where D is the dimensionality of the data set.

Option 4 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.

Option 5 prevents undesired many loops. For example if ρ was set to a high value and no cluster will be found.

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

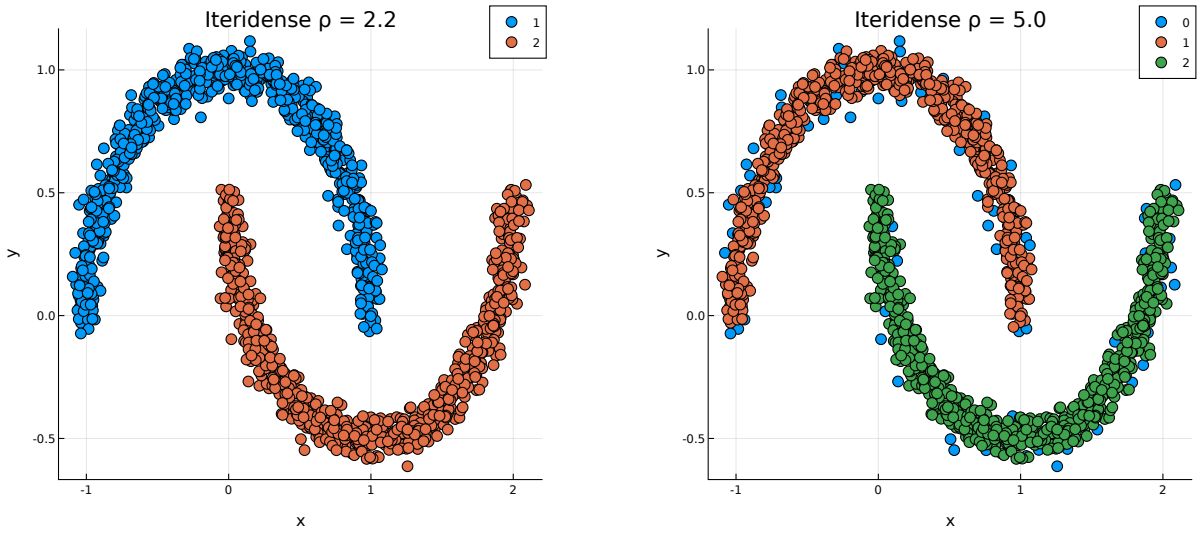
A reference implementation of ITERIDENSE in the programming language JULIA is online available [7]. Its outputs are besides the assignments of the points to clusters also the size of clusters, density of clusters, final resolution, number of clusters, the count map as tensor in the final resolution (the 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].

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 $\rho = 5.0$.



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

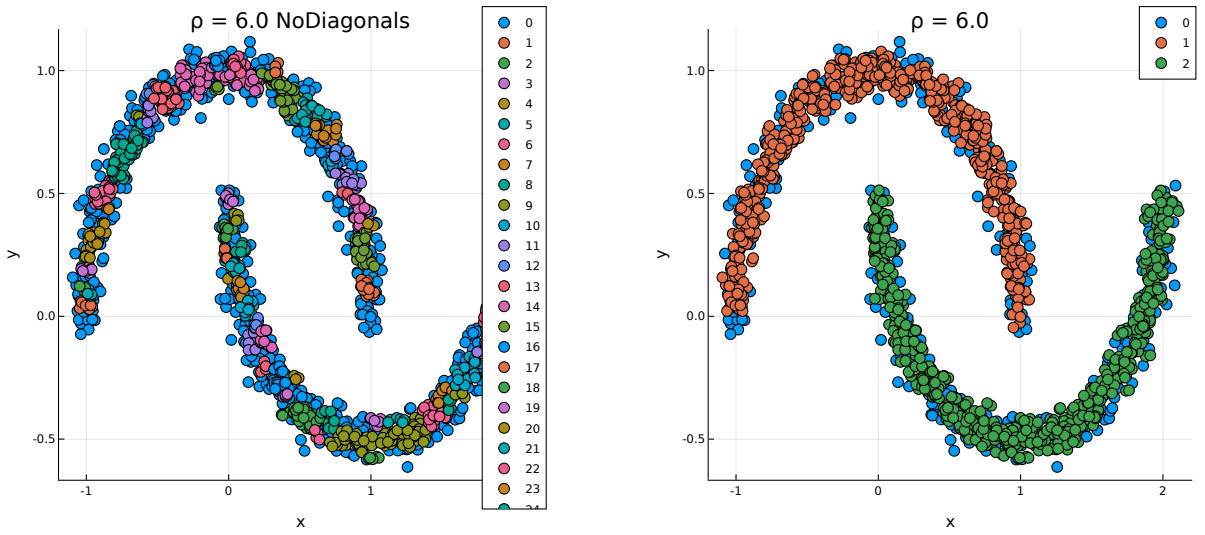
(b) $\rho = 5.0$, resulting resolution: 43

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

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 division between core points, outliers 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) $\rho = 6.0$ with option **NoDiagonals**, resulting resolution: 68

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

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

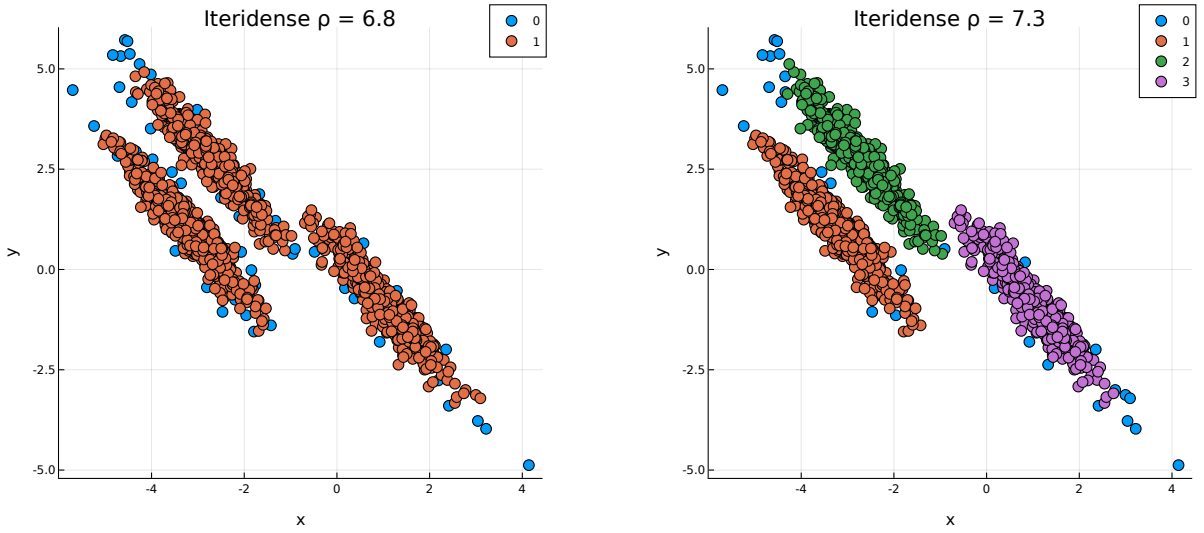
- Either start with a low ρ and increase it gradually to get a suitable result. If the result is not suitable, look at the resulting ρ_{cluster} and set for the next run ρ above their minimum.
- Or specify with **MinClusters** the desired number of clusters. If the result is not suitable, increase gradually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

$\epsilon = 0.1$ 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.

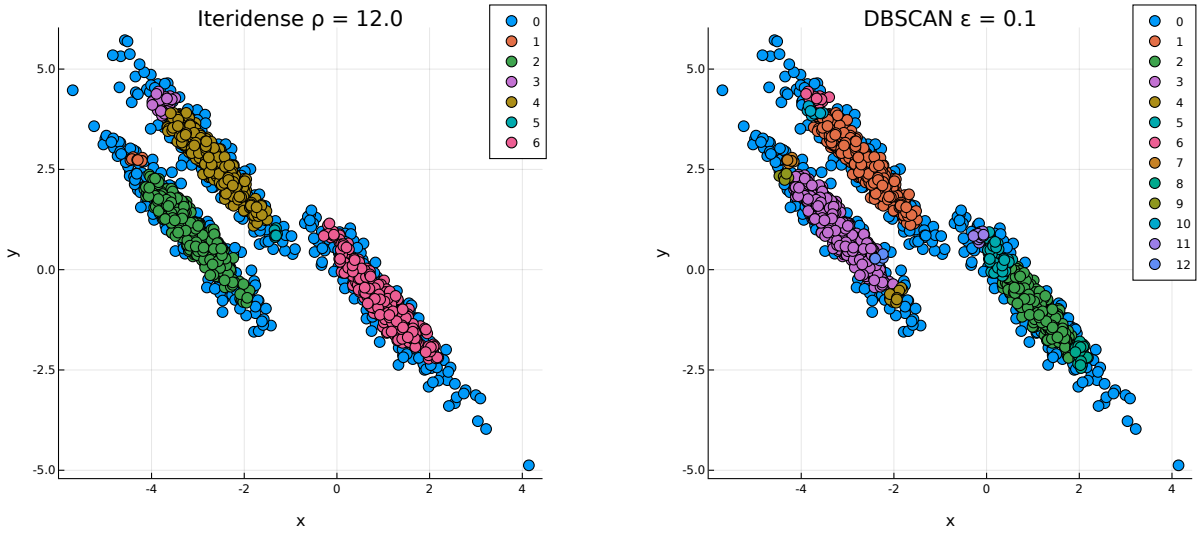
Fig. 10 (a) is another example with 1500 data points, generated using the *make_blobs* call to *sklearn.datasets* with a subsequent transformation. **MinClusters** was set to 2 and **MinClusterSize** 20. Cluster 2 has $\rho_{\text{cluster}} = 3.07$ and is a merge of a high- and low-density region. The two ways to change or to achieve a certain result are:



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

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

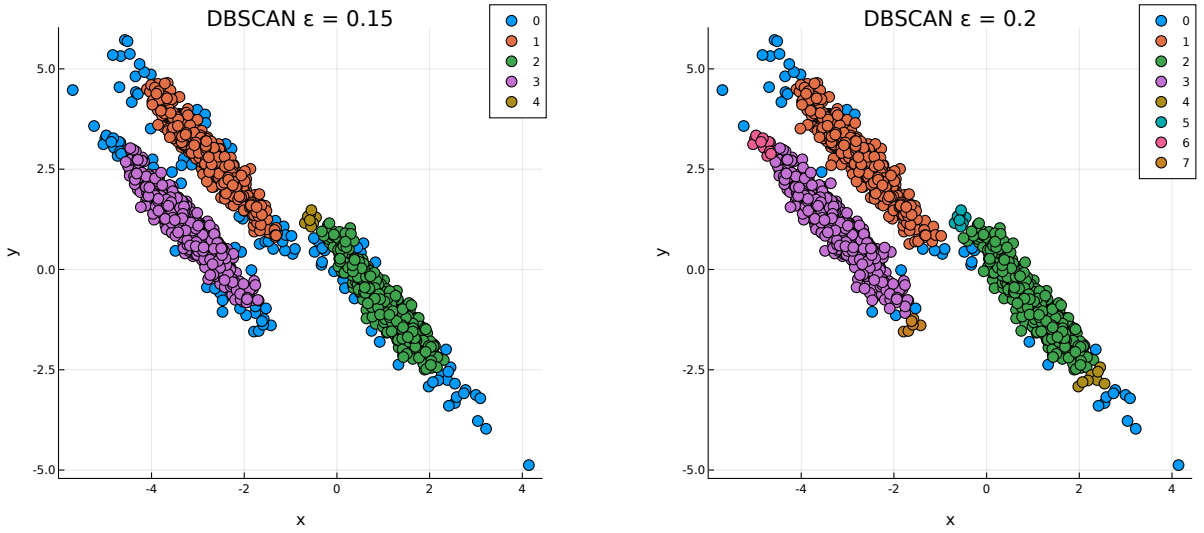
Figure 6: Result of ITERIDENSE for $\rho \geq 7.3$ **MinClusterSize** $= 6$ at anistotope clusters.



(a) $\rho = 12.0$, resulting resolution: 84

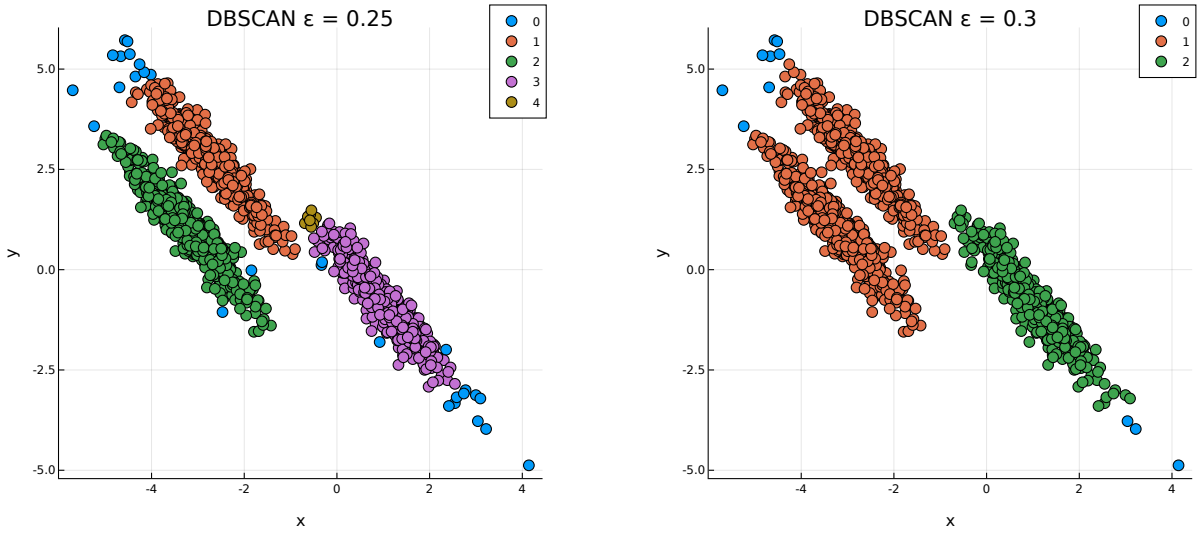
(b) $\epsilon = 0.1$

Figure 7: Result of ITERIDENSE for $\rho = 6.8$ and DBSCAN for $\epsilon = 0.1$ at anistotope clusters.



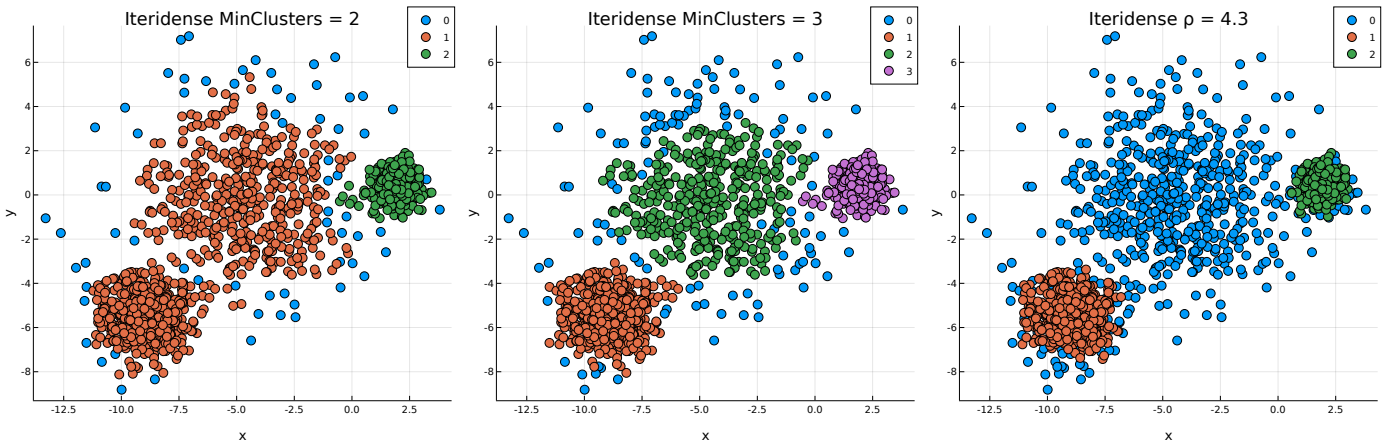
(a) $\epsilon = 0.15$ **(b)** $\epsilon = 0.2$

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



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

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



(a) MinClusters = 2 **(b) MinClusters** = 3 **(c)** $\rho = 4.3$, **MinClusterSize** = 50

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

- 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. But that this is the way to go is a feature of the the ITERIDENSE algorithm.

4.3 Computation Performance

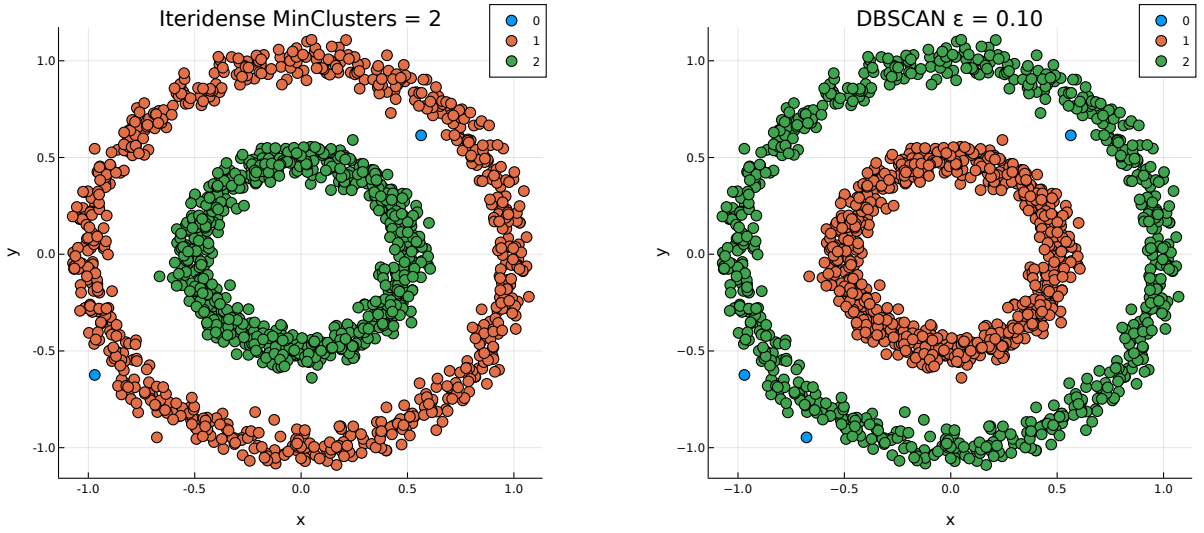
The data shown in Fig. 11 was generated using the *make_circles* call to *sklearn.datasets*.

In that case, there must be 2 clusters, therefore one can specify **MinClusters**. This will lead to a shorter computation as with the specification of ρ since the algorithm loop is stopped as soon as 2 clusters are detected, otherwise there might be further loops. For our example 10,000 cluster runs with **MinClusters** = 2 took $\approx 9.4\,\mathrm{s}$[2] and the final resolution was 13. For $\rho = 2.0$ the final resolution was 25 and with $\approx 24\,\mathrm{s}$ more time was needed. By starting the algorithm at a higher resolution, e. g. with **StartResolution** = 10 the time would decrease from $\approx 9.4\,\mathrm{s}$ to $\approx 7.1\,\mathrm{s}$.

Fig. 11 (a) shows the ITERIDENSE result. For comparison Fig. 11 (b) shows the result using the DBSCAN algorithm for $\epsilon = 0.1$ and **MinPts** = 3.

The benefit of ITERIDENSE compared to density-based algorithms is the computation time. In this example the dataset has 1500 data points. To generate the count map in algorithm step 1, every data point has to be evaluated in form of generating its coordinate value for the current grid coordinate system. This is the most costly operation, especially for high-dimensional data. Afterwards, only the cells have to be evaluated, not the data points. For DBSCAN in comparison the coordinates for every data point are only read out once but the distances between every point and neighboring points have to be calculated. The larger ϵ or the greater the point density the more points are neighbors and have to be evaluated. This can become very computation-intensive. For the data of Fig. 11 with $\epsilon = 0.1$

2. The times were measured using the feature *@elapsed* of the JULIA programming language.



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

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

and **MinPts** = 3 10,000 cluster runs took ≈ 17.8 s while the same with $\epsilon = 0.5$ took ≈ 69 s and with $\epsilon = 0.5$ both clusters were even not detected.[3] Therefore setting ϵ to a suitable value does not only affect the result a lot but also the computation time. Since there is no clear path on how to change ϵ to get a useful result, more cluster runs have to be performed than with ITERIDENSE.

For ITERIDENSE the worst case in terms of computation is to set ρ so high that the algorithm runs 1500 loops as there are 1500 data points. A single run of this takes ≈ 13.4 s. So 1500 loops take longer than 10.000 times the 12 loops from the default **StartResolution** 2 to the final resolution 13. This is because the number of cells in the grid scales in 2 dimensions squarely with the resolution. To prevent that undesired many loops are run, the parameter **StopResolution** can be set.

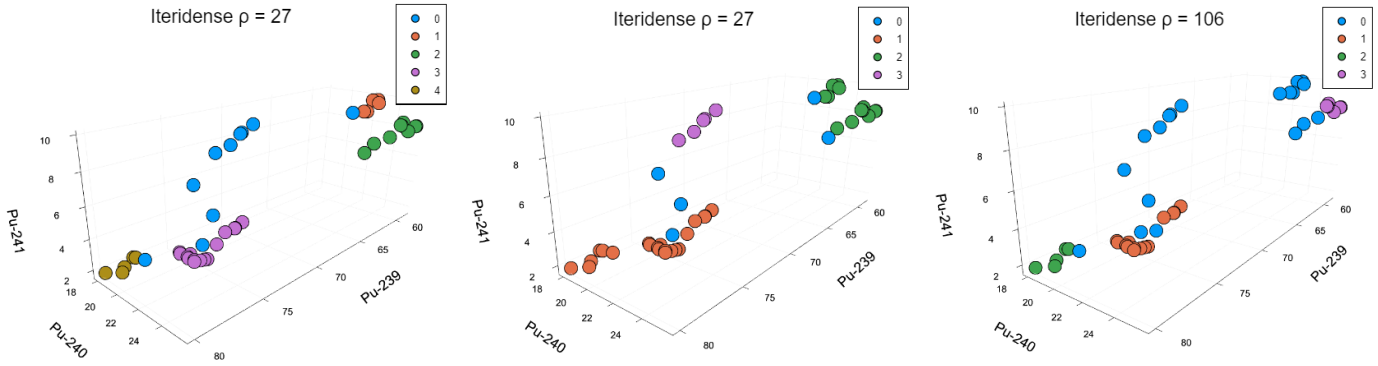
4.4 Results in higher Dimensions

Data with higher dimensions are a main use case for clustering algorithms as no human could do the clustering according to plots. Fig. 12 is an example and also demonstrate the clustering performance of ITERIDENSE. The data are the publicly available *pluton* dataset[14] 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. 12 (a). Taking all dimensions into account, the result is Fig. 12 (b). The points of high 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 $\rho_{\text{cluster}}$and ends up with Fig. 12 (c). The high value for ρ is the result of the high dimensionality of the data.
For comparison DBSCAN cannot find in this dataset more than 2 clusters.

Fig. 13 shows a use case in which ITERIDENSE shows its strengths. The data is the publicly available *PhDPublications* dataset [15]. It has 6 dimensions and we selected 3 for the visualization. Since there are 5 classes in the set, **MinClusters** was set to 5. By evaluating only the 3 shown dimensions, one

3. The computation time measurement of the DBSCAN algorithm was performed using its implementation in the package *Clustering* of the JULIA programming language [13].

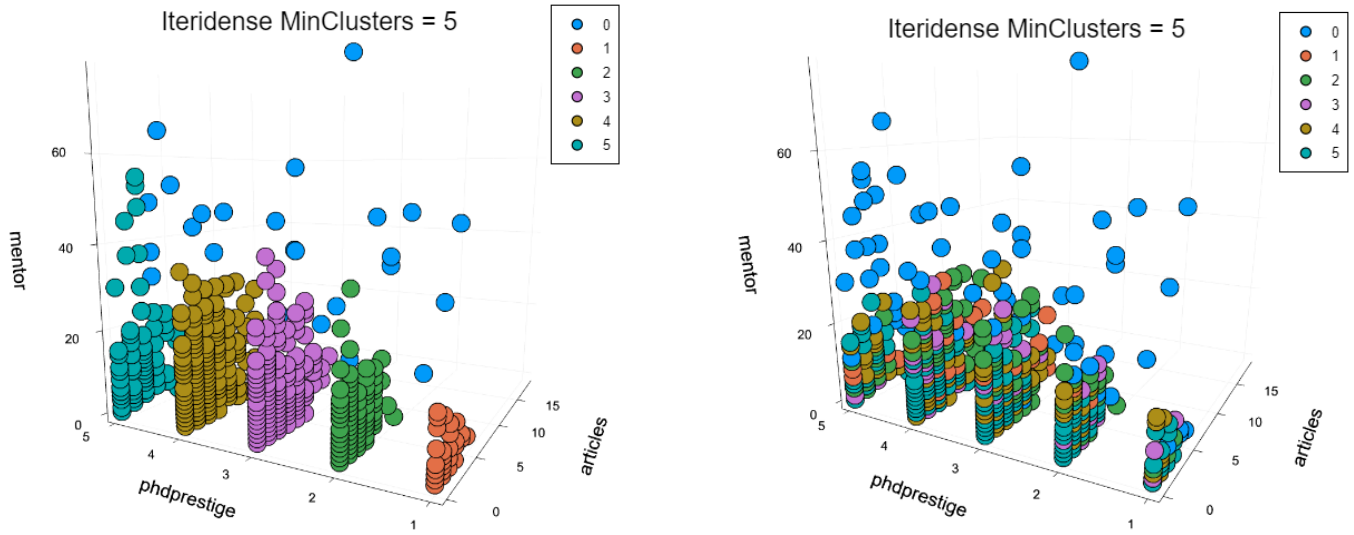


(a) Clustering with only shown dimensions taken into account.

(b) Clustering with all dimensions taken into account.

(c) like (b) but $\rho = 106$

Figure 12: Result of ITERIDENSE for the *pluton* dataset.



(a) Evaluation of the 3 shown dimensions

(b) Evaluation of all 6 dimension

Figure 13: Result of ITERIDENSE on the *PhDPublications* dataset.

gets Fig. 13 (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. 13 (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.

5 Discussion

As shown in the previous sections, 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 data set [16].

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 the 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 resolutions. This costs more computation efforts but one does not have 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 introducing for some practical applications trial and error. There are approaches to improve initial the setting of the DENCLUE parameters, see [17], 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.

6 Conclusions

This paper introduced ITERIDENSE, an iterative clustering algorithm combining grid-based and density-based methods. It is simple to use because it does not require to estimate settings in advance and because it provides two possibilities to run the clustering and for both a clear path on how to change the algorithm's input parameters to achieve suitable results. ITERIDENSE is applicable for datasets with any dimensionality.

The ITERIDENSE algorithm provides shorter computation times than pure density-based algorithms. 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 ITERIDENSE performs clustering as good to the DBSCAN algorithm and for cases of high-dimension datasets even better.

ITERIDENSE provides different options to affect either the clustering result and to improve the computation time. We provide online a reference implementation together with an example worksheet, [7], demonstrating the clustering with ITERIDENSE on the datasets shown in this paper.

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. Apr. 2025. URL: https://codeberg.org/Soloof/Iteridense.

[8] Uwe Stöhr. *Iteridense-package*. Internet. May 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] Rdatasets Team. *Doctoral Publications*. Ed. by Vincent Arel-Bundock. Internet. URL: https://vincentarelbundock.github.io/Rdatasets/doc/cluster/pluton.html.

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

[16] 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.

[17] 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.