INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

REPORT
TP UNSUPERVISED LEARNING
CLUSTERING ALGORITHMS

December 18th, 2019

Written by TRAN Trong Hieu & TRAN Le Minh
Group A1 - 5SDBD
Project link : https://github.com/kuro10/Apprentissage.git

Table of contents

I. Introduction

Machine Learning (ML) nowadays is an important field of computer sciences that gives computers the ability to learn without being programmed. The two main categories of machine learning practice today are commonly known as supervised learning and unsupervised learning. In fact, unsupervised learning task is a far more challenging area of ML, where we learn from unlabeled data instead of labelled data in case of supervised learning. In this practical work, we mainly focus on exploring different clustering algorithms.

Clustering consists of an important unsupervised learning problem. It deals with finding a reasonable structure of a dataset that are not yet well-understood. This task aims to divide unlabeled data points into clusters, where each one is a collection of data samples that are similar to others within the same cluster but dissimilar to the samples in other clusters. There are many approaches with differents methods that can be used to cluster an unknown dataset, each with their own advantages and drawbacks. Also, there is no objective criterion for determining a “good” cluster or a “correct” clustering algorithm. Hence, the choice of clustering methods clearly depends on the data type as well as the particular purpose and application, thus it needs to be chosen experimentally. The major well-known clustering methods, that we will explore in the following parts, can be categorized into :

- *Centroid-based clustering* with an example of ***K-Means clustering***
- *Connectivity-based clustering (hierarchical clustering)* with **Agglomerative clustering** approach
- *Density-based Clustering* represented by ***DBSCAN***

Besides the above-mentioned categories, there are still others approaches such as *distribution-based clustering* (Gaussian Mixture Model is a common example of this approach) or hybrid models in clustering. In this experimental work, we also cover a new hybrid clustering algorithm called ***Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN),*** which is an extension of DBSCAN by combine it with hierarchical clustering approach.

Briefly, the goal of this experimental implementation is to provide an overview about how these algorithms work and the motivations behind them as well as a summary and comparison of their performance with different types of dataset.

II. Implemented methods

1. K-Means : Centroid-based clustering

We start our practical work on clustering method with K-Means Clustering. This is a very popular unsupervised learning method, thanks to its simple implementation, simple to understand, straightforward methodology that provides most of the time a robust and effective performance especially on partitioning most datasets with numerical attributes. Its basic idea is the centroid-based clustering, which regard the center of data points as the center of the corresponding cluster, seeking to partition the dataset into equal clusters.

a. <u>What is K-Means clustering and when to use it ?</u>

The core idea of K-means algorithms is : for the predetermined ***K*** number of clusters, this method will update iteratively the ***K*** cluster centers and assign the data points to the nearest cluster center in order to respond to the two following criteria :

- The *inter-cluster* distance, i.e. distance from the each point within a cluster to its center, is *minimized* so the cluster cohesion property is ensured

- The *intra-cluster* distance, i.e. distance from one cluster to another, is *maximized* so the cluster separation property is guaranteed

This iterative process will be continued until these two criteria of convergence are met.

Algorithm-wise, firstly the algorithm assigns ***K*** data points as the initial centroids based on the chosen criteria such as *random* choice or "*k-means++",* where centroids are initialized far away between them. Then it will move iteratively the centroids to the centers of the samples, seeking to minimize the inertia, also known as the within-cluster sum-of-squares criterion, until it reached the defined tolerance, in other words the centroids do not move significantly.

We can firstly observe that this algorithm optimizes cluster centers instead of cluster borders. Furthermore, the distance metric used in k-means algorithm is the squared Euclidean distance, hence it is very good at identifying clusters with flat geometry, particularly with a spherical shape. This is why it prefers clusters with approximately similar size ***(1)***. Additionally, the official documents state that "the inertia makes the assumption that clusters are convex and isotropic", so it might not perform well with clusters with irregular shapes ***(2)***. Moreover, even when a dataset does not satisfy the two previous conditions, K-Means can still effectively perform on that dataset if their clusters are very well separated, thanks to "***k-means++"*** which initializes centroids by choosing starting points that are very far from each other ***(3)***.

For all of these reasons, we deduce k-means clustering's best use case, illustrated in ***Figure 1.***, which include the following properties :

- ***(1)*** => The dataset has even cluster size
- ***(2)*** => It has flat geometry which contains well convex shapes
- ***(3)*** => It should be very well separated

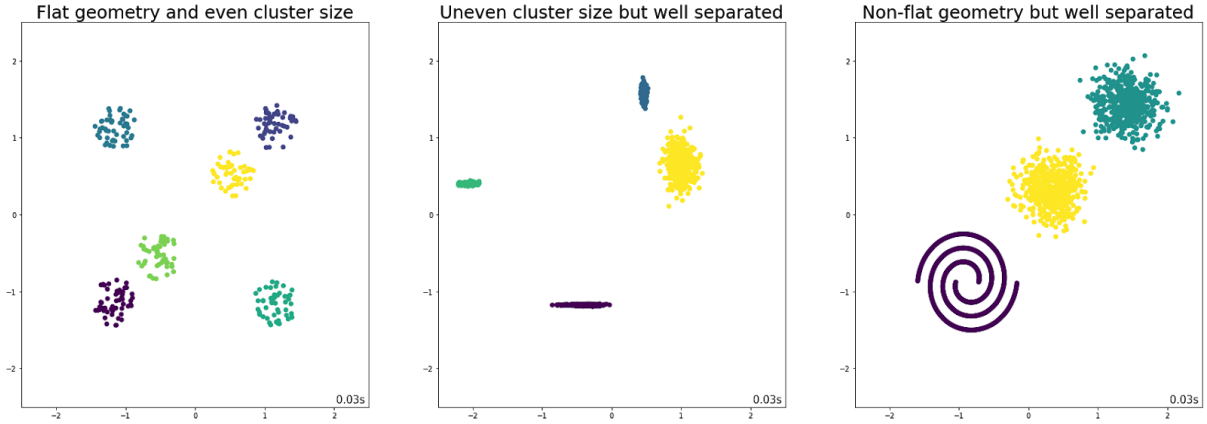

Figure 1. *Applying K-Means Clustering on three best use cases*

b. <u>How to optimally exploit K-Means Clustering's parameters ?</u>

The fact that K-Means Clustering has only one important parameter - the number of clusters **K** - explains why it is relatively simple to implement. But in some cases, the choice of ***K value*** might not be obvious, and it will be required to verify the clustering quality. Here we will present two differents experiments that compute the clustering quality and help to choose an optimal ***K value***.

Firstly, we explored K-Means Clustering's parameters on ***"spherical_5_2"*** - a typical use case of this method that satisfies all the requirements we have recently listed. By

observation, we can see that this dataset consists of 5 globular-formed clusters. But is 5 an optimal number of clusters ?

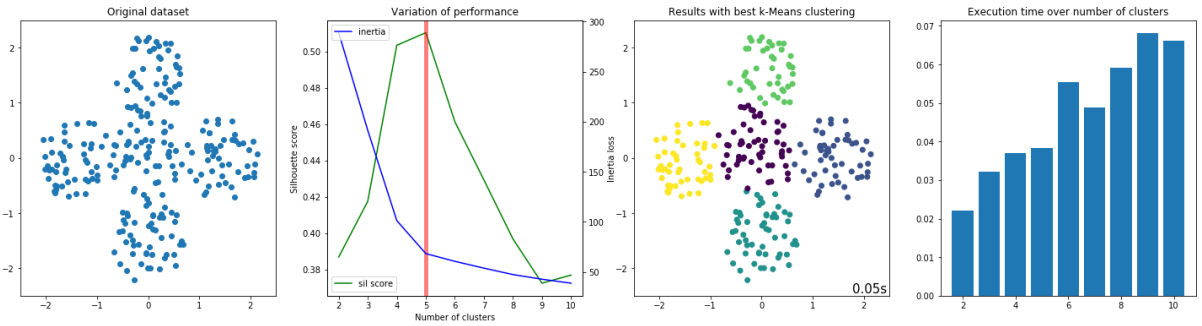

Figure 2. *Computing silhouette score, inertia loss and execution time by different k value, then apply k = 5 on "spherical_5_2"*

Our first experiment is to use the *"Elbow method"*, where we will study the variation of the inertia compared to ***k value***. We will then choose a ***k value*** that decreases remarkably the inertia, after which the higher ***k value*** would have less effect on inertia. In this case, as shown in ***Figure 2.***, 5 is indeed a suitable ***k value*** that satisfies that demand. However, since the decision was made by observation, this technique might seem relatively subjective.

That is why we carried out a second experiment, which is to compute and compare the silhouette score by different ***k values*** ranging from 2 to 10. ***Figure 2.*** shows once again that 5 is the pertinent ***K value***, considering its highest silhouette score.

In brief, the third frame of ***Figure 2.*** shows how the dataset looks like after we applied K-Means Clustering with the parameter ***k value*** of 5. The fourth frame also suggest that the number of clusters affects directly on the algorithm's complexity, which explains why the execution time increases by number of clusters. Additionally, we could observe that 4 is also a reasonable number of cluster, by observation as by two verification techniques. This fact emphasizes even more the importance of verifying the clustering quality before proceeding to a relevant number of clusters.

c. *What should we consider when using this method ?*

From the previous sections, based on its algorithm as well as its performance during its best use case, we firstly observe that, thanks to its nature of optimizing cluster centers, k-means clustering method has a pretty fast computational speed, and a potential scalability towards large datasets. It is also relatively simple to implement, as it only requires to provide the number of clusters.

However, this only parameter, i.e. number of clusters, is considered as a drawback too : in most of real life use cases, the number of clusters is not evident, so it might take efforts to find the optimal ***K value***, and a poor choice of ***K value*** will provide a poor output. Furthermore, since real-world datasets often contain many noisy points and the algorithm searches iteratively for a centroid, noise will also be considered in the calculation and distorts the final result. ***Figure 3.*** shows how sensible k-means clustering is facing a dataset that contains noises. Moreover, since ***K value*** has a direct impact on the algorithm's complexity, this method will not scale well on dataset with too many clusters.

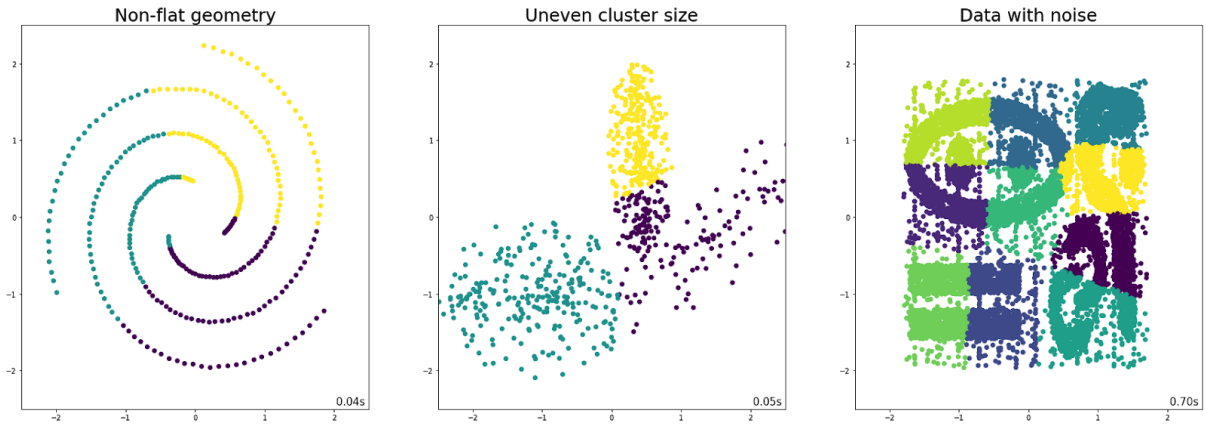

Figure 3. *K-Means Clustering showing its weaknesses*

Besides, the algorithm also states that the starting centroids need to be randomly initialized, so the performance's stability is not guaranteed, especially as this way, sometimes the algorithm only converges to a local optimum. Also, as we mentioned above, K-Means clustering is not a suitable method for dataset with non-flat geometry or non-convex as well as with eventually uneven cluster size or variable density.

2. Agglomerative Clustering : Connectivity-based or Hierarchical clustering

Through the first section, we have familiarized with the K-Means Clustering technique which relies heavily on dataset's convexity because of its nature of optimizing centers. That said, since data might not always have a regular shapes where k-means clustering express its best performance, this method performs poorly on datasets with irregular shape.

So, let's move on to the next section with the new method : Agglomerative Clustering, whose basic idea is to construct the hierarchical relationship among data for clustering, to see how does it behave facing that problem.

a. How can Agglomerative Clustering overcome K-Means' limits ?

Agglomerative clustering is in fact a strategy of hierarchical clustering : by assuming that objects are more related to nearby objects than to objects farther away, this technique connect objects to form clusters based on their distance. It yields as output a dendrogram showing the hierarchical connection between clusters that merge with each other at certain distances.

Algorithm-wise, agglomerative clustering is the "bottom-up" approach to build the clusters hierarchy tree, where each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. More precisely, the algorithm starts with each data point as a leaf, which will be connected by pairs to form clusters if the distance between them satisfies a certain distance criteria. The next step is to continue to pair these recently formed cluster together in order to create larger clusters, also based on that distance criteria. This step is repeated until all data points belong to one single big cluster, i.e. when we obtain the complete dendrogram. Finally, based on the desired cluster number, this tree will be cut at a certain height in order to provide a satisfied partitioning of the dataset.

Through the algorithm, we can see that firstly, agglomerative clustering can be used comfortably in case of any number of clusters, as this parameters are only used to determine

the suitable cut on the dendrogram without affecting the hierarchy tree's quality as well as the overall complexity. Furthermore, regarding the approach of iteratively linking every two points or two small clusters, this method performs very well on data with uneven cluster size ***(4)***. However, also because of such approach, the execution time will be exponentially high on large dataset, but this problem can be addressed by adding a connectivity constraint that will boost remarkably the computation speed ***(5)***. Moreover, with different linkage parameter, agglomerative clustering can be employed with any pairwise distance metric, thus works well on even dataset with non-flat geometry ***(6)***.
Hence, we deduce the potentially best use case for Agglomerative clustering, illustrated in ***Figure 4.*** :

- ***(4)*** => the dataset has uneven cluster size
- ***(5)*** => it has a possible connectivity constraint
- ***(6)*** => it is eventually non-convex

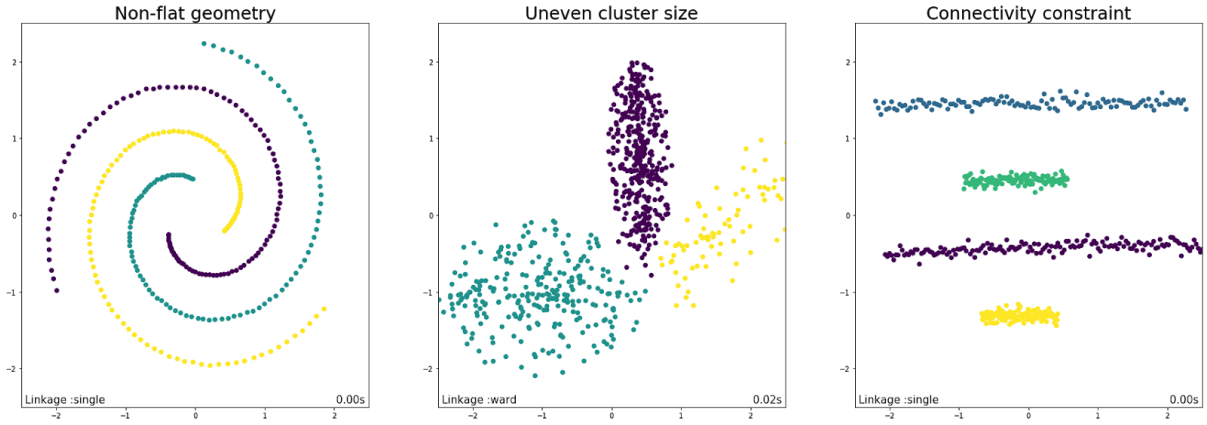

Figure 4. *Applying Agglomerative Clustering on three best use cases*

b. <u>How to optimally exploit Agglomerative Clustering's parameters ?</u>

As we mentioned above, there are two important parameters to be considered while using Agglomerative Clustering : the number of clusters and linkage method. While the former are only used to determine the suitable cut on the hierarchy tree, the importance is on the latter, which will define how should two points or two clusters be linked. Actually there are four well-known linkages methods, each having their own use cases, which are :

- Single linkage, which considers the shortest distance between pair of clusters, so it tends to produce long, loose, non globular clusters, but very sensible to noise.
- Complete linkage, which considers the longest distance between pair of clusters, so it tends to produce more compact clusters, but still sensible to noise.
- Average linkage, which considers the mean distance between all pairs of points, so it produces more homogeneous clusters, also it is less sensible to noise.
- Ward, which seeks to minimizes the total within-cluster variance, so it aims for clusters which are more dense and more concentric towards its middle.

In this part, we want to focus more on the ability of Agglomerative Clustering to deal with non convex data, which consists of a weakness stated in K-Means Clustering. For this reason, we will demonstrate our parameters choice for the *"dartboard1"* dataset, on which K-Means clustering has a poor performance because it contains non globular clusters.

By observation, we would want to see obviously four clusters according to four circles of this dataset. For the linkage choice, among four most well-known methods, the single linkage method is indeed the most suitable for our use case. So, in the end we obtained the

following clustering result, shown in the first frame of ***Figure 5.***, with the previous configuration.

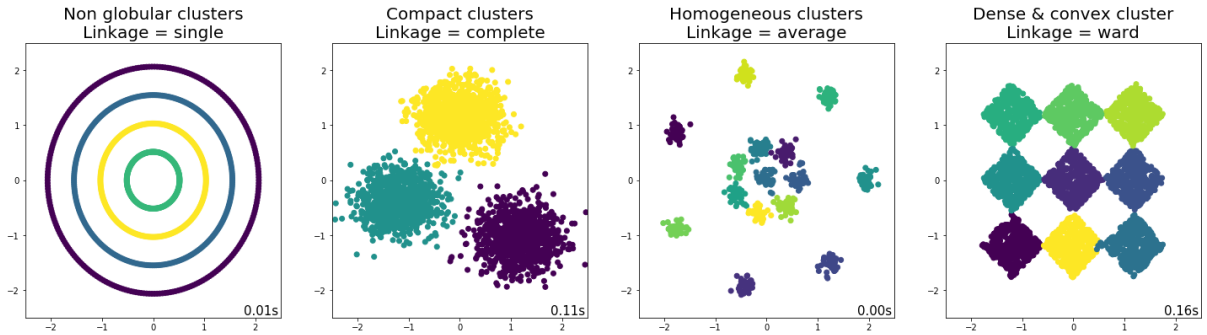

Figure 5. *Different linkage methods for different use cases*

c. *<u>What should we consider when using this method ?</u>*

Through the previous section, we notice that Agglomerative Clustering is very versatile with different linkage criteria for different data type, different use case. Most notably, it provides a very effective solution for the problem of non-convex data we mentioned at the beginning of this section. Also, since the number of clusters have no impact on algorithm's complexity, it might provide a better execution time compared to ***K-means*** if the dataset has many clusters.

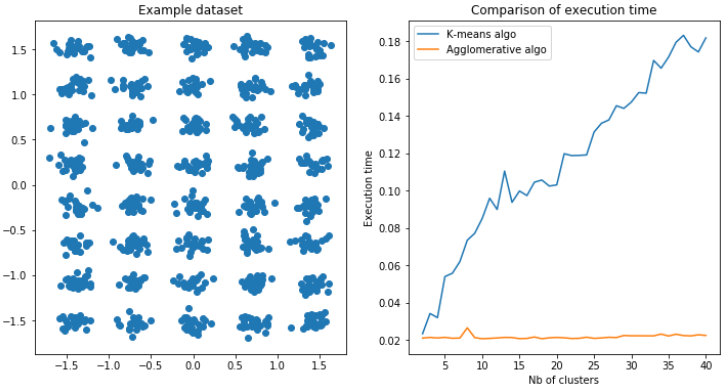

Figure 6. *Execution time comparison between K-means and Agglomerative over number of clusters*

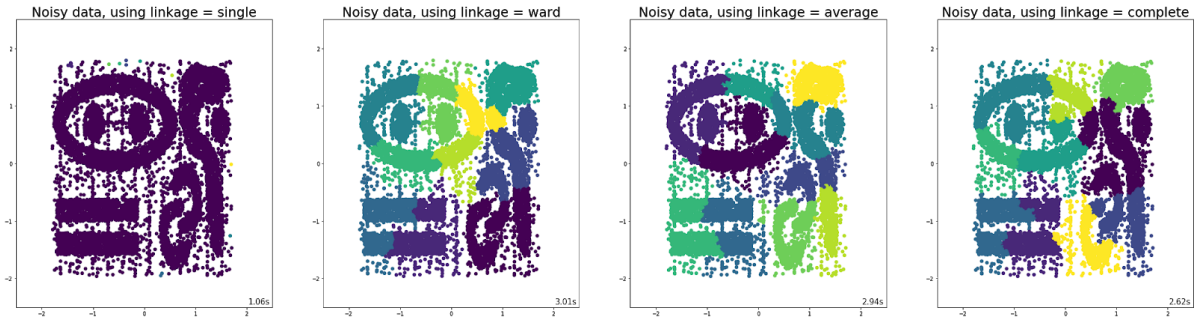

Figure 7. *Attempt to apply Agglomerative with different linkage on noisy dataset*

However, Agglomerative clustering still possesses some drawbacks. Firstly, the algorithm does not support backtracking, which means it cannot undo an unsatisfactory linking between two points or two clusters. Furthermore, this method does not scale well on large dataset because of its time complexity by the approach of trying to merge every data points together until it obtains a hierarchy tree. That is why, without a connectivity constraint, it will

show a very long execution time. Moreover, an attempt to execute agglomerative clustering on large dataset, although eventually successful, might yield an unreadable dendrogram, which makes the decision on correct number of clusters and the result interpretation harder. Also, even with different linkage available, Agglomerative Clustering is still sensitive to outliers, so it is not a suitable method to cluster dataset with noise. Besides, the clustering still requires to specify number of clusters, which suggests that its algorithm does not support automatic clusters generation.

3. DBSCAN : Density-based clustering

Going through the first two sections, we might notice that noise was always a serious problem that affects the performance of centroid-based approach and the hierarchical approach. So, in this part, we will present ***Density Based Spatial Clustering of Applications with Noises (DBSCAN)*** - a powerful technique against noisy dataset. Indeed, its principle is density-based clustering, which states that clusters are defined as areas of higher density than the remainder of the dataset, meanwhile objects in sparse areas are usually considered to be noise and border points.

a. Why DBSCAN is a robust technique against dataset with noise ?

DBSCAN's algorithm possess the two following parameters : ***eps*** - which specifies how close points should be to each other to be considered a part of a cluster, and ***min_samples*** - the minimal number of points to form a dense region. This algorithm also adopts the density reachability notion, as well as three data point types here:

- If there are at least ***min_samples*** points within the range ***eps*** of a point *p,* it is considered as a **core point.**
- If a point *q* is directly density-reachable from a core point, i.e. it is within the reach of ***eps***, but having less than ***min_samples*** points in its ***eps*** neighbourhood, then it is a **border point.**
- All points not reachable from any other point are **noise** or **outlier**.

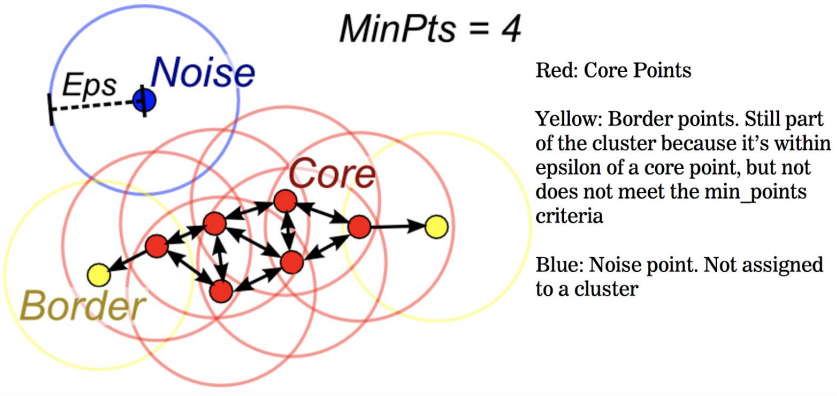

Figure 8. *Example of Core point, Border point and Noise identified by DBSCAN*

Since density-reachability is not symmetric, another notion is adopted: density-connectedness, which means *p* and *q* are density-connected if they are situated within the reach ***eps*** of each other. This relation is totally symmetric.

The previous relation allows us to define a cluster as follows : a cluster is a set of ***core points***, where all samples within that cluster are ***mutually density-connected***, and if a point is density-connected to any point of the cluster, in case of ***border points***, it is part of the cluster as well.

Therefore, the algorithm's functionality will be : commencing with an arbitrary point that has not been visited, if it is a core point, a cluster is started, else this point will be considered as noise; however it might be be found later in another ***eps*** neighborhood of a different point, hence becomes border point. Then, another point within the distance of ***eps*** of the previous core point is analyzed to see which point type it is, and this phase is continued until the density-connected cluster is completely found. The algorithm ends when all points are visited, at least once.

Through the algorithm, we can see that this method aims to separate clusters of high density from clusters of lower density, so clusters found by DBSCAN can be any shape ***(7)***. Furthermore, the way this algorithm find cluster based on density defined by the range ***eps*** and the minimal number of points within the range ***min_samples*** make it independant to the number of clusters, as well as clusters size ***(8)***. Above all, DBSCAN provides a robust approach to deal with outliers/anomalies within the dataset ***(9)***.

In brief, we deduce here an use case that can show the ultimate power of DBSCAN, presented in ***Figure 8.*** :

- ***(7)*** => the dataset has favorably a non-flat geometry
- ***(8)*** => it has uneven cluster size
- ***(9)*** => it contains noises

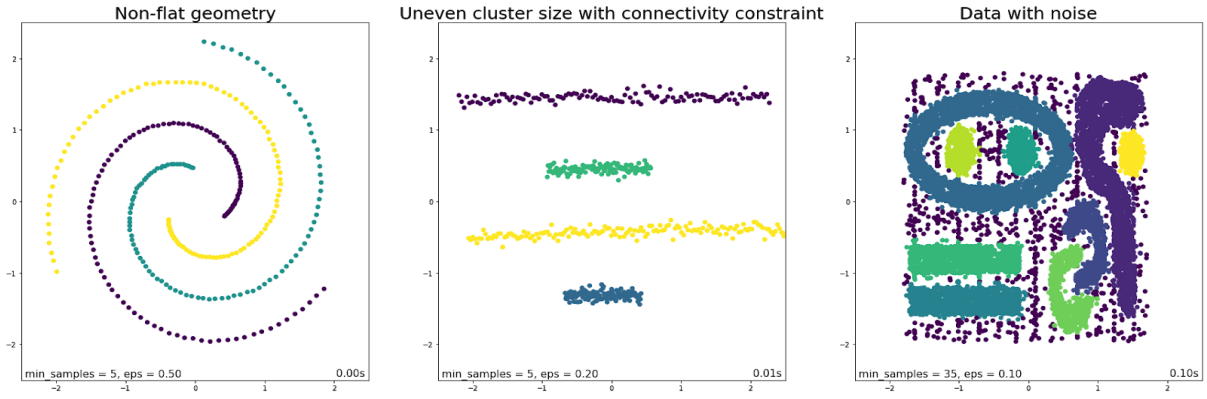

Figure 9. *DBSCAN showing its powerful performance, mostly on noisy dataset*

b. *<u>How to optimally exploit DBSCAN's parameters ?</u>*

As we mentioned above, DBSCAN requires two parameters : ***eps*** and ***min_samples***. While ***eps*** control the local neighborhood of the points, so crucial that it can be deterministic for the distinction between clusters and outliers, ***min_samples*** controls how tolerant the algorithm is towards noise. That is why a careful choice for these parameters' values is very important, especially for the ***eps***, as with a too small ***eps***, a large part of the data will not be clustered, thus considered as noise; and with a too large ***eps***, clusters will merge and the majority of objects will be in the same cluster. So the pertinent ***eps*** value should be based on the dataset's distance, but generally small ***eps*** values are preferable. As for ***min_samples***, it is recommended to choose a large value to effectively deal with large dataset or dataset with noises, which will form more significant clusters.

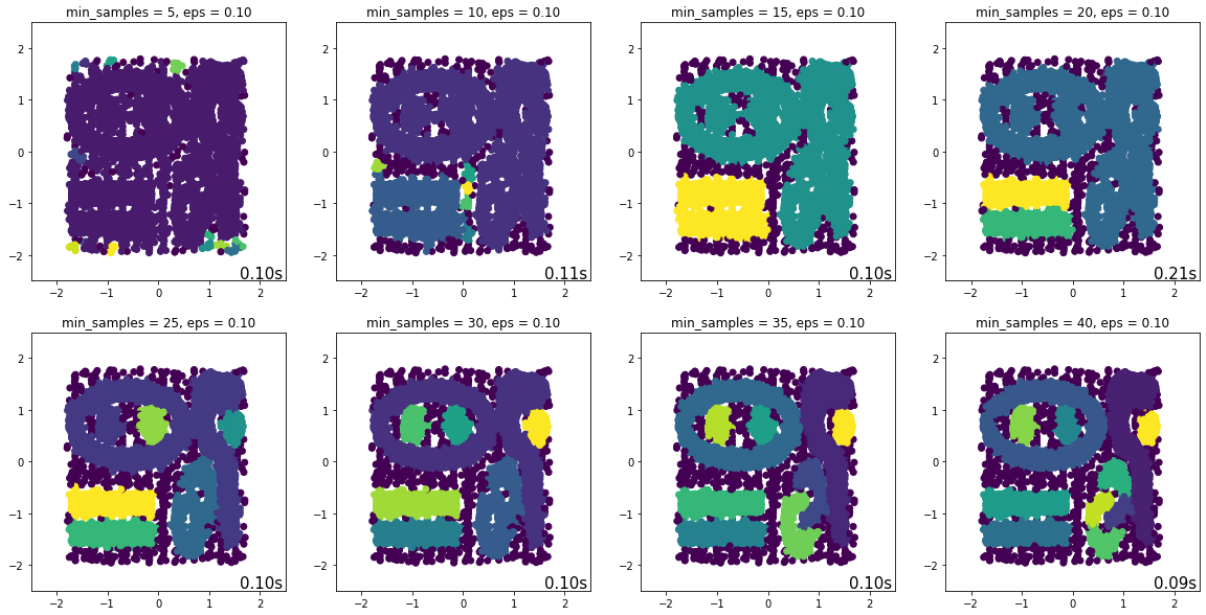

Figure 10. *Applying DBSCAN with eps = 0.1 and different min_samples on "cluto-t7-10k"*

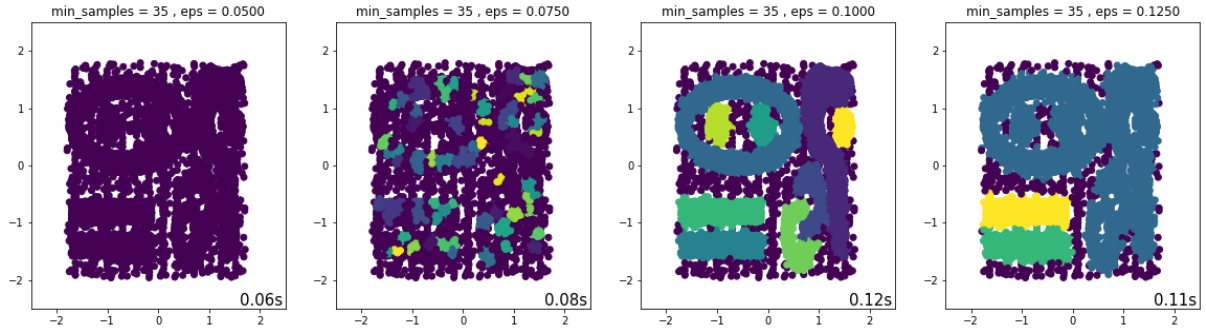

Figure 11. *Applying DBSCAN with min_samples = 35 and different eps on "cluto-t7-10k"*

Here we experiment with "***cluto-t7-10k***" dataset, which satisfies all the requirements we listed above. We observe that the clusters are tights, so we have chosen for ***eps*** a value of 0.1 . In order to find the optimal ***min_samples***, we try different values ranging from 5 to 40. ***Figure 10.*** shows that ***min_samples*** of 35 yield the best clustering result, meanwhile, ***min_samples*** prior to 35 make some clusters merge together, and after 35 they start to split into more clusters. Another experiment we carried out, presented in ***Figure 11.***, is to fix ***min_samples*** to 35 and vary ***eps*** to see if there is any more suitable ***eps*** value. In brief, the process of finding optimal parameters for DBSCAN is not simple, as it requires a lots of experiments and observations.

c. *What should we consider when using this method ?*

DBSCAN is indeed a very effective method to deal with outliers within the dataset, considering the fact that it adopts the density-based approach for the clustering. This approach also allows us to find clusters with different shapes. Furthermore, this method does not require to specify the number of clusters, which makes clustering more objective.
However, DBSCAN requires a careful parameters configuration, especially as they are not simple to choose in order to obtain a satisfactory clustering, which requires users to understand well the dataset.

4. HDBSCAN : An extension of DBSCAN

The previous section demonstrates that DBSCAN is a robust clustering technique against dataset with noise. But, in fact, it still has a serious drawback. The fact that the core distance ***eps*** need to be specified and stay constant throughout the execution makes the algorithm struggle to dataset with variable density. ***Figure 12.*** shows the attempt to apply DBSCAN on such dataset. That is why data scientists had carried out, since 2017, a new method named ***"Hierarchical DBSCAN"***, which comes as the extended solution to address this problem. It improves DBSCAN by adding the hierarchical clustering approach into the density clustering, then uses a technique to extract a flat clustering based on the stability of clusters.

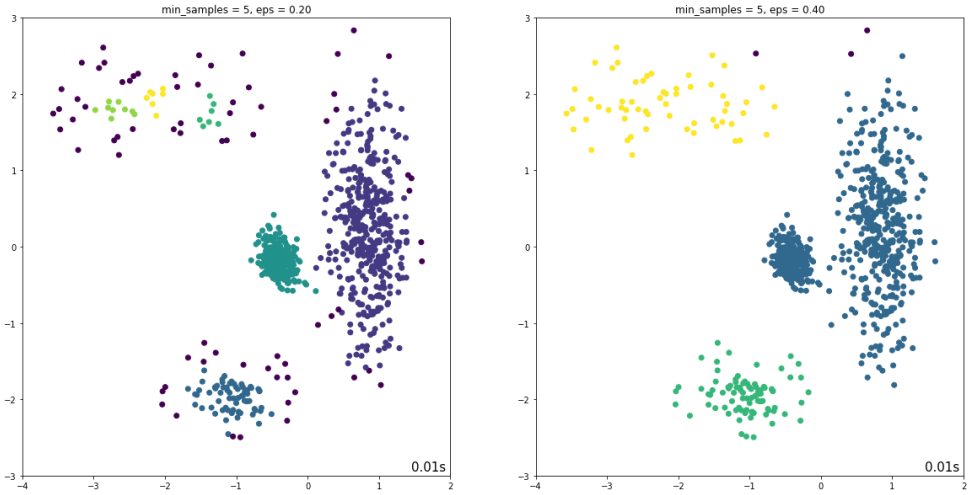

Figure 12. *A clear trade-off is observed why applying DBSCAN on variable density dataset, where in the first frame, two dense clusters are identified but the sparse is declared as noise; in the second frame, the sparse cluster is identified but two dense are merged together*

a. *What is HDBSCAN's difference compared to its original version DBSCAN ?*

Principle-wise, instead of asking users to provide the core distance, the algorithm now calculates it by itself, which is the distance between a point and its farthest neighbor defined by the ***min_samples*** parameters.

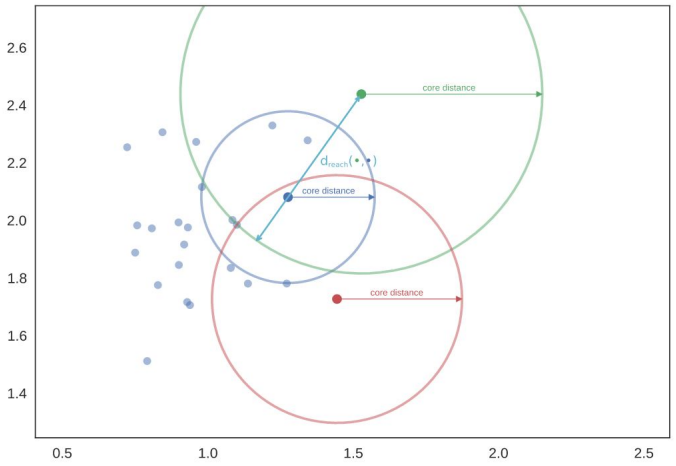

Figure 13. *Example of core distance identified by HDBSCAN*

Here, another distance metric is adopted in order to make noise more distant from each other and from clusters - the mutual reachability distance, which is define as such : mutual reachability distance between two points **A** and **B** is the maximum between **A**'s core distance, **B'**s core distance and distance between **A** and **B**. Like that, we can see that the larger ***min_samples*** is, the more points that are considered as noise.

The next step is to apply this new metric distance on the dataset. Then, we build a minimum spanning tree for the mutual reachability distance recently computed, which only connect data points whose mutual reachability distance is higher than a certain threshold. Given the minimal spanning tree, we can now convert that into the hierarchy of connected components, by sorting the edges of the tree by distance (in increasing order) and then iterate through, creating a new merged cluster for each edge.

Since this tree are still containing very large and complicated cluster hierarchy, it needs be condensed down into a smaller tree with a little more data attached to each node. Such condensed cluster tree looks like many persistent clusters whose the decreasing density over varying distance is represented as an upside down triangle shape. For this purpose, here we use another important HDBSCAN's parameter - the ***min_cluster_size***. With this parameter, we can now go through the hierarchy, separate clusters that does not reach that minimal size, have the larger cluster retain the cluster identity of the parent, or let clusters that satisfy the size persist in the tree.

Finally, in order to extract clusters, we would want to select the clusters in the previous condensed tree that have the largest total density area, ie the area of the upside down triangle; any point not in a selected cluster will be then considered as noise.

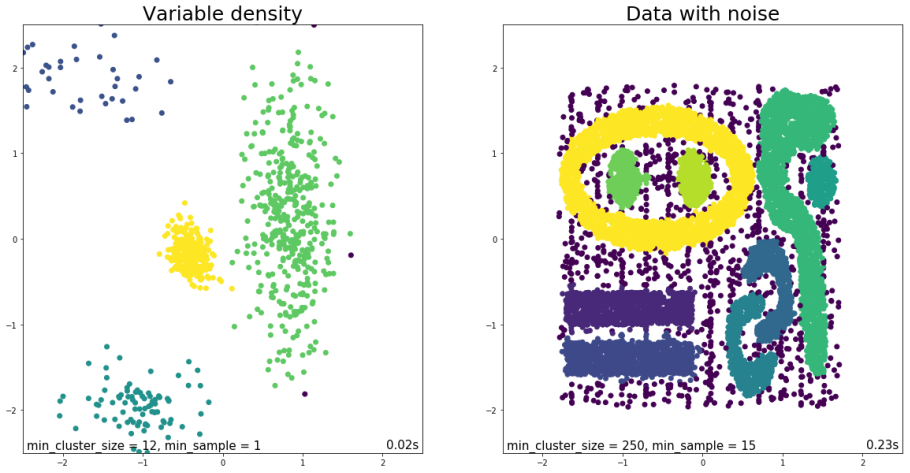

Figure 14. *HDBSCAN keep the performance it inherits from DBSCAN against data with noise, while overcoming the latter's weakness facing variable density dataset*

b. <u>How to exploit HDBSCAN's parameters to show its improvement over DBSCAN ?</u>

Through HDBSCAN's above principle, we can see that it seeks to correct DBSCAN's serious drawback of not able to cluster dataset with variable density. The new approach of applying hierarchical clustering into density-based clustering makes it more flexible to the dataset's overall density.

In order to verify this new advantage, we reapply HDBSCAN on the experiment we carried out previously with DBSCAN, first on DBSCAN's best use case - with the "***cluto-t7-10k***"

dataset, and then on the dataset with variable density that has revealed DBSCAN's weakness - the "**2d-4c-no4**" dataset.

In term of parameters, even though ***min_cluster_size*** and ***min_samples*** contribute a relatively complex role to the algorithm, according to what we demonstrated earlier, they can be simply understood as follows : ***min_cluster_size*** defines the smallest size grouping that we wish to consider a cluster, and ***min_samples*** provides a measure of how conservative the clustering will be, with larger the value will result in more points declared as noise, and clusters will be restricted to progressively more dense areas.

So, for the "***cluto-t7-10k***" dataset, considering the amount of noise and the density level of this dataset, we observe that ***min_samples = 15*** is a relevant value. About ***min_cluster_size***, we want the algorithm to consider the small globular group of points near the top right corner to be a cluster, so, according to ***Figure 15.***, after ***min_cluster_size*** of 250, that group of point is declared as noise, so a value prior to 250 will be suitable.

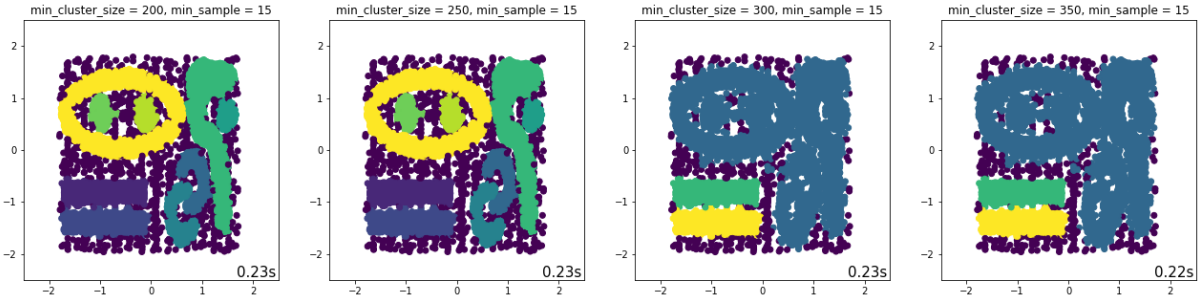

Figure 15*. Applying HDBSCAN with two different min_cluster_size on "cluto-t7-10k"*

In brief, the first experiment shows that HDBSCAN performs as well as DBSCAN on its best use case, and it even uses two parameters that can be very easy and intuitive to choose.

Now, let's move on to the second experiment where we will test the "DBSCAN's weakness" on HDBSCAN with the variable density dataset we mentioned above. The "***2d-4c-no4***" dataset is relatively simple with no noise and data are not very dense. So in this part, we fix ***min_cluster_size*** at 12 and then vary ***min_samples***. ***Figure 16.*** shows that indeed, the higher ***min_samples***, the more data considered as noise, so ***min_sample*** prior to 3 should provide the most suitable clustering, but most importantly we now see that HDBSCAN has successfully overcome DBSCAN's weakness.

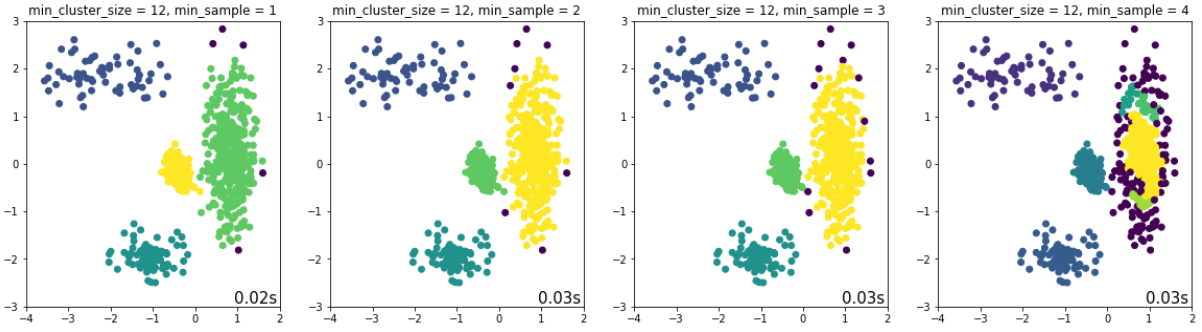

Figure 16. *Applying HDBSCAN with different min_cluster_size on "2d-4c-no4" dataset*

c. *What should we consider when using this method ?*

Through this section, we can observe that HDBSCAN still conserve the advantage it inherits from the original version DBSCAN, which is very powerful against dataset with noise and

performs well on dataset with any shape. Above all, it can now correct DBSCAN's weakness versus variable density dataset by bringing hierarchical approach into density-based clustering.

But, despite its robust performance, it is still somewhat difficult to choose optimal parameters, which will need many experiments. For this reason, this method should only be used on special use cases such as dataset with noise or non-flat geometry. Meanwhile, in would be much simpler to use general-purpose method, such as K-Means Clustering, for general use cases.

III. Comparison of studied methods

At the end of our practical work, we wish to carry out a comparison between four studied methods. So we decided to apply all four clustering techniques on each one's best use case. This way, we can have a panorama view of studied techniques, by observing their similarities/differences regarding their behaviour on different datasets. Such comparison is presented in ***Figure 17.***

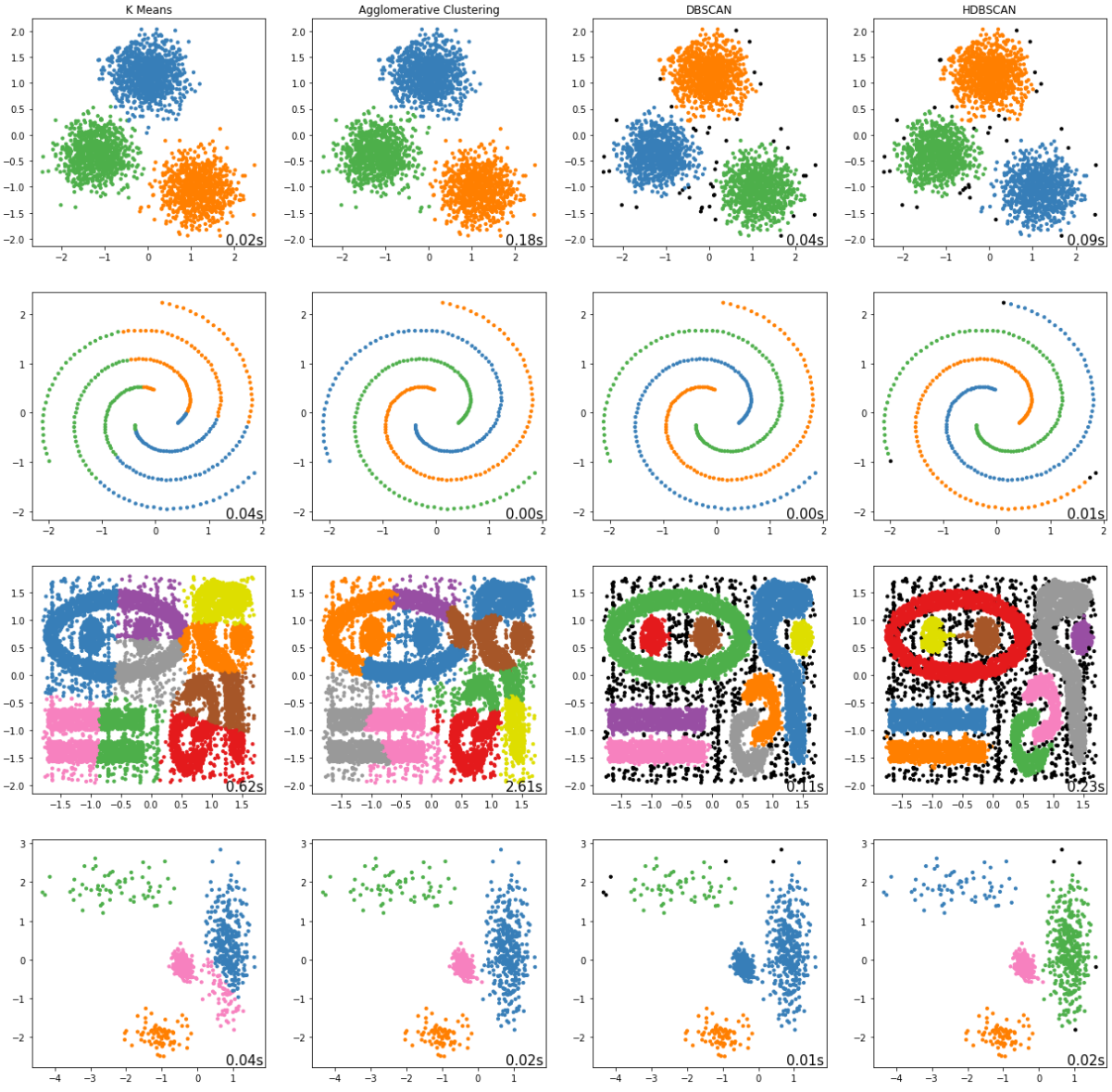

Figure 17. *Comparison of implemented methods on different datasets*

Based on this comparison, we will finally provide a recap table of each clustering method about their parameters, their best use cases and their limits.

Method	Parameters	Use case	Limit
K-Means Clustering	- number of cluster - centroid initialization method	- general-purposed - even cluster size - flat geometry - well separated	- requires to specify number of cluster - not scalable to dataset with many clusters - sensible to noise
Agglomerative Clustering	- number of cluster - linkage criterion	- uneven cluster size - eventually non-flat geometry	- no backtracking - high complexity makes it not scalable on large dataset - still sensible to noise despite different linkage criteria
DBSCAN	- neighborhood size - minimal samples in a cluster	- non-flat geometry - uneven cluster size - contains noise	- hard to optimally configure parameters - struggle with variable density dataset
HDBSCAN	- minimal cluster size - minimal samples in a neighborhood	- non-flat geometry - uneven cluster size - contains noise - variable density	- hard to optimally configure parameters

Table 1. *Recapitulative table of four clustering method*

IV. Conclusion

Through this practical work, we have discovered four different clustering methods belonging to four different categories, more precisely :

- K-Means Clustering which is the most general-purpose method that represents the Centroid-based Clustering family.
- Agglomerative Clustering as the bottom-up strategy of Hierarchical Clustering, known for its versatility, especially on non-convex data.
- DBSCAN - representing the Density-based Clustering category - as a robust technique against dataset with noise.
- HDBSCAN - a hybrid clustering method, upgraded version of DBSCAN which corrects the latter's weakness facing variable density dataset.

We also learned the importance of unsupervised machine learning in general and clustering in particular, where the task is to find a reasonable structure of a dataset that is not yet well-understood. For this end, not only we have discovered the panorama of different clustering method, but also the way data scientists are trying to bring more innovative approaches that improve the quality compared to their predecessors.

Nevertheless, we have only experimented on artificial 2D datasets. That is why, our next challenge will be to go deeper on this field in order to know how to apply these technique onto real-life, multidimensional data.