1. Problem definition and introduction

Clustering algorithms provide an automated way of classifying numerous records by assigning labels based on proximity to other records. Before the invention of DBSCAN there were multiple methods used to label data but none of them offered a solution that could discover clusters of arbitrary shape with minimal domain knowledge prior to the execution of the algorithm and that would have a good efficiency on large databases. Achieving all those needs at once was the motivation behind creating DBSCAN.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an algorithm used for identification of clusters in a multi-dimensional dataset. The grouping is done by measuring the density of records. A cluster has two main parameters, Eps and MinPts. Eps describes a neighborhood distance from a point whereas MinPts provides the minimal number of points that must be included in a neighborhood for a point to be considered a core point. Core points that are density-connected create a cluster. A point belonging to a neighborhood of a core point but not being a core point itself is considered a border point. If a point has neither enough other points inside of its neighborhood nor does it belong to any of core point's neighborhoods, it is labeled as noise. The algorithm goes through the dataset and checks as which of these three groups should any point be labeled. If a point is a border or a core point it is also assigned to specific cluster.

2. Description of the proposed algorithm/solution with reference to the literature

The solution provided in this project is based on one paper titled "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise" by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu [1].

The DBSCAN implementation starts with an arbitrary point and finds all points located not further than length Eps away from it. If the starting point fulfills the requirements of a core point a cluster is created. All other neighboring points are checked, and analogous tasks are done to complete the whole cluster. After no density-reachable points are available, DBSCAN picks the next point from the dataset and continues to try and create new clusters. If two clusters have points close enough to each other so that the shortest distance between any two points of those clusters is less than Eps they will be considered one cluster by the algorithm.

To determine the input parameters, we can use simple heuristics. For MinPts it is suggested that we chose a value two time greater than the number of dimensions in the dataset. To find adequate value for Eps we can look for the elbow point on a plot showing the distances in descending order of all points to their k-th nearest neighbors, where $k = 2 \cdot MinPts$.

3. Description of the implementation

The following code block shows the pseudocode for the proposed DBSCAN implementation.

```
function DBSCAN (set_of_points, eps, min_pts):

  cluster_id = 0
  for each point in set_of_points:
  point.was_visited = False

  for each point in set_of_points:
  if point.was_visited == True:
  continue to the next point

  point.was_visited = True:
  neigboring_points = RegionQuery(point, eps)

  if size(neigboring_points) < min_pts:
  //cluster with ID of -1 describes a noise point
  point.cluster = -1

  else:
  cluster_id = cluster_id + 1
  ClusterPoints(cluster_id, neigboring_points, point, eps, min_pts)

  return sets of points with same cluster_id

function RegionQuery(point, eps):

  return the set of points neighboring the input point

function ClusterPoints(cluster_id, neigboring_points, point, eps, min_pts):

  point.cluster = cluster_id

  for each neigboring_point in neigboring_points:
  if neigboring_point.was_visited == False:
  neigboring_point.was_visited = True
  new_neigboring_points = RegionQuery(neigboring_point, eps)

  if size(new_neigboring_points) >= min_pts:
  for each new_neigboring_point in new_neigboring_points:
  add new_neigboring_point to neigboring_points

  //cluster with ID of 0 describes a not assigned point
  if neigboring_point.cluster == 0:
  neigboring_point.cluster = cluster_id
```

The code is going to be written in Python. Points are going to be implemented as a class. They will store information on which cluster do they belong to, wether were they already visited by the algorithm and an ID number based on the location in the dataset. The RegionQuery function will use the triangle inequality propetry and a reference point for distance approximation to lower the time needed for finding the points belonging to a neigborhood. The

results will be returned as a dictionary of groups where the key will indicate the group ID and the value will contain an array of ID numbers of points belonging to that group.

4. User's manual

Project consists of two python files and was prepared to load a two dimensional data from a .csv file.

First *calculate-eps.py* should be run. It takes as input two command line arguments, first argument should contain the path to the data and the second argument the value k, by the authors of the DBSCAN paper suggested as $k = 2 \cdot MinPts$.
Sample command executing the calculate-eps.py script:

```
python calculate-eps.py data1.csv 4
```

The result is displayed as a plot, allowing us to approximate the value of eps by identifying the elbow point.

Having all input parameters we can run *the dbscan.py* script. This script takes three command line arguments, path to the data, eps value and $MinPts$ value.
Sample command executing the dbscan.py script:

```
python dbscan.py data1.csv 0.6 4
```

Script displays time taken for two approaches to searching for points belonging to neighborhoods. First, for every point checks the distance to every other point and the second one makes use of the triangle inequality theorem. Script also calculates the Silhouette core and Davies-Bouldin index and shows a plot containing all data points colored according to the cluster they belong to.

5. Description of used datasets

Artificial datasets were downloaded from [2]. Dataset were chosen to test the algorithm against different types of clustering to see on which does DBSCAN preform the best. Exact datasets are going to be shown in the next chapter together with results.

6. Experimental results

6.1. Dataset 1

First dataset is shown on figure 1. It consist of 15 circular clusters, that together resemble shape of three circles of differing sizes that share the center point. It could be assumed that DBSCAN will group the points well because of consistent densities of the clusters and the fact that they all are disconnected from each other.

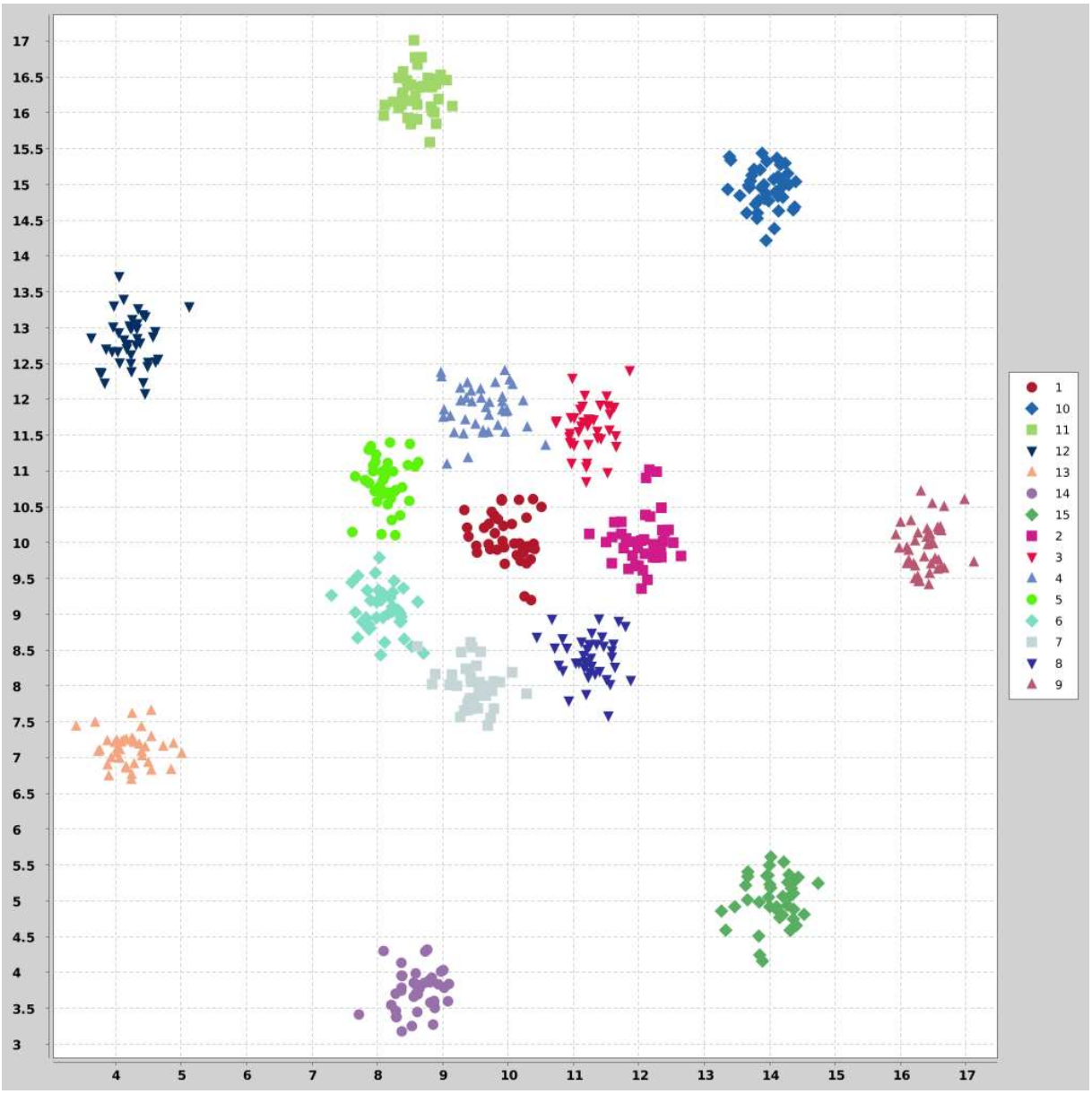

Figure 1. Dataset containing 15 disconnected clusters with comparable densities.

calculate-eps.py was run with the k parameter set to 4. Resulting graph is displayed in figure 2.

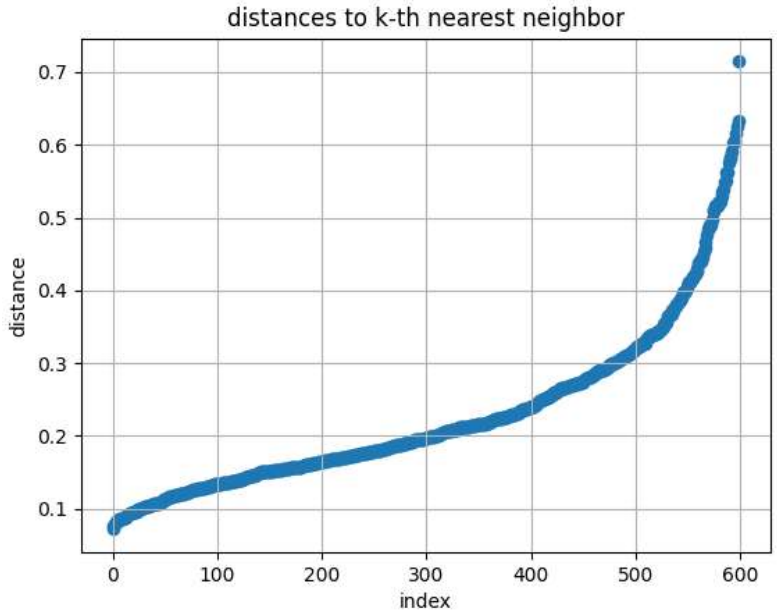

Figure 2. Distance to 4-th neighbor from each point of the dataset.

The value of *eps* was approximated to 0.35.

Figure 3 shows the graph with points assigned to clusters by the script.

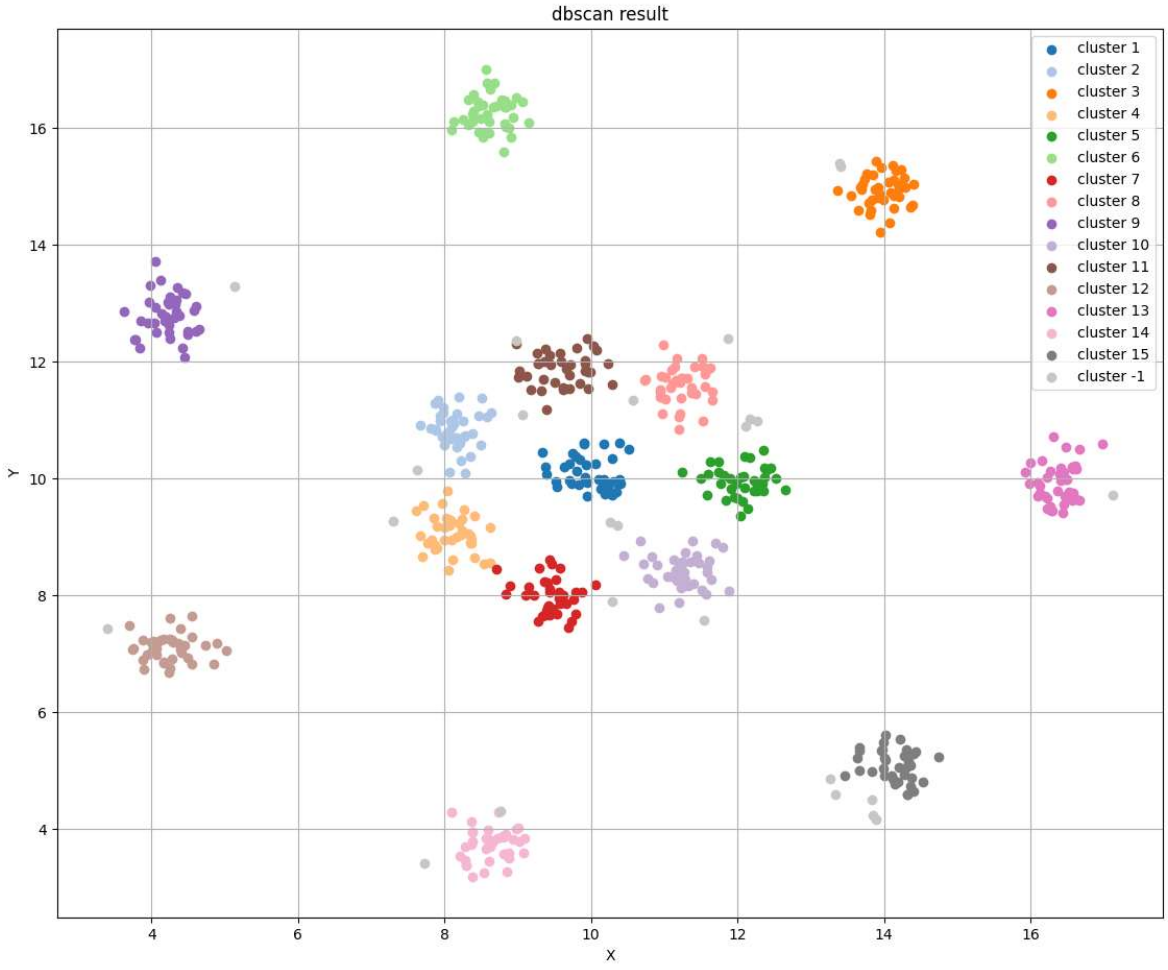

Figure 3. Graph with points assigned by the script.

It took 0.326 s for the brute force approach to finish calculations and 0.0465 s for the triangle inequality approach. Silhouette score of this assignment equals 0.791 and Davies-Bouldin index equals 0.399.

6.2. Dataset 2

Second dataset, shown on figure 4 consists of two c-shaped disconnected clusters with somewhat consistent densities. DBSCAN should work well on this set.

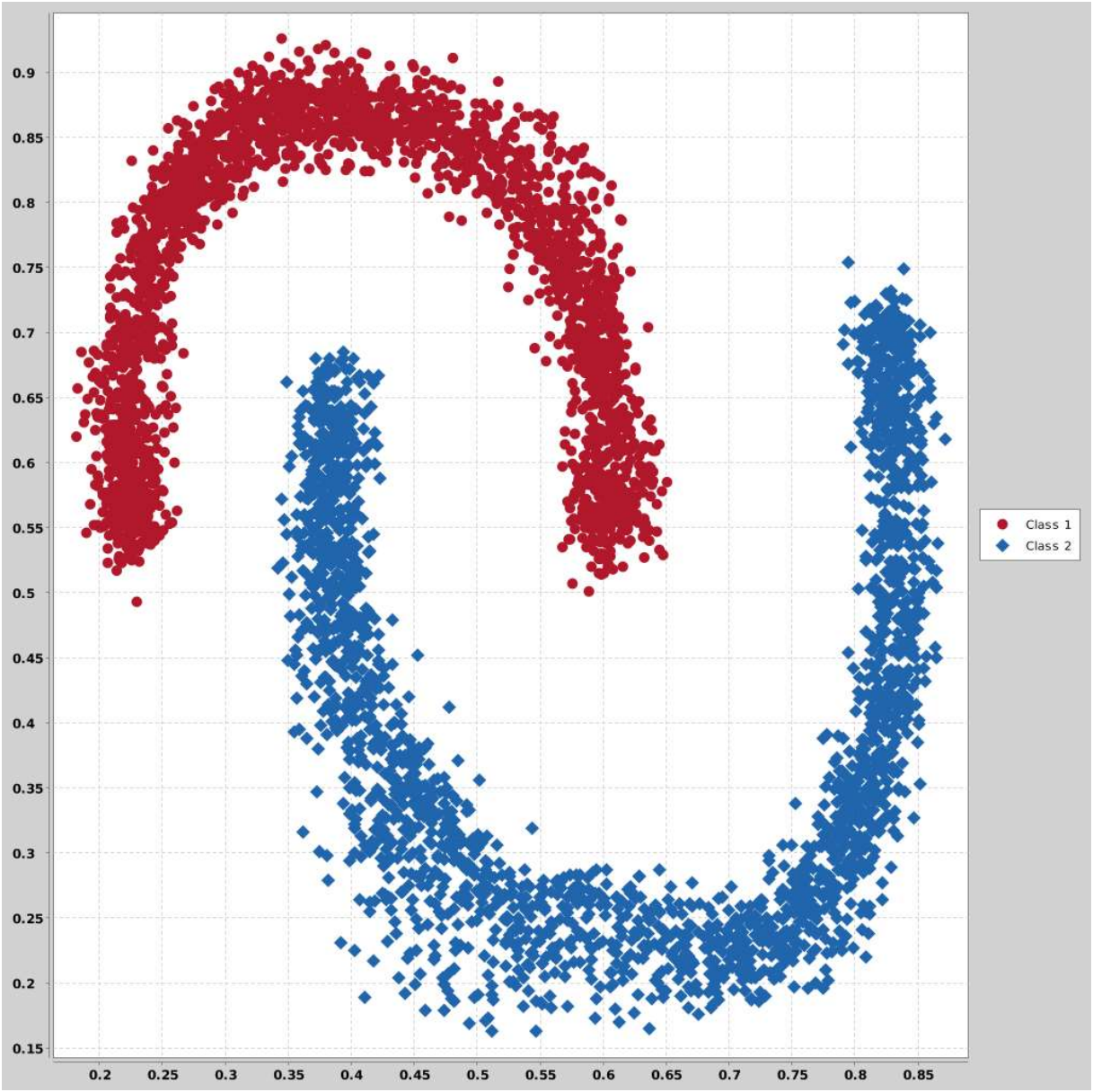

Figure 4. Dataset 2. Two disconnected c-shaped clusters.

calculate-eps.py was run with the k parameter set to 4. Resulting graph is displayed in figure 5.

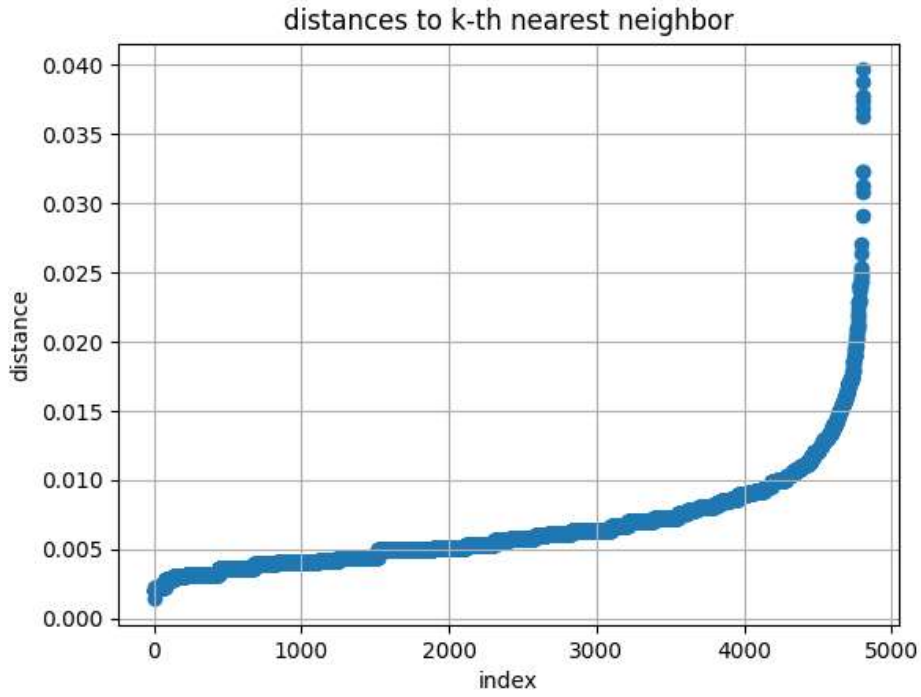

Figure 5. Distance to 4-th neighbor from each point of the dataset.

The value of eps was approximated to 0.013.

Figure 6 shows the graph with points assigned to clusters by the script.

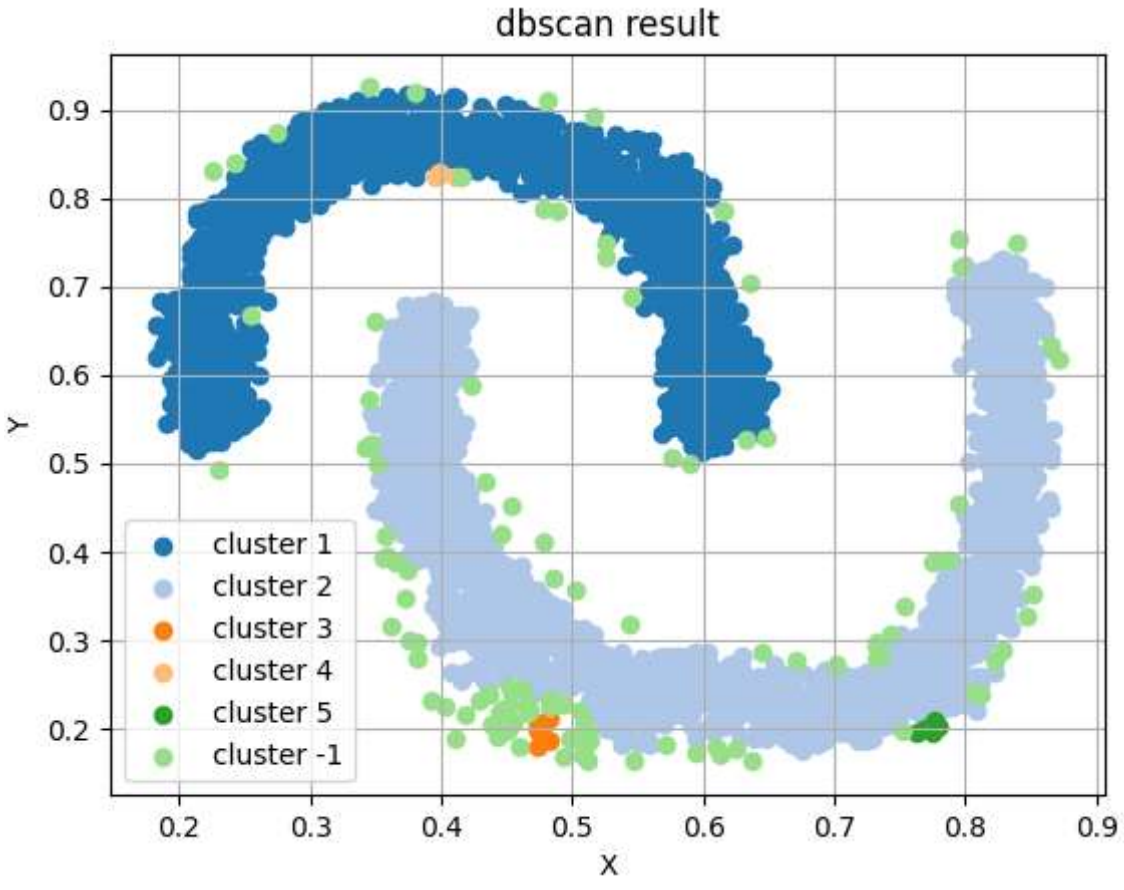

Figure 6. Graph with points assigned by the script.

It took 21.113 s for the brute force approach to finish calculations and 1.499 s for the triangle inequality approach. Silhouette score of this assignment equals -0.149 and Davies-Bouldin index equals 1.670.

6.3. Dataset 3

Third dataset shows clusters of differing density that are intersecting each other. It can be suspected that DBSCAN will not do well on this example. The dataset is shown in figure 7.

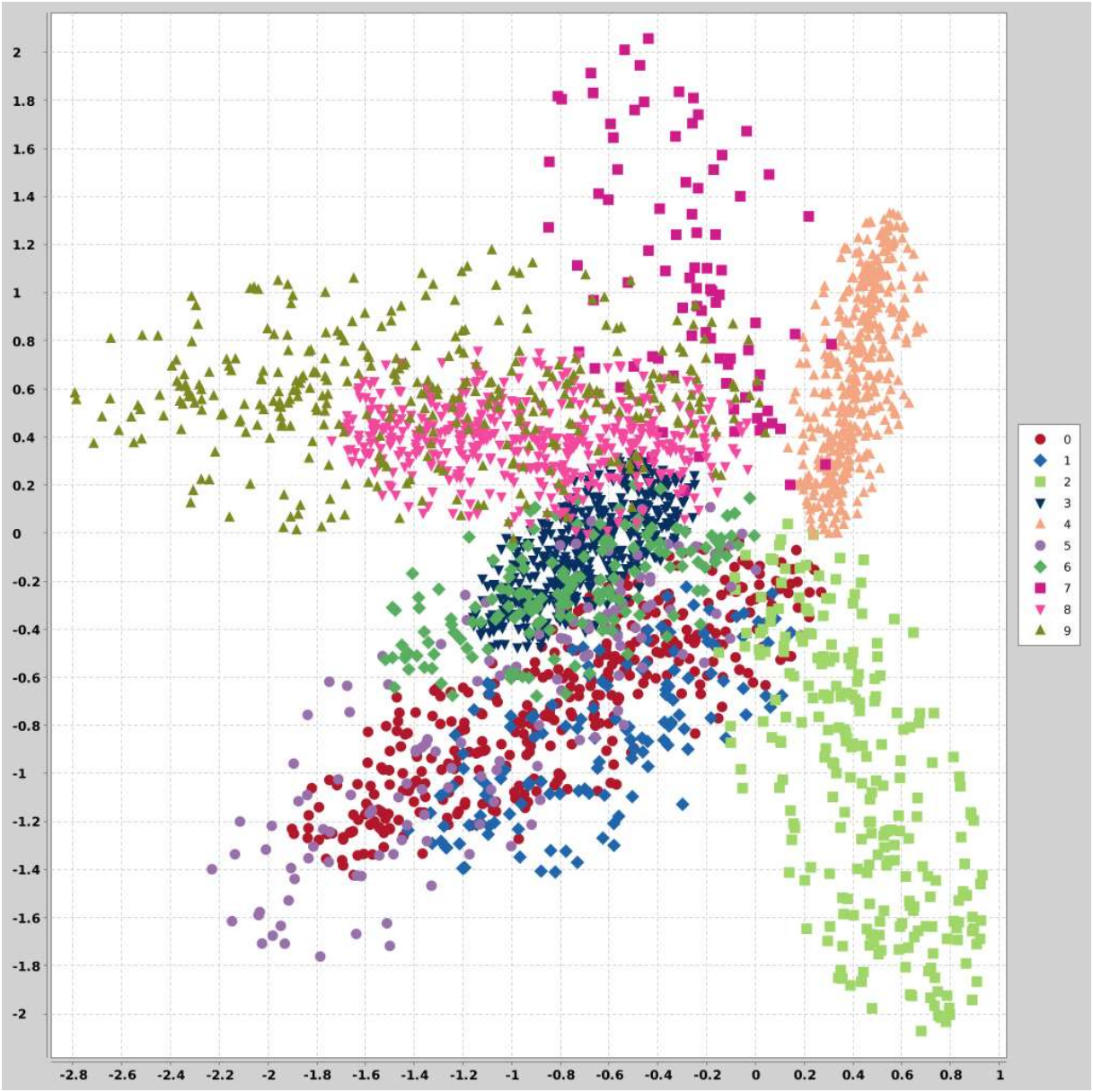

Figure 7. Dataset 3. Multiple intertwined clusters with differing densities.

calculate-eps.py was run with the k parameter set to 4. Resulting graph is displayed as figure 8.

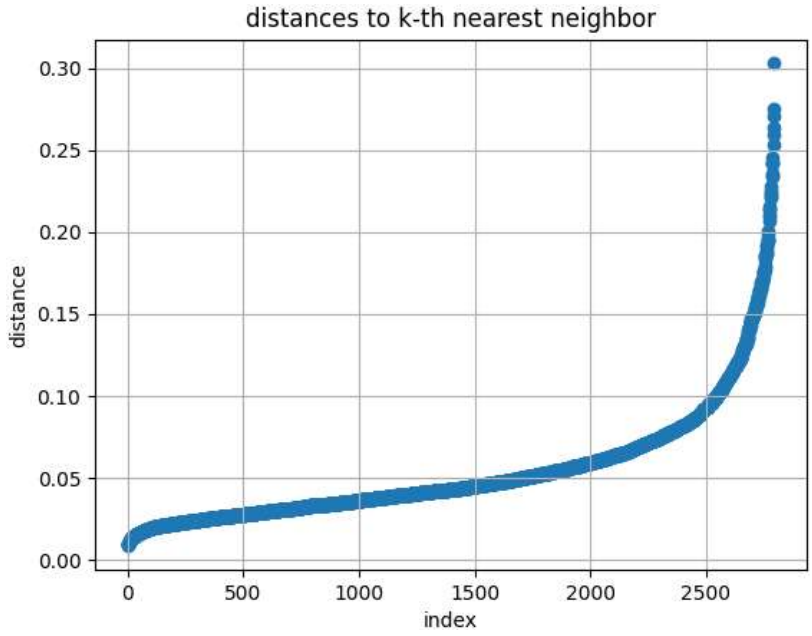

Figure 8. Distance to 4-th neighbor from each point of the dataset.

The value of *eps* was approximated to 0.10.

Figure 9 shows the graph with points assigned to clusters by the script.

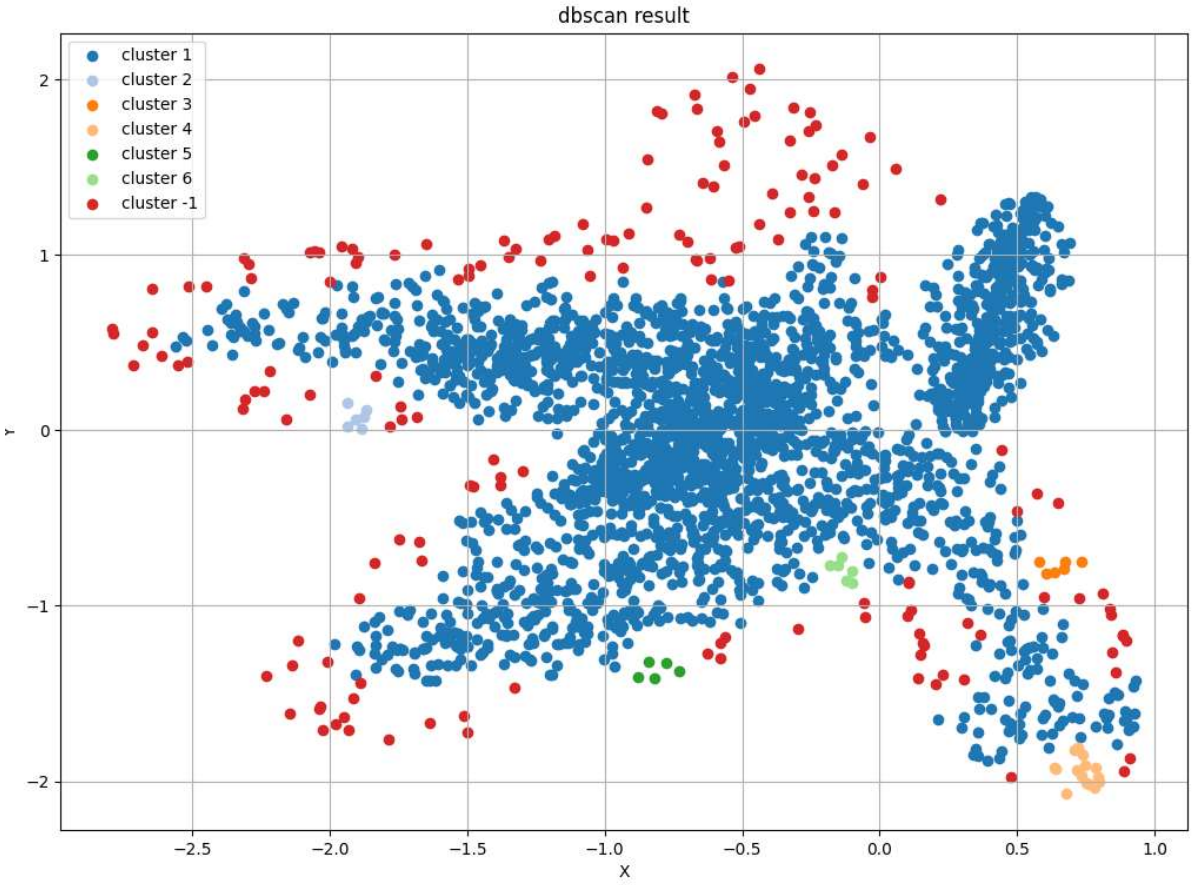

Figure 9. Graph with points assigned by the script.

It took 7.151 s for the brute force approach to finish calculations and 0.985 s for the triangle inequality approach. Silhouette score of this assignment equals 0.001 and Davies-Bouldin index equals 0.989.

7. Conclusions

Dataset 1 posed no challenge for the DBSCAN algorithm and the script managed to cluster the points well. Both Silhouette score and Davies-Bouldin index have values that can be considered good. DBSCAN found few noise points that weren't present in the original dataset, but those discrepancies do not seem to be of high importance.

Dataset 2 could be considered easy for the DBSCAN algorithm to cluster, yet the script did poorly. Noise points found by the algorithm should be instead added to the clusters they are close to. The reason why the script made this assignment most likely lies in the wrong approximation of the eps value. After few iterations with different values for eps better result ought to be found. Both evaluation scores rated this attempt to cluster points as unsatisfactory.

Dataset 3 was meant to be difficult for DBSCAN to assign properly. This set was chosen to see how will it score with unfavorable data. The algorithm found less clusters than it should and completely missed every of them. Due to the fact that basically all clusters intersect instead of dividing them DBSCAN just merged them together completely failing the task. Even though the evaluation scores are more favorable than those given to the evaluation of dataset 2, it's clear that DBSCAN does not work well for this kind of clustering. Most likely a different clustering approach should be taken to asses this dataset better.

DBSCAN work well for clusters with consistent density that are disconnected from each other. Disconnected clusters with diverse density might be still assessed correctly if the distances between clusters are considerable by trying different values for the eps parameter. Intersecting clusters are most likely impossible to be assessed well by the DBSCAN algorithm due to the mechanism it uses to find new clusters.

8. Bibliography

[1] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96).

[2] https://github.com/deric/clustering-benchmark