Using densities to detect nested clusters

RUSHIKESH SANE

Master's Thesis

UNIVERSITY OF
EASTERN FINLAND

School of Computing

Computer Science

April 2020

UNIVERSITY OF EASTERN FINLAND, Faculty of Science and Forestry, Joensuu
School of Computing
Computer Science

Rushikesh Sane: Density-based clustering revisited
Master's Thesis,
Supervisor of the master's thesis: Professor Pasi Fränti
April 2020

Abstract: Clustering is one of the most widely used applications in machine learning. It finds use in applications like recommendation systems and fraud detection systems. Similar objects are grouped together to form a cluster. There are situations where there is a need to find a nested cluster, which means finding cluster contained inside another cluster. Results of a density-based algorithm, capable of detecting nested clusters are studied in this thesis. This is a centroid based algorithm. It is built on top of k-means algorithm. The concept of weighted centroid is introduced. The centroid of each cluster is weighted between 0 and 1. Centroid weights are calculated from the density of their clusters. Weight of a centroid is directly proportional to the density of its cluster. Centroids with higher weights have denser clusters. They attract points from a lesser circumference and centroids with lesser weights attract points from a higher circumference. In this way, smaller cluster inside a bigger cluster can be detected. Once the clusters are formed, their densities are calculated again and from the density, weight of every centroid is calculated. Since the higher weight centroids attract only nearby points, it can extract nested clusters present inside a bigger cluster. In k-means, the number of clusters must be given as input and the algorithm returns centroids and partitions. Similarly, this density-based algorithm requires the number of clusters in advance and the clustering solution is returned in the form of centroid weights, centroid locations, and partitions.

Keywords: clustering, density, density-based clustering methods, weighted distance

Acknowledgement

I would like to extend my gratitude towards the University of Eastern Finland and its faculty members for allowing me to study in a very professional and studious place without worrying about tuition fees and the cost of living. I will always carry with me, all the things I learnt during my sojourn in Joensuu. I am more than sure that they helped me foster not only in my career, but also helped me in shaping my personality.

I would like to extend special thanks to Professor Pasi Fränti for guiding me throughout this project and keeping me motivated to produce satisfactory results. Apart from that, I learnt strict professionalism and got to see much passion towards profession from him.

Last but not the least, I would like to thank my parents and remember my grandparents because of whose blessings I believe, I got the opportunity to study in Finland.

Contents

Table of Contents

List of abbreviations

CI	Centroid index, a cluster-level measure of clustering quality
CSI	Centroid similarity index, a point-level measure of clustering quality
DBSCAN	Density based spatial clustering of applications with noise
GT	Ground truth
KDE	Kernel density estimation
K-means	*k*-means, a clustering algorithm
RS	Random swap, a clustering algorithm
SSE	Sum of squared errors

List of notations

N	Number of objects in a data set
n_i	Number of objects in the partition p_i
K	Number of clusters in a data set
D	Number of attributes in a data set
X	Set of N data vectors $\{x_1, x_2, \dots, x_N\}$
C	Set of K cluster centroids $\{c_1, c_2, \dots, c_K\}$
P	Set of N cluster partitions $\{p_1, p_2, \dots, p_N\}$
W	Set of centroid weights $\{w_1, w_2, \dots, w_K\}$
(i)	Density of the cluster i represented by the centroid c_i
(p, q)	Distance between p and q

1. Introduction

Clustering is one of the most natural and intuitive qualities of a human brain which helps us to distinguish between different objects at various levels. It is difficult for humans to process and categorize everything into a separate category. Thus, humans tend to categorize similar objects into *clusters* where each cluster is characterized by common features of the objects it has. According to [1], clustering is an *unsupervised* machine learning algorithm which aims at grouping similar objects together. Each object is represented as a data vector in clustering. A data vector has all the attributes of the object represented by it. Attributes are the adjectives that describe the object. It is a piece of information that is useful in clustering data. Attributes can be numeric or non-numeric. For example, in a dataset of movies, a movie is an object represented by a data vector and attributes of the movie like title, actor, genre and the release date are its features. Number of dimensions of a data vector is the number of its features.

The group of similar objects is called a cluster and all objects in one cluster are represented by a *prototype*. The entire cluster is represented by its prototype. The definition of a prototype depends on the nature of the data. Centroid and medoid are used as prototypes commonly and we are using centroid as a prototype in this thesis. These prototypes are calculated using distance measures like *Euclidean distance*. The Euclidean distance is easily extended to multidimensional data.

Classic textbook algorithm called *k-means* is used for clustering the data, but it does not cluster data vectors based on their densities. K-means also cannot detect nested clusters. We propose a unique density-based clustering approach to solve this shortcoming of k-means. This algorithm is based on *random swap* algorithm [2], which is a modified version of k-means. More information about random swap and k-means is found in the later part of the thesis.

1.1 Related work

This is the continued work of Jarkko Piiroinen's thesis [3] related to density-based clustering using weighted centroids. In his version, he has made some conclusions which are mentioned in this thesis too and the experiments have been continued assuming those conclusions to be true. Since this algorithm is based on top of k-means and random swap, it comes with all the pros and cons of k-means and random swap. DBSCAN [4] is a popular density-based clustering algorithm. There have been many attempts to modify DBSCAN to enable it to detect embedded and nested clusters. One of the variants is called Entropy based DBSCAN (EnDBSCAN) [10], which divides data into blocks and divides them into different computer nodes, to make full use of the data nodes. There is another variant of DBSCAN as described in [5] which tries to detect nested adjacent clusters to get rid of problems in DBSCAN and its variants.

1.2 Objectives of thesis and problem definition

This thesis proposes a unique density-based algorithm which can detect nested clusters. It is a centroid based algorithm like k-means. It finds densities and has a prototype associated with

every cluster. Like any clustering algorithm, it is assumed that the data is spread across several clusters and interesting patterns are to be found. The problem of clustering using weighted centroids is given as:

> Given a set of data X, number of clusters K and centroid weights W, find locations of the centroids, and partitioning P such that an objective function is minimized.

1.3 Thesis structure

In Chapter 2, the background and details about clustering and concepts used in clustering are explained in simple language. Different methods of clustering, distances used in clustering and other technicalities related to clustering are explained. In Chapter 3, prerequisite algorithms and concepts are explained. K-means and random swap are the basics to understand the work in this thesis. Chapter 4 explains three different density-based algorithms which already exist to start the process of understanding newly proposed density-based algorithm. Chapter 5 has explanation about the concept of weighted centroids and weighted distances. These concepts are used in the main algorithm proposed in this thesis. Chapter 6 presents the algorithm step by step followed by an example to show how the algorithm works. The experiments conducted and their results are drafted in Chapter 7. There are cluster validation measures which are worth understanding and are explained in this chapter. Detailed analysis about the results, comparison with other existing algorithms is also given. Chapter 8 has conclusion and the potential future work.

2. Clustering

2.1 Basic concepts

Clustering algorithms are used to group similar objects together based on their distance functions. In each dataset, we may have many objects, some being like each other and some being different. This similarity or difference is determined by studying features of the objects in the dataset. Features are the properties which uniquely find an object in a dataset. For example, let us consider the dataset represented in Figure 1.

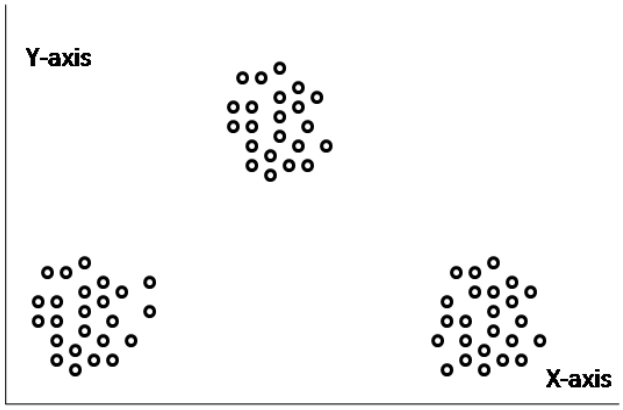


Figure 1: Sample dataset to understand clustering

In Figure 1, we have sample data vectors spread along the attributes shown as x-axis and y-axis. Hence x-axis and y-axis are the features of these data vectors as each data vector can be uniquely found with the combination of its features that is, x and y co-ordinates. In real-life, these features can be any tangible property of an object. If the spread of data vectors is observed, it can be seen that there are a few data vectors concentrated in the left part of the figure, few data vectors concentrated in the right part of the figure and few in the middle top. Four points are picked from the dataset to study their similarities and differences from each other.

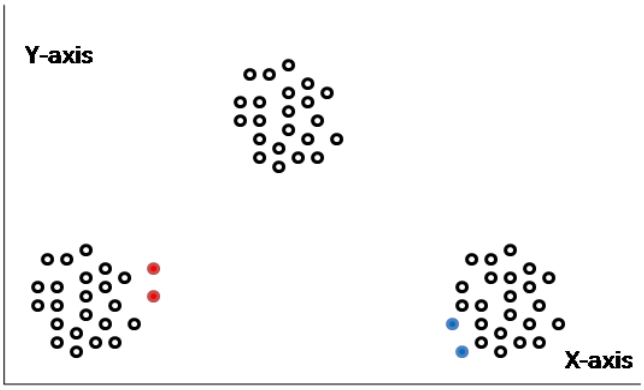


Figure 2: Select two sample points from each region

It can be visually seen that red points are more like each other than with blue points. Similarly, blue points are more like each other than with red points. This similarity is shown because the Euclidean distance between the features of red points is lesser than the Euclidean distance between features of red and blue points. Here the features are x and y coordinates and the difference between the features is the Euclidean distance between the data points. The features vary according to the nature of the dataset.

An example which everyone experiences in daily life is that of online shopping. When one buys a phone, one gets recommendations from the online shopping portal with the message of a form, "*People who bought this phone also bought this mobile cover and these headphones*." Here, the similar objects are grouped together by detecting similar features and are then recommended by the online shopping portal using the underlying clustering algorithm. In the same way, if the data set in the Figure 2 is clustered, we will get to see clear distinctions as shown in Figure 3.

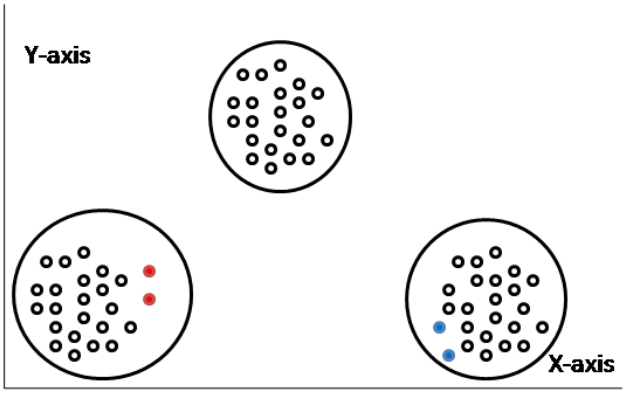


Figure 3: Clustered data set

These distinctions are called clusters and the dataset is said to be clustered. Since all the data vectors in each cluster are like each other, they are represented by a single vector called *prototype*. Centroid is the most commonly used prototype in clustering. Centroid is used as a prototype in centroid based clustering algorithms. Centroid is a vector which has all attributes of the vectors in the dataset. The value of centroid attributes is the mean value of the attributes of all the other vectors in that cluster. Prototyping can make it easier to visualize large datasets with millions of data vectors and thousands of clusters. It reduces the data vectors just to the number of clusters present in the given dataset. Other prototypes like *medoid* also exist. Medoids represent cluster with a vector whose average dissimilarity to all vectors in the cluster is minimal. Unlike centroids, medoids always belong to the dataset. Centroids for the dataset in Figure 3 are shown in Figure 4.

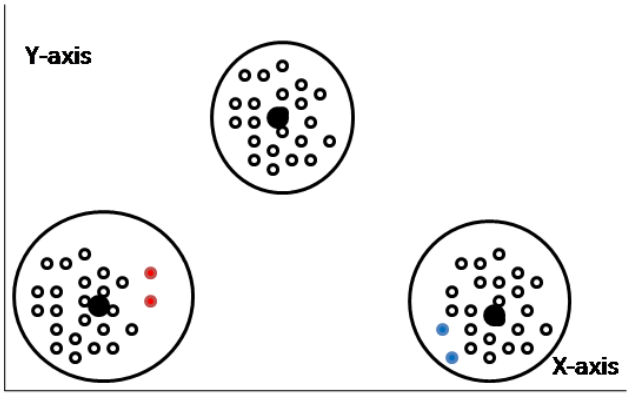
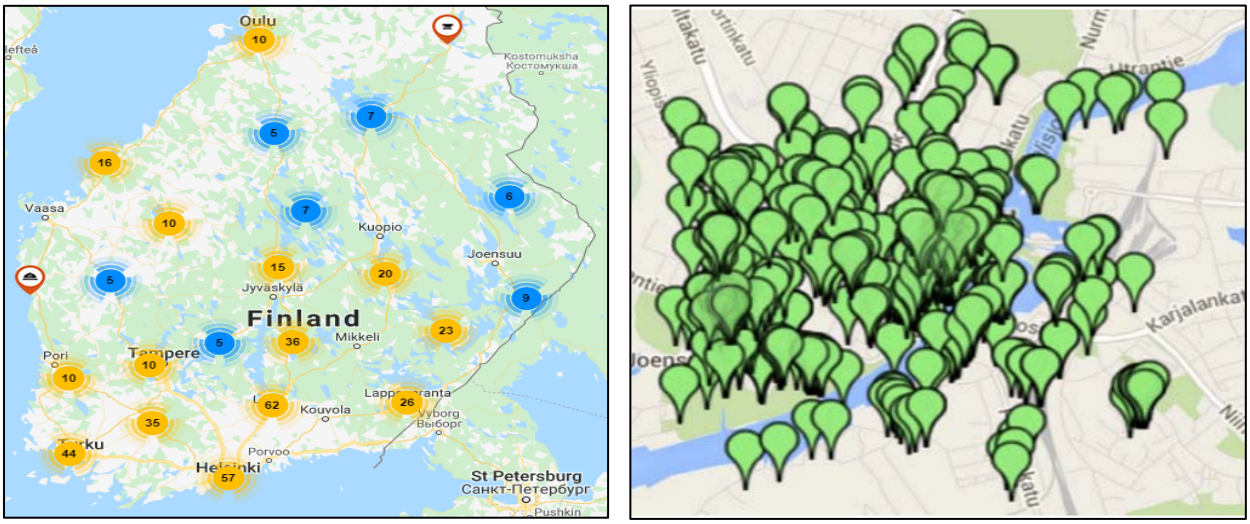


Figure 4: Clusters with centroids

The main advantage of using centroids (or any prototype) is reduction in the number of vectors to study. Figure 5 has a real-life example which shows the importance of having clustered data. Crude observations, statistical analysis and other scientific inferences can be easily and efficiently studied. In many cases, it also improves the visual representation.



a. Map with cluster[1] b. Map without cluster[2]

Figure 5: Maps with and without clustering

Figure 5a is a map with lots of places, represented by a cluster. Here, the places near each other are shown as one single object (represented by a cluster). On the contrary, Figure 5b is a map without clustering. Clustering places helps the visuals to remain clean and clearly visible without taking up much space. It is visually difficult and cumbersome to find places of interest in the Figure 5b. It also has other disadvantages like cluttering other pointers (overlapping) and increased time to load all the data points on the map

[1] https://way.fi/haku?category=ravintola&map=y
[2] http://cs.uef.fi/sipu/pub/aece_2018_4_8.pdf

2.2 Distances

There are many types of distances used in clustering. Some of the common distances are *Euclidean, Manhattan* and *Cosine* distances. Clustering algorithms are based on distance between two vectors and hence it is necessary to select a distance measure while implementing any algorithm.

2.2.1 Euclidean distance

Euclidean distance is the normal straight-line distance between the given vectors which is calculated using following formula:

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} \quad (1)$$

Euclidean distance is symmetric. Variables p and q in the equation refer to data vectors and subscripts *1, 2…, n* refer to the dimensions in an n-dimensional dataset.

2.2.2 Squared Euclidean distance

Squared Euclidean distance is like Euclidean distance with the only exception that square root of the sum of differences of features is not taken. In other words, square of Euclidean distance is called squared Euclidean distance. The formula for squared Euclidean distance is given as follows:

$$d(p,q) = d(q,p) = (q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2 \quad (2)$$

2.2.3 Manhattan distance

It is defined as the distance between two vectors when travelled along the axes. Manhattan distance is preferred in using medoid based algorithms because it uses absolute values which are robust and can be easily generalized to higher dimensional datasets [6].

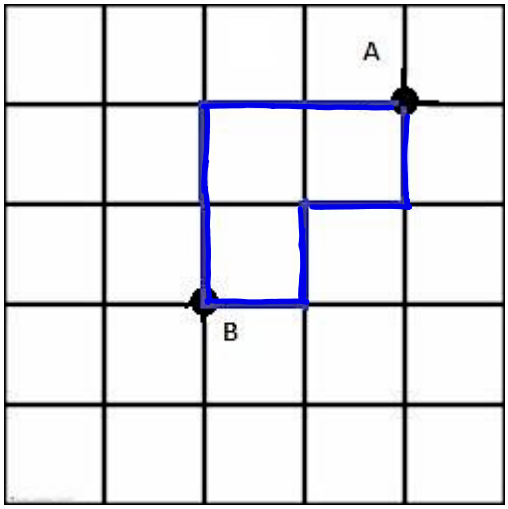


Figure 6: Manhattan distance

Manhattan distance between points A and B is along the axes as shown with blue lines. Manhattan distances are always the same irrespective of the direction of the path taken. Figure 6 has two options of using Manhattan distance between points *A* and *B*. Manhattan distance between two vectors is greater than Euclidean distance between the same two vectors. It is the sum of the lengths of projections of the segment between two vectors into co-ordinate axes. In simpler terms, it is the sum of horizontal and vertical components in a plane between given two vectors. The mathematical formula to calculate Manhattan distance is given as follows:

$$d(p,q) = \sum_{k=1}^{n} | p_{k-} q_k| \tag{3}$$

Where p and q are the points and k refer to the features of the given points.

2.2.4 Edit distance

Edit distance is used to find distance between two strings. It is the minimum number of modifications required to convert one string to other. The modification operations are insertion, deletion, and replacement. It is used in applications like automated spell checks. This distance measure, however, is very application specific. Text clustering is a favorable application to use edit distance as the distance measure. Example of edit distance calculation is as follows:

String 1: Sit

String 2: Sat

Edit distance between string 1 and string 2 is equal to one in this example because only one operation is needed convert string 1 to string 2. We get string 2 by replacing *i* in string 1 with *a*.

Other types of distances like *Cosine, Jaccard, Mahanolobis* are also used in some applications but the distances explained in this section are used most commonly. Since there are many different distance measures available, selecting the best distance measure is a bit confusing. The factors that need to be studied before choosing the best distance measure are application of the algorithm and the type of input data [7].

2.3 Clustering methods

Various clustering methods exist which use different implementation, strategies to detect clusters and objective functions. When clustering methods are changed, there is a change in cluster configuration and input parameters, which can change results too. The method of clustering is chosen based on the application and the type of input data. Every clustering method has its own advantages and disadvantages. Based on these constraints, a suitable clustering method is chosen. Different clustering methods are given in the following sub-sections.

2.3.1 Hierarchical clustering

Hierarchical clustering focuses on forming clusters by joining vectors together and forming one single cluster. Joining of vectors depends on the strategy used and there are many strategies to join vectors together in hierarchical clustering. Agglomerative and divisive clustering are the types of hierarchical clustering in which, agglomerative clustering uses bottom up approach to merge clusters while divisive method uses top down approach. The basic idea lies in the fact that close vectors should form a single cluster and well separated vectors should be classified into different clusters. In agglomerative approach, multiple small clusters are joined together to form a large cluster which again is joined with some other cluster to form even larger cluster. It forms hierarchy in clustering which is represented as a dendrogram. Dendrograms are a classic way to represent hierarchical clusters but they are not used much in modern data analysis or clustering. However, they clarify the concept of hierarchical clustering. The main advantage of hierarchical clustering is that it produces a nested tree of partitions and therefore is more informative than non-hierarchical clustering. Hierarchical clustering has a high computational cost [8].

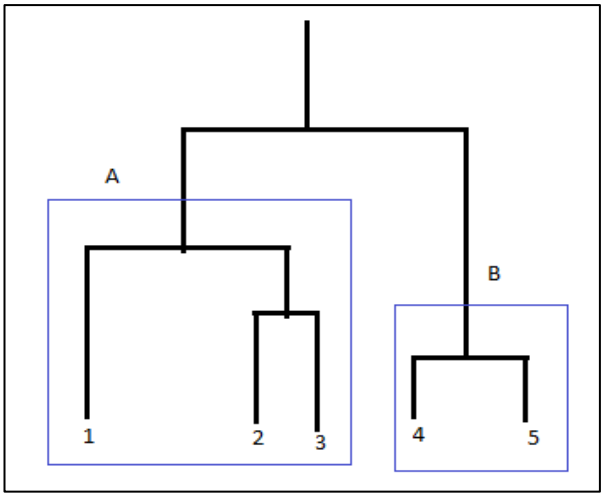


Figure 7: Dendrogram in hierarchical clustering

2.3.2 Centroid based clustering

Centroid based clustering revolves around the idea of having a centroid to represent every cluster. In crude terms, centroid is a vector having mean features of all the data vectors belonging to that cluster. In this method, the data is partitioned in such a way that every partition should have at least one data vector. K-means is a classic textbook algorithm which is centroid based. Centroid based method is a popular method in academics and research [9].

2.3.3 Distribution based clustering

Distribution based clustering is based on the idea of clustering data using the distribution models like Gaussian distribution. Data is viewed to be spread across, following a certain distribution model and clusters are found by trying to fit the given dataset to given distribution model. The downside of using distribution-based clustering is its problem of over fitting. They can capture other statistical properties like correlation and dependency.

2.3.4 Density based clustering

This thesis focuses on density-based clustering approach. Clusters are regions where many data vectors are accumulated, that is, there is a dense region of data vectors forming a cluster. Between the two clusters, there is a region of no or very few data vectors. In density-based clustering, high density regions are separated from each other by low density regions. DBSCAN is one of the most popular density-based clustering algorithms. There are many variants of DBSCAN like Fast DBSCAN [4], EnDBSCAN [10], adaptive density based spatial clustering [11]. Algorithm in this thesis is a density-based algorithm, and centroid based at the same time

2.3.5 Grid based clustering

It is a clustering approach in which the dataset is divided into finite space of grid and every grid is treated as a separate cluster in the beginning. The grids are later merged into a single clustering by testing grid data vectors for certain criteria. Grid growing algorithm [12] is an example of grid-based clustering in which two neighboring grids are merged into a single cluster if number of data vectors in the adjacent grids is comparable.

3. K-means and random swap

The thesis in this algorithm is based on random swap algorithm. Random swap algorithm is built on top of k-means algorithm. Hence k-means and random swap algorithm are the pre-requisites to understand density-based clustering algorithm studied in this thesis.

3.1 K-means algorithm

K-means is a classic textbook centroid based algorithm which is used to detect clusters in each dataset. Forming clusters by assigning a vector to the nearest centroid and calculating the centroids is the general idea in k-means. This is iterated until a good cluster is formed. Good cluster is formed when objective function reaches a certain value. Objective function acts as the stopping criteria in k-means. Details about objective functions appear in sub Section 3.1.4.

3.1.1 Cluster initialization

K-means requires us to select the value of *k* in advance. It is an input parameter to the k-means algorithm. Once the value of *k* is selected, *k* number of centroids are randomly placed across the dataset. This is called *centroid initialization*. There are many methods of centroid initialization [13] in clustering. Placing centroids randomly is one of them. This is also referred to as placing *seeds*. There is another cluster initialization technique called *RUNFP* [14] (it is referred to as *Maxmin* in [13]), in which *k* points are selected in such a way that they are farthest from each other. This is used in an algorithm called *k-means++* [17]. Another initialization technique called *GREP (group representative points)* [15] exists in which first centroid is taken to be the center of general data present and other centroids are placed by examining how the new center is closer to a data vector than existing centers in terms of Euclidean distance.

From [13] one main result is that no matter which initialization, k-means can improve it only if there is cluster overlap. Therefore, no matter which initialization technique is chosen, k-means may not work with data even if it is having clearly separated clusters. In case of nested k-means, overlap is expected but not guaranteed. Some methods are sensitive to dataset and may result in different results, or slower or quicker results. Their sensitivity to outliers and local optima also has an effect with the change in centroid initialization techniques [16]. In this thesis, centroids are initialized as random points across dataset. More advantages and scientific evidences of careful centroid initialization are found in [17].

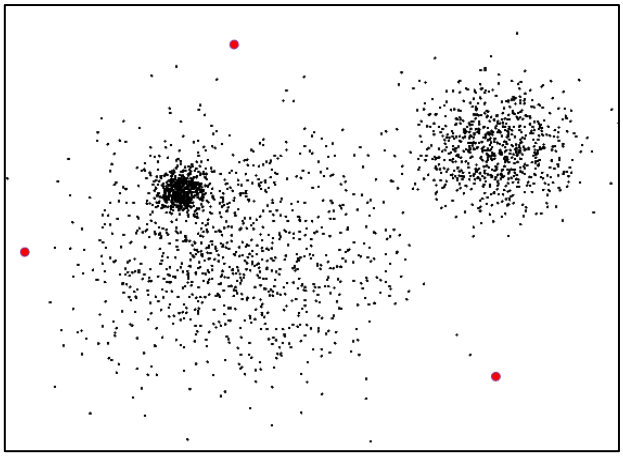


Figure 8: Randomly initialized centroids

A sample dataset is shown in the Figure 8 and red markers show randomly spread seeds. Visually, three clusters in the data set can be easily distinguished, but for the algorithm to go ahead, three centroids are randomly initialized.

3.1.2 Partitioning

Once the centroids are placed randomly, distance of every data vector is calculated from every centroid in the cluster. A data vector is taken at a time, its distance from all the centroids is calculated and the vector is placed in the cluster of the centroid which is nearest to that vector. This is the step where distance between vectors is calculated. In this thesis, squared Euclidean distance is used as the distance measure to calculate distance between any two vectors. Once all the data vectors are associated to some cluster, algorithm proceeds to the next step of centroid calculation. The algorithm for calculating nearest centroid is given in Figure 9.

```
GetNearestCentroid(x_i, C): returns nearest centroid C_nearest
    d_nearest = ∞;
    for j = 1 to K
        d = distance(x_i, C_k);
        if(d < d_nearest)
            C_nearest = C_k ;
            d_nearest = d ;
return C_nearest ;
```

Figure 9: Partitioning algorithm in k-means

From notations in Figure 9, x_i is a data vector and C has all the k centroids. This function is called for all the data points in a loop. At the end of this function, every data vector is assigned to a cluster. We say that the data is *partitioned* at the end of this function.

3.1.3 Calculating centroids

Once all the data vectors are associated with a cluster, the centroids are re-calculated. Initially, a centroid is randomly placed hence it is not necessarily the centroid of all data vectors it is closest to. After the data is partitioned, centroid is re-calculated by taking mean of all features of all data vectors belonging to that cluster. This process is repeated for every cluster and all the centroids are re-calculated. Calculating centroid in k-means involves the same process as required to calculate centroid for any other application. Following function is looped *k* times to calculate new centroid for every cluster.

```
CalculateCentroids(X, P, j): returns new centroids C_new
    sum = 0;
    count = 0;
    //calculate sum of all vectors in partition
    for i = 1 to N
        if p_i = j
            sum = sum + x_i
            count++;
    C_new = sum/count;
return C_new;
```

Figure 10: Algorithm to calculate centroids in k-means

Partition has all vectors in each partition as calculated by algorithm to get nearest centroids. Centroids are calculated by taking mean of all *k* features of all vectors in a partition. The vector centroids are returned by this algorithm.

3.1.4 Objective function

Grouping similar data helps in generalizing information about vectors or objects. Process of clustering groups similar data vectors into one cluster and separates those from the data vectors which are not similar. If this is achieved, good clustering solutions are achieved. K-means algorithm runs in iterations and every iteration produces a clustering solution. The solution produced in the current iteration is compared to the solution produced in the earlier iteration. New solution is accepted if it is better than earlier solution. This comparison of solutions in the current and earlier iterations is done by objective function. There are many objective functions available to evaluate clustering solutions, but the objective function used in this thesis is the *Sum of Squared Errors (SSE)*. The formula for SSE is given as below:

$$SSE = \sum_{i \epsilon k} \sum_{x \, \epsilon \, C_i} dist^2(C_i, x) \tag{4}$$

where, C_i is the centroid of cluster *i, x* is a data vector in that cluster and *k* is the total number of clusters. For better understanding, SSE can be broken down into three steps viz. errors, square and sum. Centroid of all the vectors in a cluster is calculated and the entire cluster is

denoted with that single centroid. Distance between centroid and any data vector in its cluster is called as error. Centroid is the mean of all vectors and hence every vector in the cluster has some distance (or error) from the vector. The error is measured as Euclidean distance. This distance or error when squared is squared error and the sum of squared errors of all data vectors is defined as SSE.

The distance between centroids of two clusters is the inter-cluster distance. Intra-cluster distance is measured in variety of ways like maximal distance between any two vectors in a cluster *k*. We try to achieve clusters such that inter cluster distances are maximized and intra cluster distances are minimized. There is a correlation between SSE and the number of clusters. SSE decreases if the number of clusters is increased. An objective function helps us in understanding the quality of a cluster in the k-means iterations and the value of objective function helps in estimating cluster compactness. Apart from SSE, there are many other objective functions like *Dunn index*, *graph cut-objective functions*, *ratio cut* and *WSS* [18]. The objective functions are maximized or minimized. In the case of SSE, the objective function is minimized. Selection of objective function is not a trivial job. It depends on factors like nature of the dataset and clustering algorithm used. Complete algorithm for k-means is given in Figure 11.

```
KMeans(X, P, C): returns clustered dataset
    InitializeCentroids();
    SSE_previous = SSE_current = ∞;
    repeat
        SSE_previous = SSE_current
        for i = 0 to N
            p_i = GetNearestCentroid(x_i, C);
        for j = 0 to K
            cj = CalculateCentroids(X, P, j);
        SSE_current = CalculateSSE();
    until(SSE_current < SSE_previous)
```

Figure 11: K-means algorithm

3.1.5 Limitations of k-means

Though k-means has advantages like simplicity to implement, it has some limitations too. One of the limitations is to know the number of clusters in the beginning. The user needs to predict and feed the value of *k* to the algorithm before running it. For example, the dataset in Figure 12 ideally has 15 clusters. That is, the value of *k* should be 15 to get correct results. But if the value of *k* is incorrectly supplied, the clusters are not ideal, and it is not the best solution. Figure 12 and Figure 13 show the difference in output of the same dataset when number of correct and incorrect *k* value is supplied. Figure 14 shows another incorrect clustering solution when number of clusters supplied as input value is lesser than optimum number of clusters in the dataset. A detailed analysis and explanation of these figures is provided in Section 3.2.

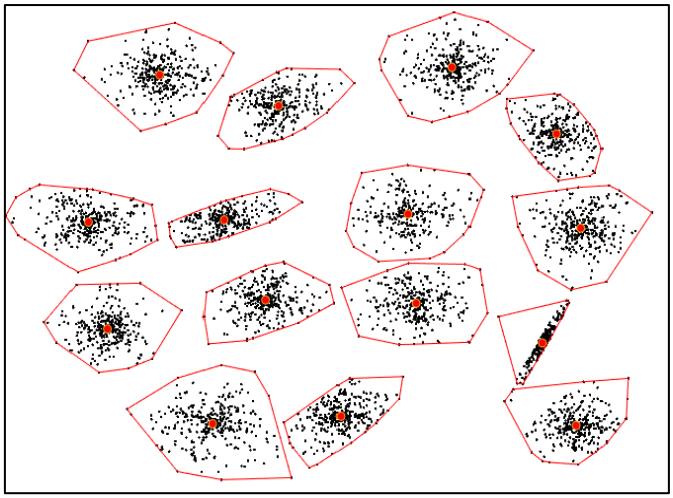


Figure 12: Correct clustering when correct *k* value is supplied

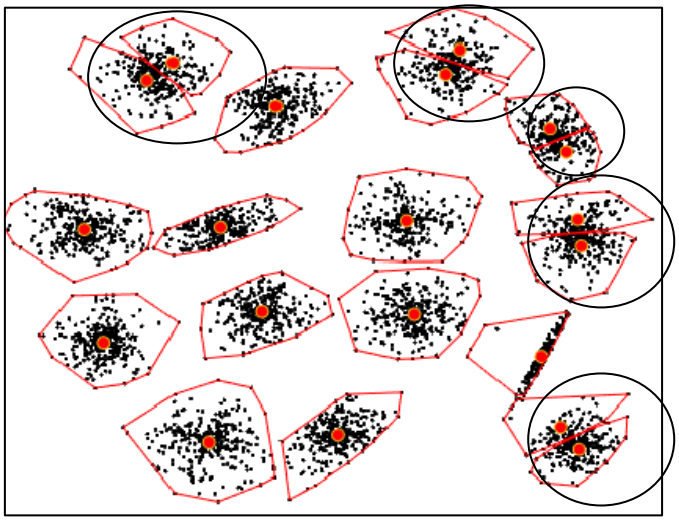


Figure 13: Clustering solution when *k* value is more than ideal *k* value

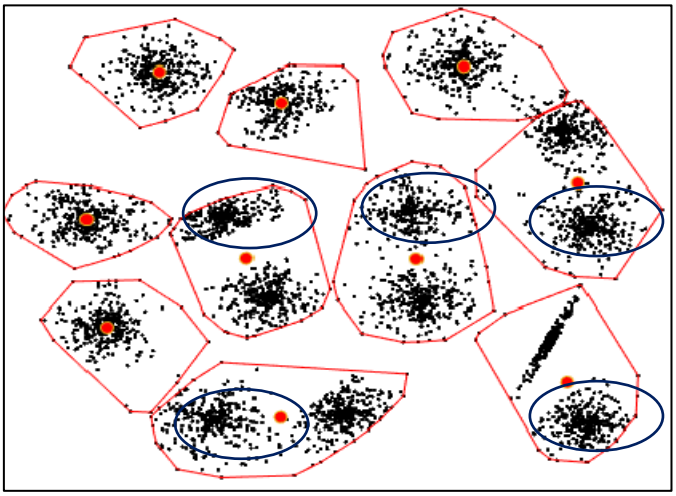


Figure 14: Clustering solution when *k* value is lesser than ideal k value

Another limitation of *k* means algorithm is its inability to detect the nested clusters. A smaller cluster inside a bigger cluster is a nested cluster. It is a concentration of few data vectors spread over a huge set of vectors.

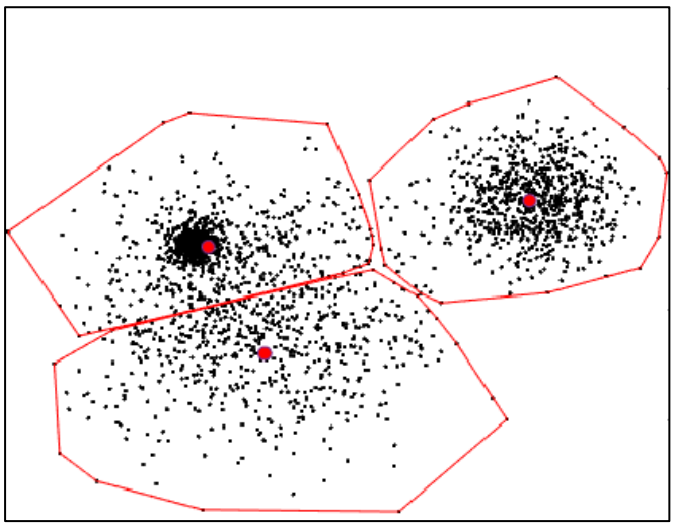


Figure 15: Clustering nested datasets using k-means

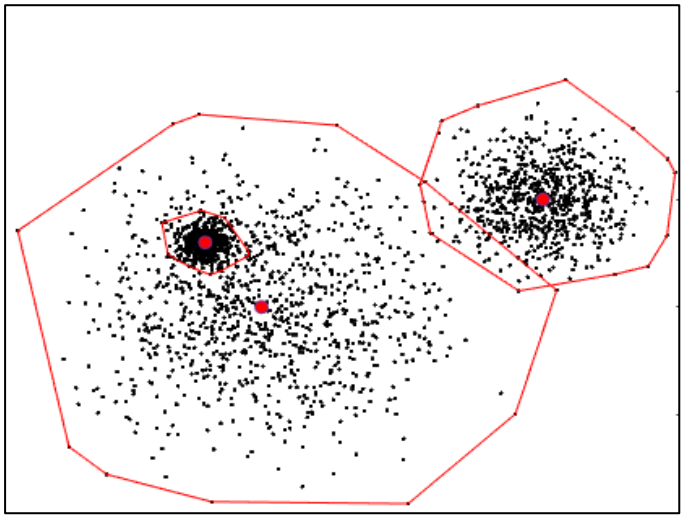


Figure 16: Detection of nested cluster

Figure 15 shows the clustering resulting of normal k-means algorithm. If the dataset is visually observed, there is a huge spread of vectors in the central part, separated by a gap with another set of data vectors in the right top corner. There is also a small concentration of vectors in the upper part of dataset. This dense spread of vectors appears as a nested cluster as visible in Figure 16. Visually, solution in Figure 16 is better than the solution in Figure 15. If the dataset does not have any nested cluster, density-based algorithm in this thesis will give the non-nested normal clustering solution as the output.

3.2 Random swap

Random swap algorithm was introduced by Pasi Fränti and J. Kivijärvi in [19] to improve clustering in unsupervised classification. This algorithm is based on k-means. The problem

definition of this algorithm is the same as k-means, where it takes *N* data points which must be partitioned into *k* clusters and use centroid as a prototype. The objective function used is SSE which is the same as used in k-means. K-means may not able to go ahead further if local minima are reached and it may stop. This leads to incorrect solution. To tackle this problem, random swap algorithm was developed.

Initially, a centroid is randomly chosen and swapped to a random location. K-means algorithm is then run after random swap for a pre-decided number of times. These steps are repeated for pre-decided number of times. If the new solution after swapping and k-means has lower SSE than the earlier iteration, then the new solution is accepted. Else some other centroid is randomly chosen and swapped again. The steps involved in random swap algorithm are:

1. Randomly swap a centroid
2. Run k-means

Algorithm for randomly swapping a centroid is given as follows:

```
SwapCentroid(X, C): returns a random centroid
    j = random(C);
    i = random(X);
    C_j = i;
return C;
```

Figure 17: Algorithm to randomly swap a centroid

The complete random swap algorithm is given in Figure 18.

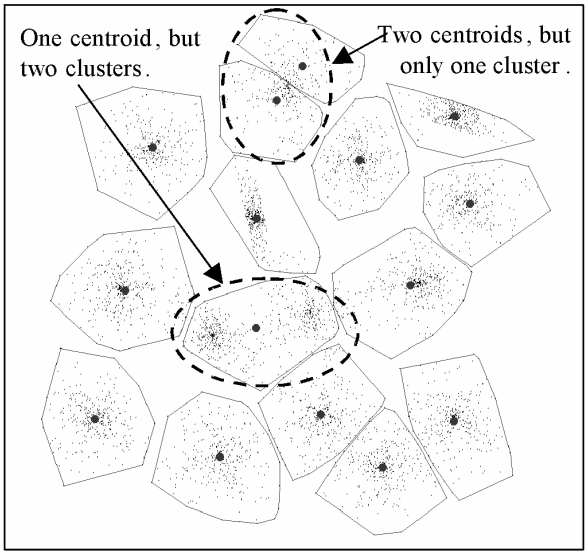
```
RandomSwap(X, C, P): updates C and P
    Repeat T1 times
        C_new = SwapCentroid(X, C);
        P_new = Partitioning(X, C);
        Reapeat T2 times
            (C_new, P_new) = KMeans(X, P_new, C_new);
        if(SSE(C_new, P_new) < SSE(C, P))
            (C, P) = (C_new, P_new);
```

Figure 18: Random swap algorithm

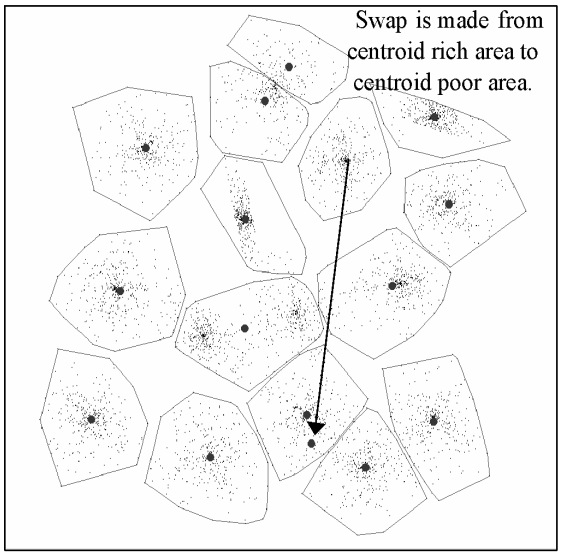
Number of iterations to be carried out in k-means is a free choice in random swap algorithm. It was seen in [19] that mere two iterations of k-means are enough in getting high quality clusters. When random swap happens and by luck if a centroid is moved from centroid rich area to centroid poor area, then clustering improvement will effectively take place and will yield better solution than the earlier iteration. There are three types of swaps in random swap namely *trial swap, accepted swa*p and *successful swap* according to [20]. Accepted swap is the

one in which SSE improves, successful swap is the one in a greater number of centroids are in correct locations as compared to earlier iteration and if neither improves, it is a trial swap.

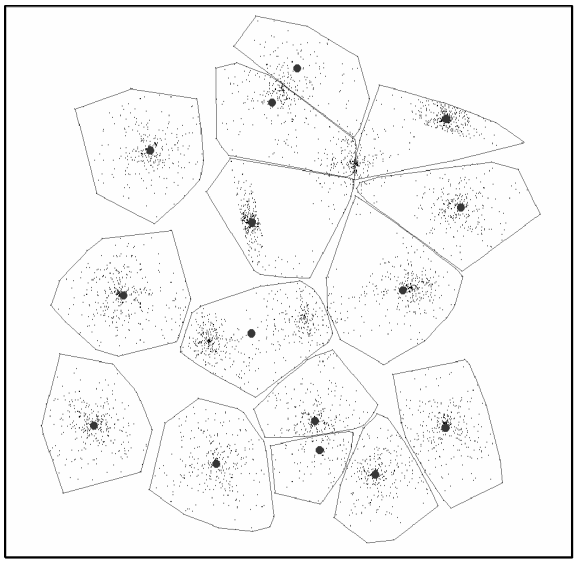
In random swap, centroid initialization does not play a significant role in determining the final solution. This is because whenever a solution seems like halting, the swapping of centroids helps it go ahead. The process and overall running of random swap algorithm is explained using the Figure 19.



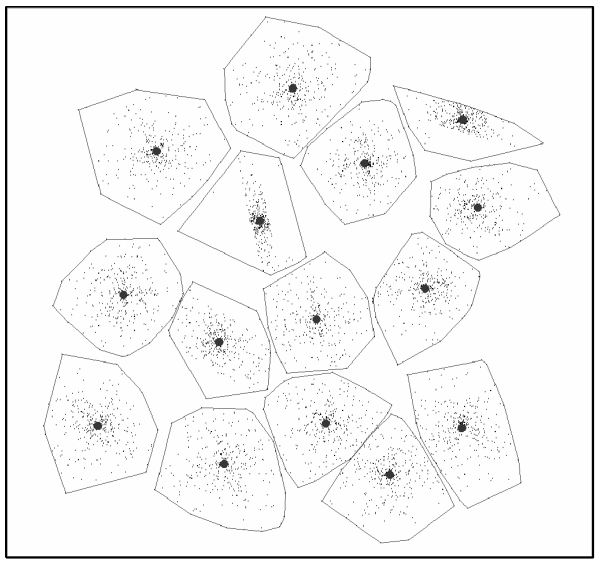
Solution after some iteration



Swapping centroid in next iteration



Local repartitioning



Final solution

Figure 19: Random swap at every step.
Adapted directly from [6].

4. Density-based clustering algorithms

There are many density-based algorithms that currently exist. These algorithms follow the idea that clusters exist as a group of points having some density. In short, higher density regions are classified as clusters and are separated from other clusters by a region of lower density. In this section, few density-based algorithms are explained to understand density-based clustering before heading to the main algorithm of this thesis.

4.1 DBSCAN

DBSCAN [21] is one of the most commonly used algorithms in clustering. It groups together the points which are close to each other. The faraway points are considered as outliers and are not included in the current cluster. DBSCAN requires two parameters as input namely minimum distance (ε) and minimum points (*minPts*). Every point is classified as a *core-point, edge-point,* or an *outlier*. Given a point *p* in the dataset, if *p* has at least *minPts* within a distance of ε, it is called as a core point. This ensures that point *p* has enough points around it and is not an outlier. It also means that point *p* is in a high-density region. All *minPts* within ε and are called *directly density reachable* points from *p*. A point *q* which is reachable by *p* through a link of directly density reachable points is called *density reachable*.

A point *q* is called an edge point if it is found within a distance of ε from any core point *p*, but *q* itself is not a core point. Both edge points and core points can be included in the cluster. But core points lie in the interior of the cluster while edge points lie on the border of the cluster to separate core points from outliers.

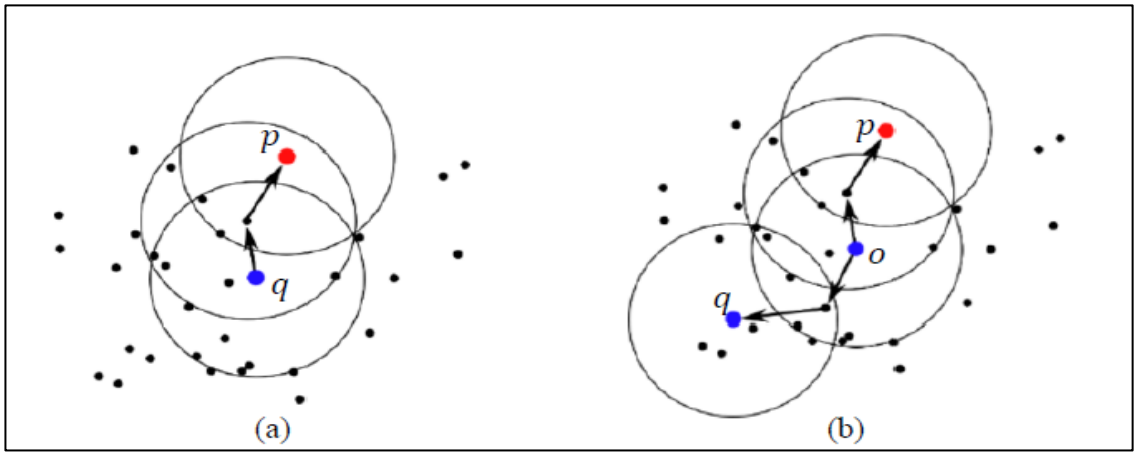


Figure 20: Density-reachable and directly density reachable adapted from [22].

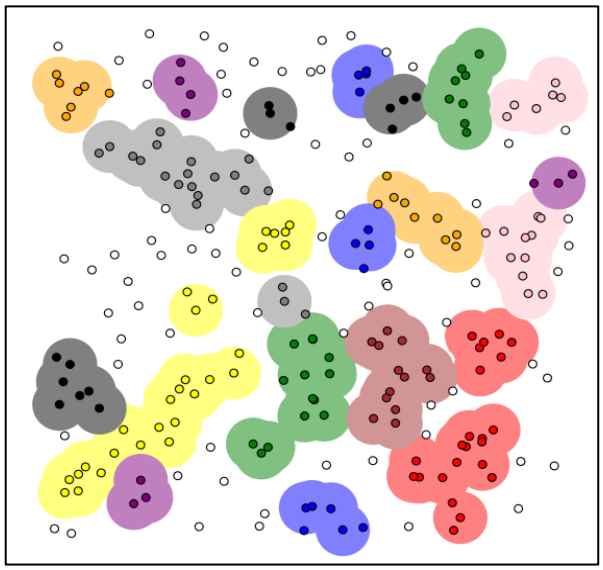
Any random point in the dataset is chosen as a seed, that is, the starting point in algorithm. It is determined if the selected point is a core point, edge point or an outlier and is marked so. If it is a core point, its adjacent point is tested for the same. In this way a cluster is formed by progressively merging all core and edge points in one cluster. Once a single cluster is formed, unvisited points are checked to see if they form another cluster or are outliers. DBSCAN has an average time complexity of O(*NlogN*) and worst-case time complexity of O(N^2). Every point is visited at least once in this algorithm to check for reachability. But a point can be visited

even more than once. For example, if point *p* is detected as an outlier for currently growing cluster, it may be detected as a core or edge point for some other cluster. Unlike k-means, DBSCAN does not require the number of clusters to be known in advance. It can effectively classify points as outliers. So, noise detection and removal of noise is achieved using this algorithm.

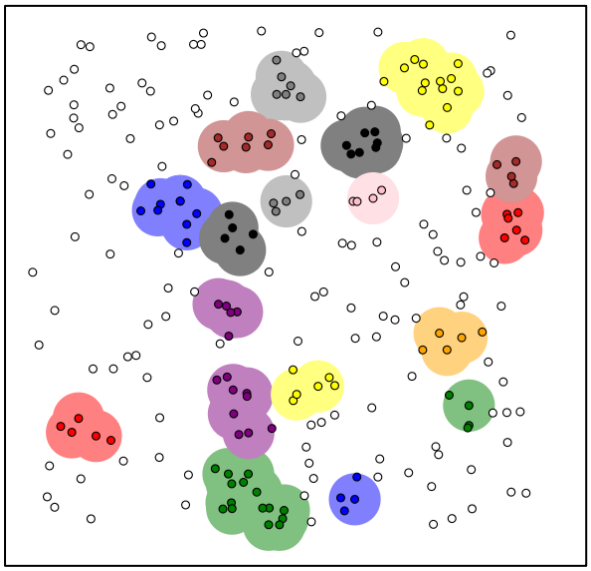
On the other hand, DBSCAN suffers from the curse of dimensionality in datasets having multiple dimensions. Multiple dimensions are hard to think of, impossible to visualize, and, due to the exponential growth of the number of values with each dimension, complete enumeration of all subspaces becomes intractable with increasing dimensionality. This problem is known as the curse of dimensionality [23].

Input parameters affect the performance of DBSCAN to a great extent. There are some guidelines while selecting input parameters *minPts* and ε. The number of dimensions of the dataset is known to the user. It has been experimented [24] that *minPts* = (number of dimensions + 1) works well in many cases. Or *minPts* = (number of dimensions * 2) is also not a bad choice. Lesser the *minPts*, more the number of clusters are detected. Having *minPts*=1 will not make any sense as every data point will be classified as an independent cluster.

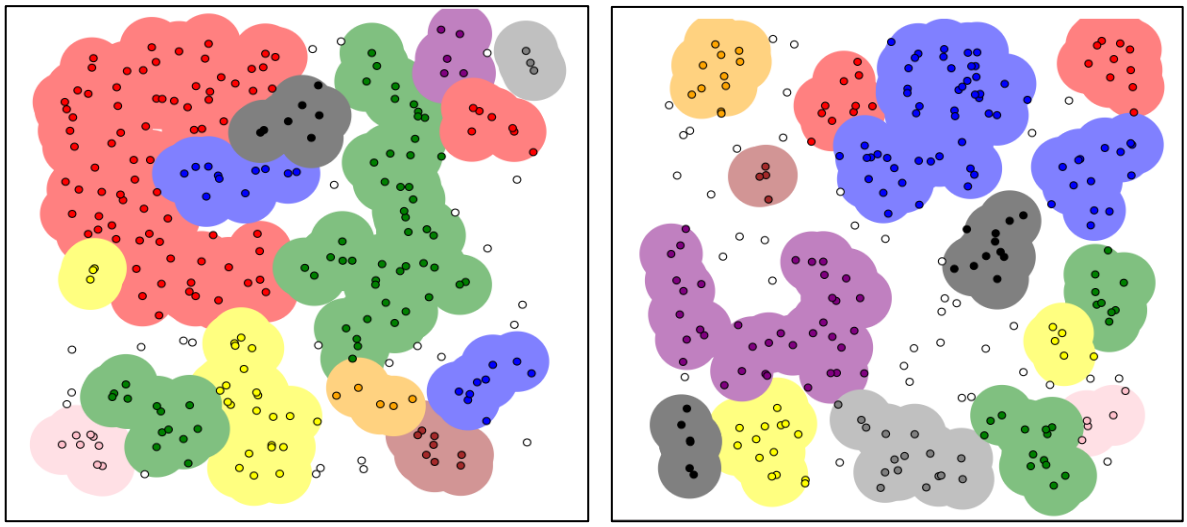
Large values may help points merge into a single cluster. Thus, large but lesser clusters are formed with higher value of ε and more clusters are formed with smaller value of ε. The clustering solutions are directly affected by input parameters minPts and ε. Figure 21 explains this argument. For the sake of simplicity, ε is not normalized to any scale, it is used just as a unit distance.



ε = 1 minPts = 3



ε = 1 minPts = 4



ε = 1.3 minPts = 3 **ε = 1.3 minPts = 4**

Figure 21: Clustering solutions of DBSCAN with varying ε and minPts[3]

The examples in figure 21 show different clustering solutions with varying values of ε and *minPts*. It is visually clear that a greater number of smaller clusters are formed with lesser value of ε and as ε increases, bigger clusters are formed. With increasing *minPts*, there is a probability to detect a greater number of outliers. The points without any color are the outliers, that is, points not belonging to any cluster. In real life data, outliers are usually random noise or unwanted data. In some cases, they have critical information. With increasing *minPts*, the average cluster size reduces and there is a tendency of forming a greater number of clusters.

4.2 Grid growing algorithm

Grid growing algorithm is another clustering algorithm which clusters data by separating high density regions from low density regions. It was proposed by Zhao et al. (2015) [25]. It is a grid-based approach but can also be categorized as density-based clustering. Other alternatives like k-means and hierarchical clustering have some disadvantages which are tackled by this grid growing algorithm. Unlike k-means, this algorithm does not require the information of number of clusters in advance and it is much faster than hierarchical clustering which has a time complexity of $O(N^3)$. This algorithm in general has three steps which are as follows:

1. Grid Construction
2. Initial Clustering
3. Merging grids

In the grid construction phase, a grid is constructed over the dataset. Grid is a collection of horizontal and vertical parallel lines. The size of grid depends on spread of dataset. Figure 22 shows grid construction.

[3] https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

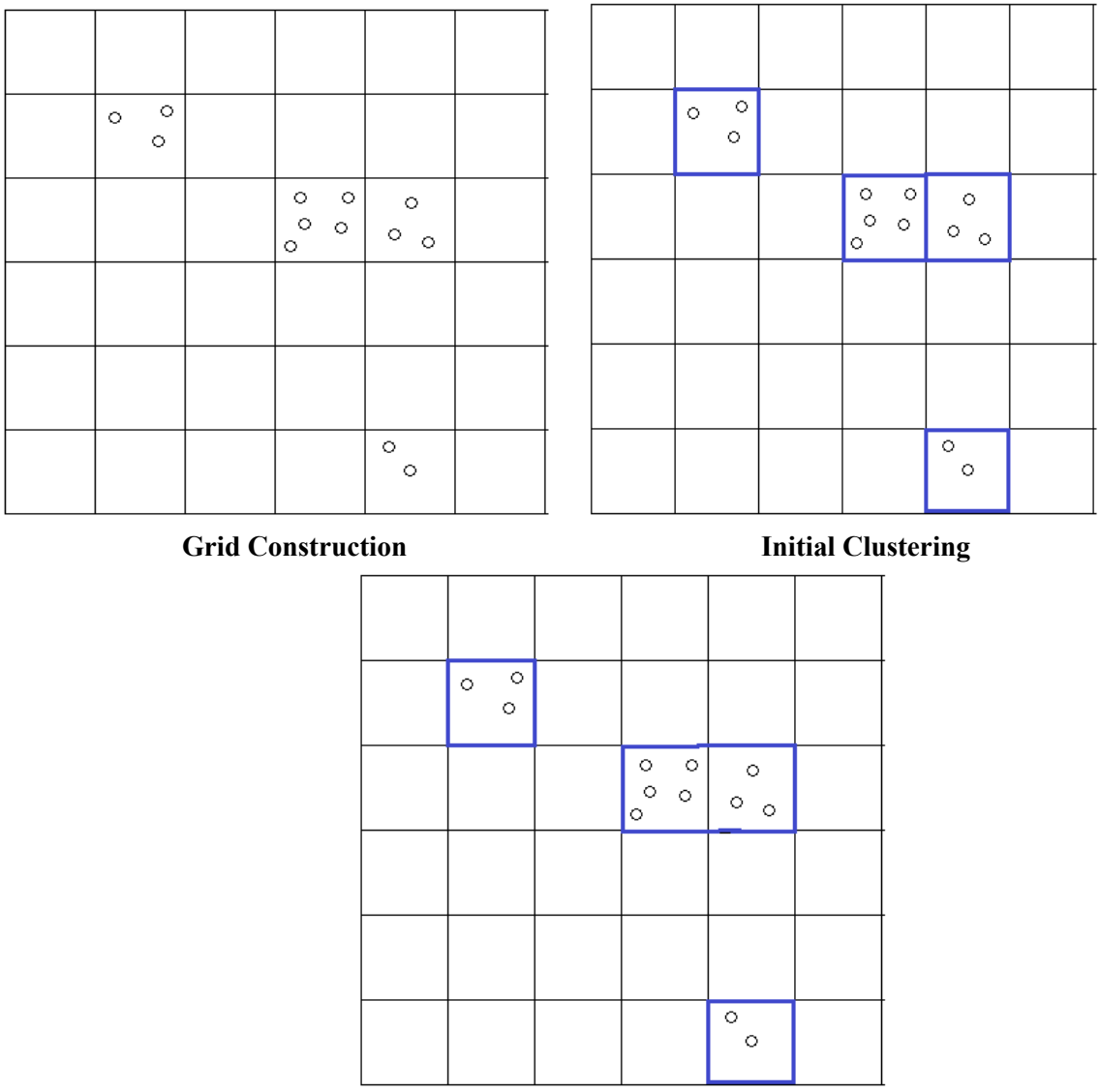


Figure 22: Steps in grid growing algorithm

In grid construction phase, every data point is assigned to a single grid. Next step is to get the first clustering, in which every grid is considered as one single cluster. Every point in one grid is a part of one single cluster. This step is simple enough to perform. Next step is merging the clusters into one single cluster based of several factors. This merging can happen by merging multiple clusters based on 4-neighbors or 8-neighbors' choice. Grids with no points are not merged. This algorithm has advantages of both k-means and DBSCAN and has time complexity of O(*NlogN*), which makes the algorithm fast.

4.3 Density peak clustering

Density peak clustering algorithm is based on the idea that cluster centers have a higher density than their neighbors and there is a region of low density between two regions of high density [26]. Clustering algorithms like k-means, k-medoids efficiently detect spherical clusters but are

not able to detect non-spherical clusters efficiently [32]. DBSCAN detects arbitrary clusters well but has two input parameters and deciding those is a non-trivial task. Density peak algorithm works well for arbitrarily spread clusters. This algorithm calculates the local density (ρ) for every point and its distance (δ) from a high-density point. Local density ρ_i for a point i is calculated as:

$$\rho_i = \sum_j \chi(d_{ij} - d_c) \tag{5}$$

Where, d_{ij} is the distance between point i and point j that satisfies triangular inequality. $\chi(x) = 1$ if $x > 0$ and $\chi(x) = 0$, otherwise and d_c is the cut-off distance. In simple terms, ρ_i is the number of points that are closer than d_c to i. δ is the minimum distance between point i and a point with higher ρ. It is calculated as:

$$\delta_i = \min(d_{ij}) \text{ such that j: } \rho_i < \rho_j \tag{6}$$

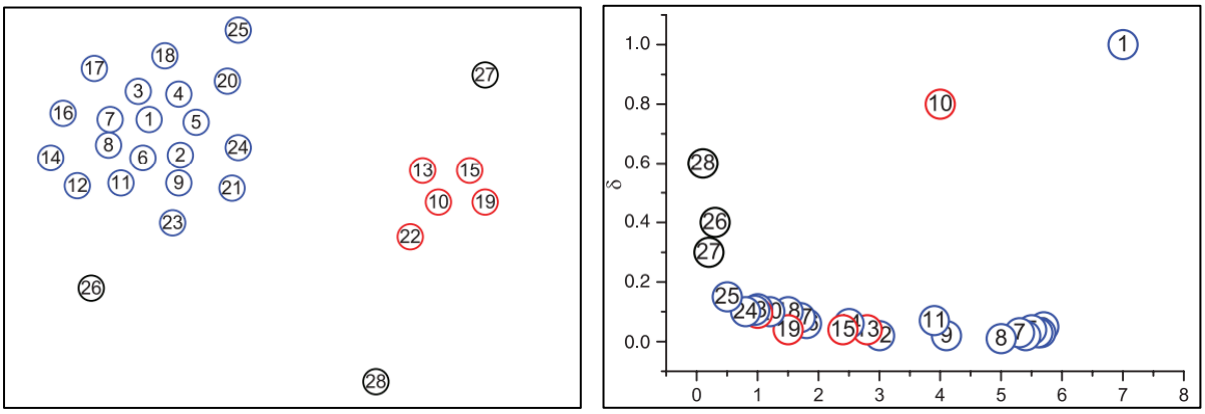


Figure 23: Data points with decreasing density and decision graph for those data points, adapted from [26]

Numbered points in Figure 23 are the points with decreasing density, which means, point 1 has the highest density and point 28 has the lowest density. Points with high values of both δ and ρ become the cluster centers. Points with only high δ are the points far away from the region of high density. They are formed as separate cluster called outliers. In this algorithm, d_c is the required input and it was observed in [26] that as a rule of thumb, d_c should be chosen such that the average number of neighbors is around 1 to 2% of the total data points in the data. In Figure 23, other points with blue circles are the points having δ closest to point 1. Points with red circle are the points having δ closest to point 10. Since point 1 is the point with highest ρ, its value of δ is taken to be 1 as it will not satisfy Equation 6. From the decision graph, the high-density points are seen.

5. Density-based weighing of distances

5.1 Weighted centroids

A centroid is a prototype which is used to represent the entire cluster. It is a vector whose features have a mean value of features of all the data vectors in that cluster. In addition to these features, we also introduce weights to every centroid in this thesis. The weights of a centroid can take any value from 0 to 1. The range of centroid weight is thus [0,1]. The weights play a significant role in attracting points from a distance into its cluster. Process of assigning weights to the centroids and the use of having weighted centroids is explained in Section 5.3.1. But it cannot be understood without first understanding the concepts of weighted distance.

5.2 Weighted distance

Distance between two vectors, or between a vector and a centroid is usually calculated using measures like Euclidean distance, squared Euclidean distance, Manhattan distance and Mahalanobis distance. The same distance measures are used in this thesis, but these distance measures are *weighted by weights of centroids*.

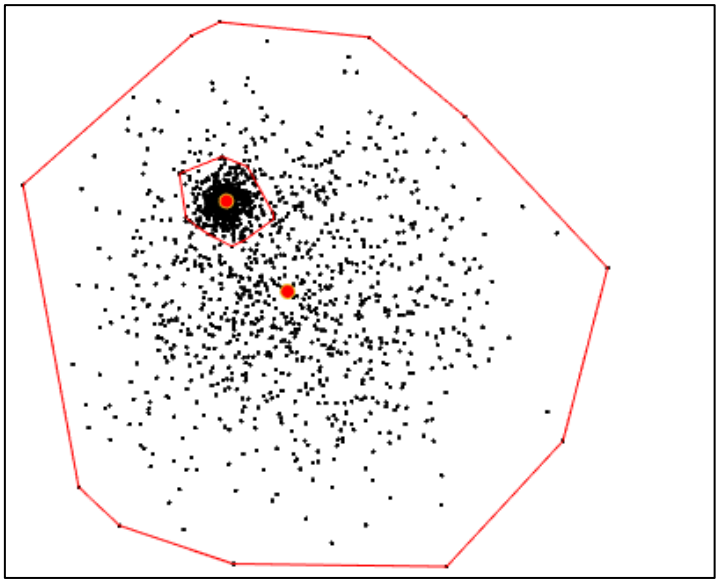


Figure 24: Nested Cluster

Figure 25 is a zoomed in view of Figure 24, which will illustrate the use of weighted centroid and weighted distance. It is seen that there are two clusters, one nested inside the other. In the Figure 24, centroid of the nested cluster is called $c1$ and centroid of the outer cluster is called $c2$. It is assumed for the sake of simplicity that the weight of $c1 = 0.55$ and the weight of $c2 = 0.18$. Why these weights are assumed in such a way will be clear only in the later sections when the entire algorithm is explained. A point $p1$ from the dataset is taken and its distance to the two clusters is studied using Euclidean distance and weighted Euclidean distance.

In Figure 25, distance ($p1$, $c1$) is 340 and distance ($p1$, $c2$) is 700. If k-means algorithm is taken into consideration, point $p1$ will be assigned to cluster with centroid $c1$ as the distance ($p1$, $c1$) is shorter. But from Figure 24, we see that a better clustering solution is obtained if $p1$ is assigned to cluster with centroid $c2$ even if it is closer to the densely compact cluster.

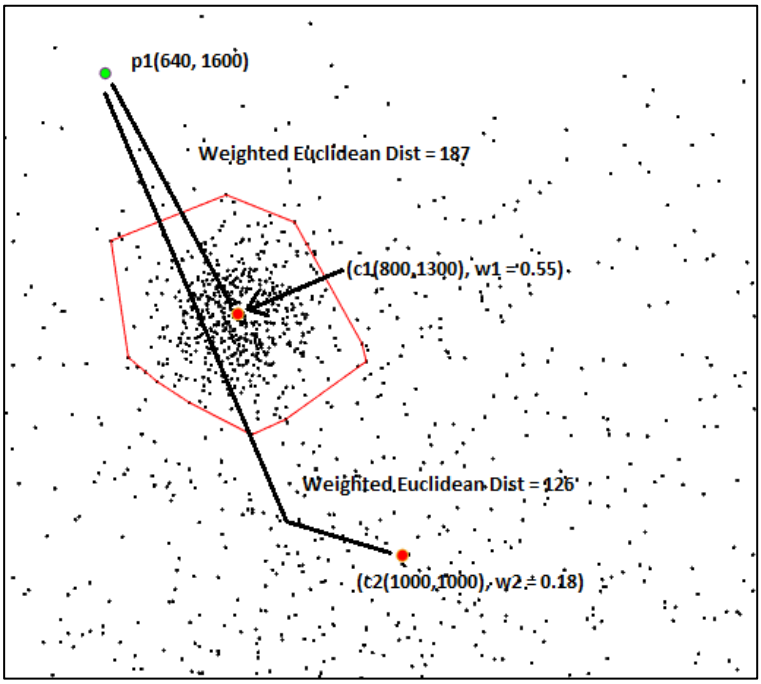


Figure 25: Point classification using weighted Euclidean distance

Weighted distance is calculated as follows:

$$WD = w * d(x, c) \tag{7}$$

WD is the weighted distance, w is the weight of centroid c and d (x, c) is the distance between centroid c and a data vector x. Weighted distance is simply normal distance times the weight of the centroid. In Figure 24, weighted distance ($c1$, $p1$) is calculated as 0.55*340 = 187 and weighted distance ($c2$, $p1$) is calculated as 0.18*700 = 126.

Point $p1$ is closer to centroid $c2$ than to centroid $c1$ when weighted distance is considered. In this way, it can be assigned to the cluster with centroid $c2$, which is the outer cluster and that is the thing wanted as seen from ground truth solution in Figure 24. Using weighted distance, a centroid can attract far away points in its cluster. This is based on a simple mathematical principle that multiplying a number with higher number between [0, 1] will give lower value as compared to multiplying same number with a lower number in the range [0, 1]. In practice, it means that centroids with higher weight can attract points from lesser circumference and centroids with lower weight can attract points from higher circumference.

5.3 Calculating centroid weights

Weight of a centroid plays a key role in attracting the vectors to its cluster. Weights depend on their density. Calculating centroid weights involve three steps which are:

1. Calculating mean distance
2. Calculating density
3. Calculating weights from cluster density

Each of these steps has been explained in detail.

5.3.1 Calculating mean distance

Given a cluster and its centroid, mean distance of that cluster is the average distance of all points from the centroid.

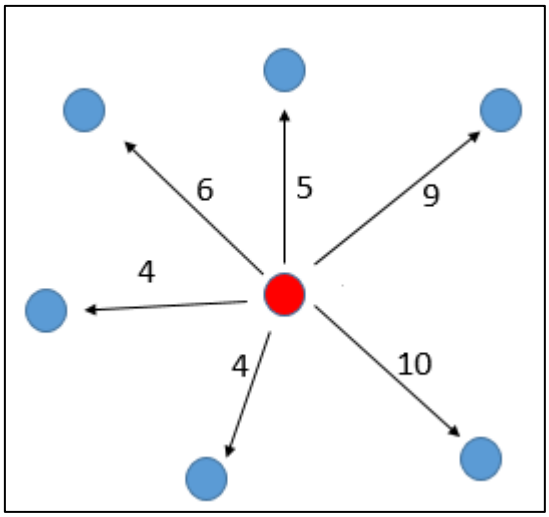


Figure 26: Mean distance is 6.33

Consider Figure 26 which shows a single cluster. Blue circles are the data points and red circle is the centroid. Distance of data points from the centroid are also shown. The mean distance is calculated using the formula given in equation 8.

$$\text{MD} = \text{Total distance}/n \tag{8}$$

Where *MD* is the mean distance, *n* is the number of vectors in a cluster and *total distance* is the sum of distances of all data points in one cluster to their centroids. In Figure 26, mean distance is calculated as:

$$\text{MD} = (4 + 4 + 10 + 9 + 5 + 6)/6 = 6.33$$

5.3.2 Calculating density

Discussions related to density of clusters come into picture at this stage. Density means the degree of compactness of a structure. In computing, it means the amount of information stored

by a substance. In clustering, density is the amount of information stored or contained by that cluster. Here, density is calculated by number of points in a cluster within mean distance. The formula for density is:

$$Density = \frac{n}{\mathrm{MD}} \tag{9}$$

Where *n* is the number of data vectors in a cluster. In Figure 26, density of the cluster will be:

$$\text{Density} = 6/6.33 = 0.947$$

The density of a cluster is directly proportional to the data points it has, and it is inversely proportional to the mean distance. Mean distance gives the approximate spread of a cluster in all directions. It is easier to estimate cluster size (and not the density) if its mean circumference is known. Density is higher in a cluster with lesser spread and high number of points in contrast to cluster with higher spread and lesser number of points.

5.3.3 Calculating centroid weights from cluster density

Updating weights change the power of centroids to attract data points. Centroids with lower weights can attract points from larger distances and centroids with higher weights can attract points from smaller distances. Centroid weights are calculated using the formula:

$$Weight(i) = \frac{Density(i)}{\sum_{j=1}^{k} Density(j)} \tag{10}$$

Where *k* is the total number of clusters in the dataset. Weight of a cluster is its density divided by the sum of densities of all the clusters in a dataset. Let us assume that there are five clusters in total in a dataset and one of those clusters is the cluster as shown in Figure 26. Let the sum of densities of all clusters be 5. Therefore, weight of the centroid of the cluster in Figure 26 is calculated as:

$$\text{Weight} = 0.947/5 = 0.19$$

Weight of a centroid is always proportional to the density of its cluster. Weight can also be called as normalized density as it is the density that scales down to the range [0,1].

There can be instances where a cluster has only one point. In such clusters, centroid and the only data point are the same. Here, the mean distance becomes zero, which makes its density infinite and then eventually leads to the weights having indefinite form. To avoid such situations, mean distance and density of such clusters are set to 1. This might not be the best choice, but better techniques can be investigated in the further studies.

6 Density-weighted random swaps

In this section, the algorithm used in this thesis is explained. This is a density-based algorithm using centroids. We use k-means and random swap algorithm and a new algorithm is built on top of it.

6.1 Algorithm

This section is the heart of this thesis as it the main algorithm worked on and experimented with is explained. It is a unique centroid based clustering algorithm which can detect nested clusters. It can be used as a replacement to k-means, DBSCAN and other clustering algorithms. The given algorithm works well even when there are no nested clusters. The density rich areas are detected by the algorithm and it tries to create a cluster out of density rich areas separated by the regions of lower density. Figure 27 shows the overview of this algorithm.

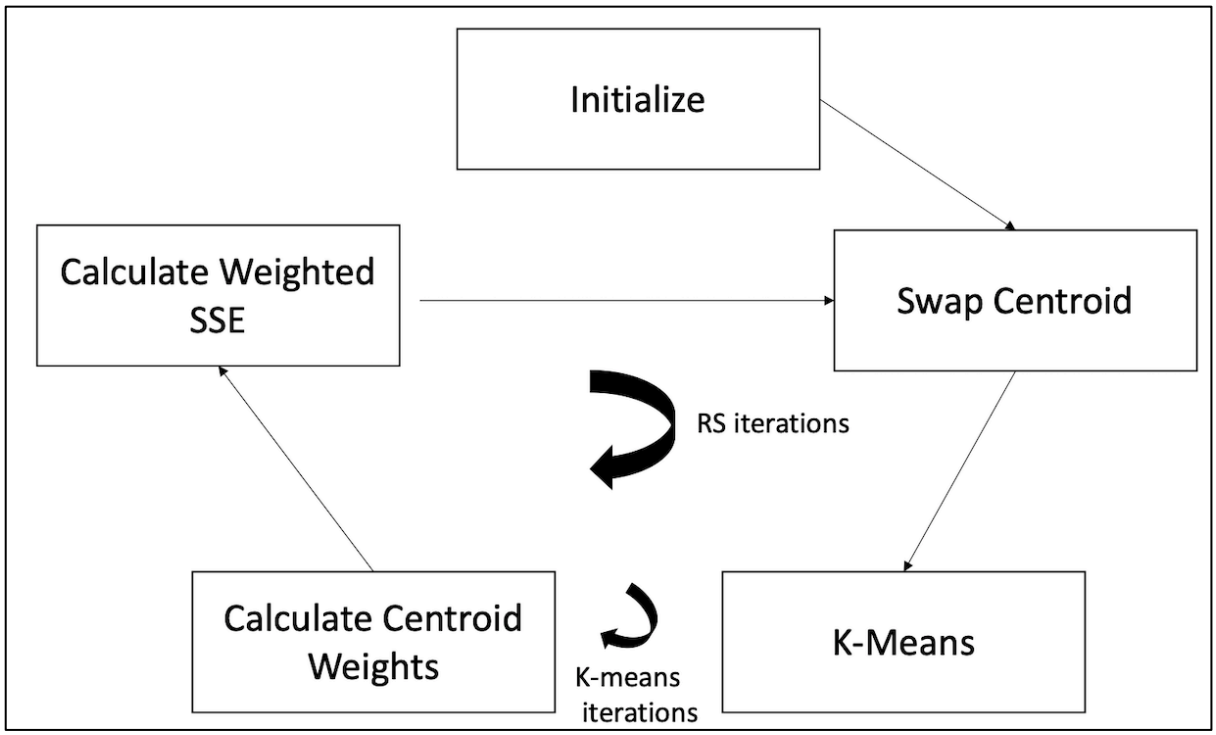


Figure 27: Steps in density-based clustering algorithm using random swap

There are five crude steps involved in density-based clustering algorithm using weighted random swap. These five steps are explained one by one in the following sub-sections.

6.1.1 Initialization

As in k-means and random swap, the centroids are initialized randomly. *K* centroids need to be initialized in the same way as in k-means. But along with the centroid features, centroid weights also need to be initialized. Centroid weights get updated in the later steps of the algorithm but initially, centroids need to assume some weights. In this thesis, weights are assumed in such a way that the sum of weights of all the centroids is equal to 1. Every centroid is given an equal

weight between 0 and 1 in the beginning. For example, if the dataset has 3 centroids, the weight of every centroid initially will be equal to 1/3 = 0.33. Similarly, if there are 100 centroids, weight of every centroid initially will be equal to 1/100 = 0.01. The formula for centroid weights initially is given as:

$$\text{Initial weights of centroids} = 1/k \tag{11}$$

where, k is the number of clusters in the dataset.

6.1.2 Swapping centroid

Once the centroids are initialized, random swap algorithm starts executing. All steps after this are the steps of random swap algorithm that have been explained in the Section 4. Since this is not exactly random swap, but weighted random swap, all steps are explained in detail again in this section as well. A centroid is chosen randomly from the set of current centroids and swapped to a random position in the dataset. With the swap, the centroid keeps its earlier weight. That is, weight of the centroid does not change before and after swapping. Only the values of centroid features or dimensions change.

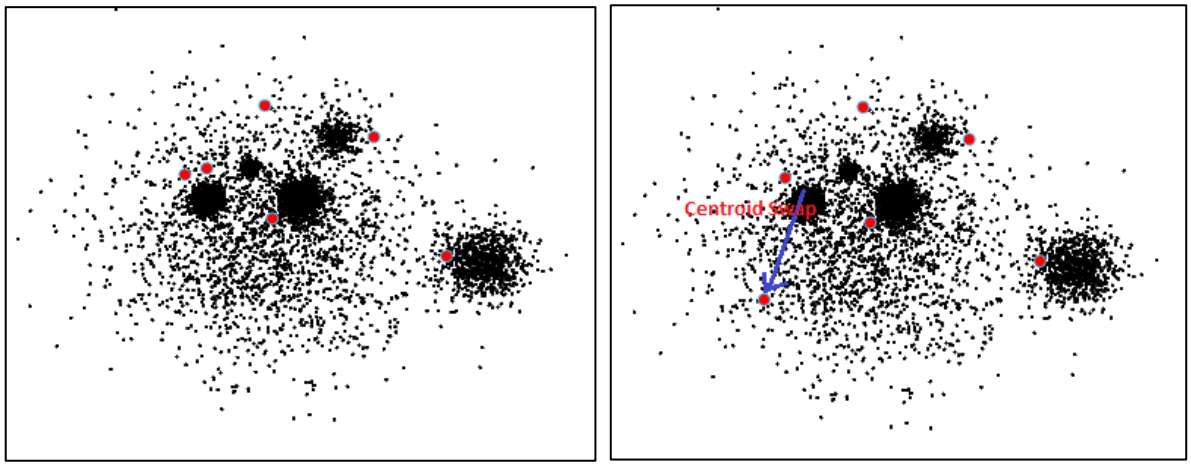


Figure 28: Random swapping of centroid

A centroid is chosen, and it is simply swapped randomly to a new location without any logic. The algorithm of this swap is same as in Figure 17.

6.1.3 Local repartitioning

Once random swap happens, new clusters need to be formed and their centroids need to be calculated before going ahead as the centroid swapped does not have any points associated with its cluster. Local repartition happens by running k-means algorithm only for a single iteration since there is only one active centroid in the current centroid list as compared to earlier centroid lists, and then two k-means iterations for fine tuning the solution. Fast k-means is used in local repartitioning and hence its time complexity is O(αN) [31]. Local repartitioning algorithm is given in Figure 29.

```
LocalRepartition(X, C, P, W, C_new): returns new partition
    for i = 1 to N
        if(i belongs to partition of swapped centroid)
            pi = GetNearestCentroid(x_i,C,W);
    for i = 1 to N
        if(x_i, C_new) < (x_i,C_pi)
            p_i = C_new ;
        else
            p_i = C_pi ;
return P;
```

Figure 29: Local repartitioning algorithm

The algorithm for getting nearest centroid, which is used in Figure 29, is given in Figure 30:

```
GetNearestCentroid(x_i, C, W): returns nearest centroid
    nearest_distance = ∞ ;
    for i = 1 to K
        distance = W_i * distance(xi, C_i);
        if(distance < nearest_distance)
            nearest_distance = distance ;
            nearest_centroid = i;
return nearest_centroid;
```

Figure 30: Algorithm to get nearest centroid

To find the nearest centroid, weighted distance is used in algorithm given in Figure 30.

6.1.4 K-means

K-means algorithm works in the same way here as described in the Section 3.1. However, some things have been changed to support weighted distances. K-means does not run for unknown number of times here because SSE is not calculated and compared after every k-means iteration. In k-means used in this algorithm, number of k-means iterations to be run is decided by the user. By default, it is 2. It was shown [27] that high quality clustering solutions are obtained by keeping the value of number of k-means iterations to 2. Another modification in k-means here is that the objective function is not calculated after every iteration. This is because the number of iterations is pre-decided, and the algorithm is let run for that number of iterations. Objective function is calculated at the end of all k-means iterations before the next random swap iteration begins. If the value of objective function in current iteration is lesser than value of objective function in earlier iteration, the solution is accepted.

Another important modification in this k-means is the calculation of centroid weights. The centroid weights are updated at the end of every k-means iteration. K-means algorithm for used in this density-based algorithm is as follows:

```
KMeans(X, C, P, W): updates P and W
    for i = 1 to T
        for i = 1 to N
            P_i = GetNearestCentroids(x_i, C, W);
        for j = to K
            C_j = CalculateNewCentroid(X, C, j);
        CalculateNewWeights(P, C);
```

Figure 31: Modified k-means for weighted density-based clustering

Another thing different in this k-means is that centroid initialization is not needed. As centroids are initialized randomly at the beginning of main random swap algorithm, initializing them one more time does not make any sense.

6.1.5 Calculating centroid weights

This step determines weights of the centroids. In the initialization section, it was explained that every centroid is assigned equal weight in such a way that sum of the weights of all centroids is 1. These weights are updated after every k-means iteration along with the centroid attributes. The entire algorithm to calculate weights of all the centroids in a dataset is:

```
CalculateNewWeights(X, C, P, W): updates W
    density_sum = 0;
    for i = 1 to K
        md_i = CalculateMeanDistance(X, C_i, P_i);
        d_i = CalculateDensity(md_i, P_i);
        density_sum = density_sum + d_i;
    for i = 1 to K
            W_i = CalculateClusterWeight(d_i, density_sum);
```

Figure 32: Algorithm to calculate new weights

6.1.6 Objective function

An objective function in random swap ensures that the best solution is accepted, and the errors are minimized. There are many choices available to select objective functions, but weighted SSE is used as objective function in this thesis. Like SSE, weighted SSE is also an objective function which is minimized. In this algorithm, objective function is not calculated at the end of every k-means iteration. Rather, it is calculated at the end of every random swap iteration. The formula for weighted SSE is as follows:

$$Weighted\ SSE = \sum_{i=1}^{N} w_i * \left|\left|x_i - c_{pi}\right|\right|^2 \quad (12)$$

The optimization function is still sum of squared errors. Since the distance measure is weighted distance, SSE is also multiplied by the weights of centroids. And the solution is accepted only if weighted SSE of current iteration is lesser than weighted SSE of earlier iteration.

```
CalculateWeightedSSE(X, C, P, W): returns weighted SSE
    sum = 0;
    for  i = 1 to K
      for j = 1 to count(Pi)
          sum = sum + wj * SquaredEuclideanDistance(nj, Ci);
    return sum;
```

Figure 33: Calculate weighted SSE

The complete algorithm used in this thesis is as shown in the Figure 34:

```
DensRS(X, C, P, W, RS_Iter, KMeans_Iter, K): updates W

    InitializeCentroids(X);
    InitializeWeights(W);
    KMeans(C, X, P, W);

    for i = 1 to RSIter
        error_previous = error_current;
        RandomSwap(C, X);
        LocalRepartition(C, X, P, W);
        error_current = CalculateWeightedSSE(X, C, P, W);
        if(error_current < error_previous)
            //Accept Solution
            SaveNewPartitions();
            SaveNewWeights();
```

Figure 34: Density based clustering algorithm using weighted random swap

6.2 Example

Figure 35 shows step by step execution of density based random swap algorithm with weights and densities at each step. In the given example, random swap was run 1000 times and k-means was run 2 times in every RS iteration. Final solution was accepted after 4th RS iteration.

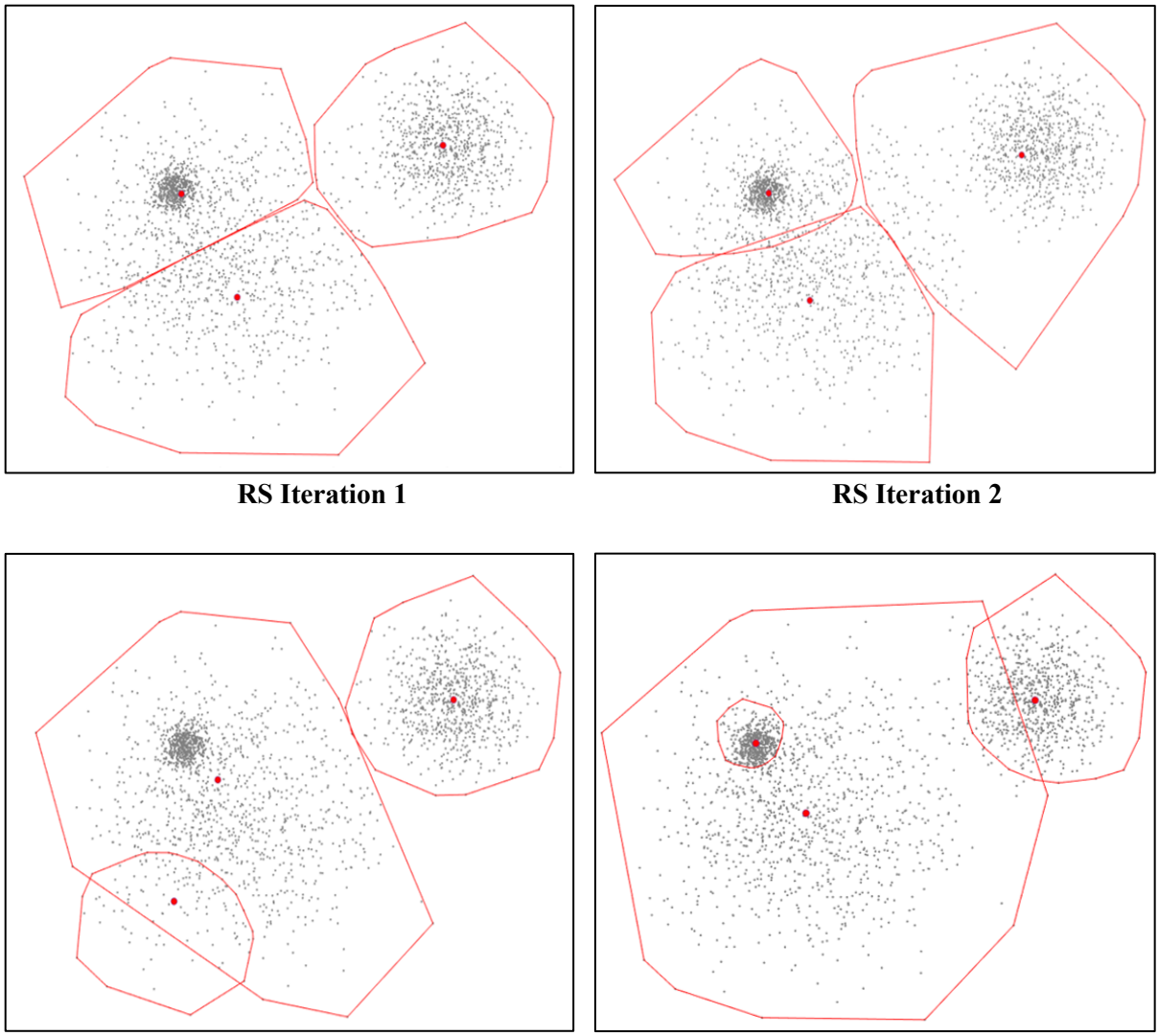


Figure 35: Output after every RS Iteration

Since, equal weight is assigned to all centroids initially, no centroid has more or lesser power to attract data points far away from it. Hence first iteration in RS is like first iteration in k-means. When the weights of centroids are changed for the first time after first RS iteration, the algorithm starts detecting nested centroids gradually. It is seen that solution to above dataset is also found by using weighted k-means instead of weighted RS. Since the only algorithm used in this thesis is weighted random swap, above example is shown using weighted random swap itself. Detailed analysis about Figure 35 is made and weights and centroids at the end of every iteration are shown. Bigger arrow shows the current centroid will be swapped to new location in the next iteration.

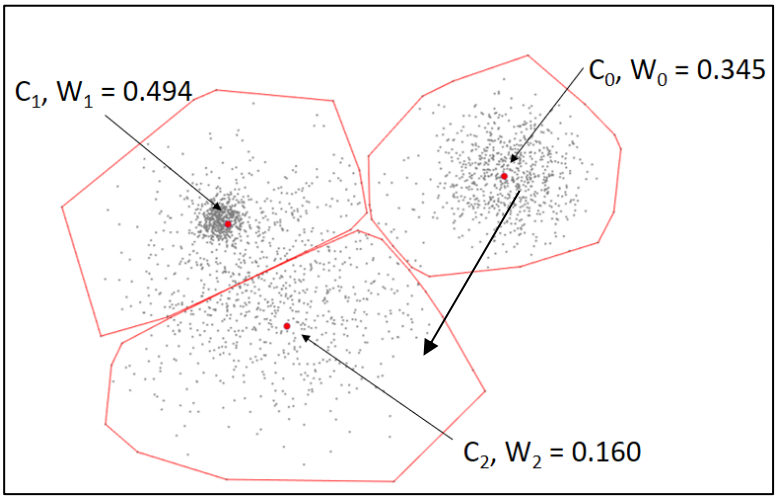


Figure 36: Centroid weights at the end of RS iteration 1

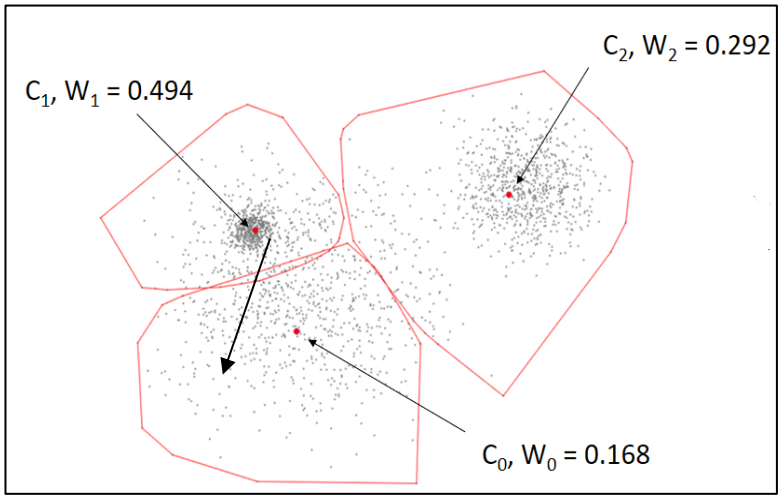


Figure 37: Centroid weights at the end of RS iteration 2

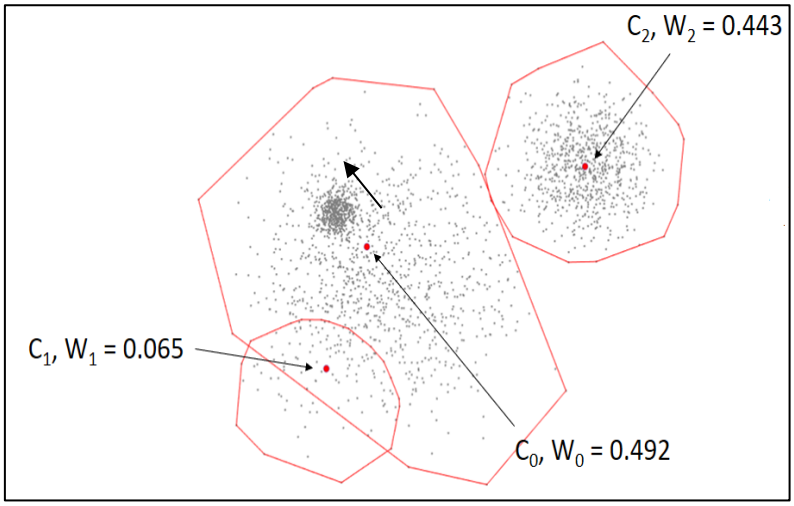


Figure 38: Centroid weights at the end of RS iteration 3

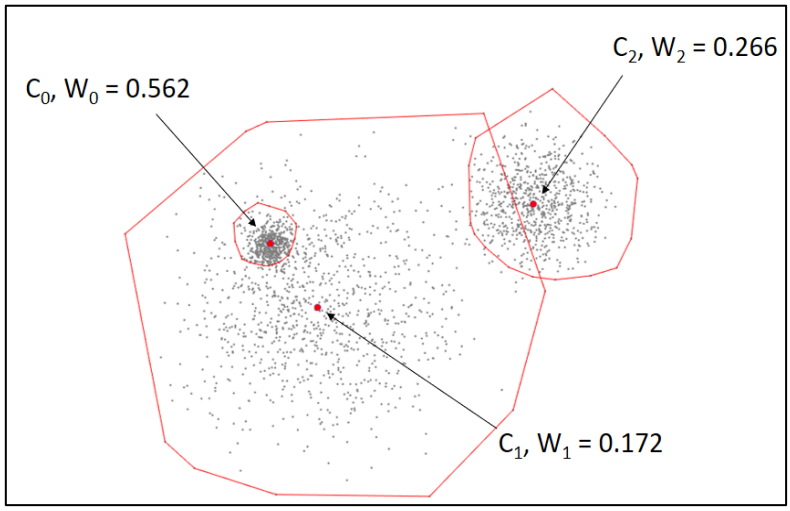


Figure 39: Centroid weights of final solution

In this way, weights are adjusted after every RS iteration and the ability of centroids to attract points changes according to their weights. The SSE for all iterations is given in Table 1:

Table 1: Performance with iteration count

Iteration Number	Weighted SSE	CI	CSI
1	$61.9 * 10^6$	1	0.56
2	$56.4 * 10^6$	1	0.60
3	$52.5 * 10^6$	2	0.76
4	$32.8 * 10^6$	0	0.99

SSE is minimized at iteration 4. It is not necessary that SSE will be minimized after 4^{th} RS iteration every time. This also depends on the way swaps take place. But for such trivial datasets, few RS iterations are needed. CSI and CI values are also tacked for every iteration. In final solution where SSE is the minimum, CI reaches 0 and CSI almost reaches 1. Concepts of CI and CSI are explained in section 6.2. For now, we can assume that clustering solutions get better as CI approaches 0 and CSI approaches 1. The reason CSI does not reach 1 is explained in Section 7.4.

6.3 Problems with calculating weights dynamically

Calculating centroid weights is an overhead of the algorithm described in this section. As per Piiroinen, Jarkko 2019, the weights are calculated dynamically at the end of every random swap iteration if the solution was accepted. The weight adjusting random swap algorithm according to Piiroinen, Jarkko 2019 is given in Figure 40.

WeightAdjustingRandomSwap(X, W, P, C): → (C, P)
REPEAT T TIMES (Cnew,j) ← SwapCentroid(X, C)
Pnew ← LocalRepartition(X, W, Cnew,P,j)
(Cnew, Pnew) ← K-means(X, W, Cnew, Pnew)
IF f(Cnew, Pnew,W) < f(C, P)
THEN (C, P) ← (Cnew, Pnew)
W ← CalculateWeights(X, P, C)
RETURN (C, P)

Figure 40: Pseudo code for random swap algorithm with weight calculation step. Adapted from [3], Figure 47.

The weights are calculated once the solution is accepted, that is, after calculating weighted SSE. This means, weighted SSE is calculated using stale weights. Therefore, the weighted SSE calculated after random swap is not correct. Incorrect weighted SSE leads to accepting of incorrect clustering solutions. If incorrect solution is accepted, some centroids simply appear to be misplaced. Figure 41 shows an example of incorrect solution accepted from the thesis Piiroinen, Jarkko 2019.

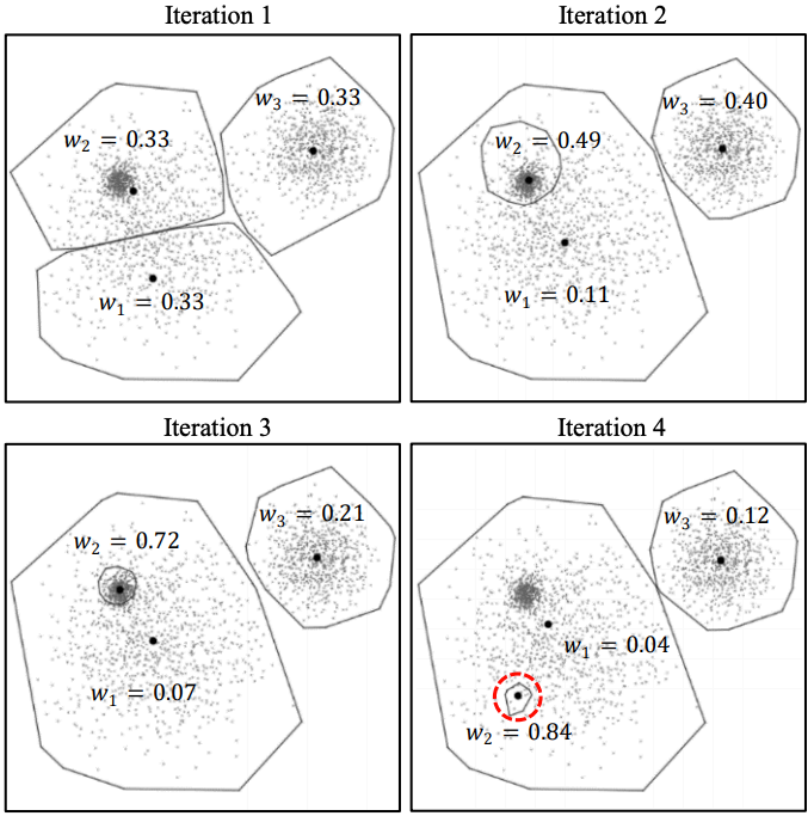


Figure 41: Incorrect solution accepted. Adapted from [3], Figure 49.

No improvement in clustering results were seen even after 10000 random swap iterations. In these cases, weight of a randomly swapped centroid becomes extremely low and then it also produces lower weighted SSE. Once the centroid weights become extremely low, it is difficult to find new locations for them that would result in lower weighted SSE. Even when one of the

centroids was moved from centroid rich area to centroid poor area, weighted SSE did not improve.

The solution for this problem was correcting the place of calculating centroid weights. The correct place of calculating weights is at the end of every k-means iteration. In this thesis, the weight of centroids is calculated in every k-means iteration and fresh weights are provided as input to the function calculating weighted SSE. The weight calculation step is added in k-means as shown in Figure 32. All the results with this improvement are shown in Section 6.

7. Experiments

7.1 Tools used for experiments

All experiments were run on the server of UEF, machine learning group. Running big datasets like birch for 100 times was a very time-consuming process. Hence some automation scripts were written to run the algorithm multiple times and generate the evaluation measures CI, CSI, and SSE data automatically. The inputs for this automation tool were dataset name, k-means iterations, random swap iterations and the number of times this algorithm needed to be run. For doing detailed analysis, provisions were made to generate centroid weights and partitions after every k-means iteration where necessary. This enabled to get deeper insights on what algorithm is exactly doing per iteration. The visuals in this thesis are generated using matlab. A tool in matlab was written which took dataset, centroid file and partition file as inputs and generated the clustering output. Another tool was written in matlab which could give mean distance, density, and weight of every desired cluster in any dataset. The second tool was used for detailed analysis of the datasets and algorithm.

7.2 Cluster evaluation measures

For supervised machine learning algorithms, there are several measures like accuracy, precision and recall for evaluating quality of the model. For cluster analysis, finding *goodness* of a cluster is challenging as data does not have any labels associated with it. K-means is an unsupervised machine learning algorithm as there is no label associated with any cluster. But in universities and research centers, different algorithms are developed, tried, and tested. It is necessary to compare the algorithm under development to the existing algorithms even though the algorithms are unsupervised. Research is performed and results are tested over a set of carefully selected sample datasets. The ground truth, which is the best solution of these sample datasets, is known in advance so that the results can be compared to evaluate quality of new algorithm in research process. Numerical measures that are used to evaluate cluster quality are divided into following categories:

1. Internal Index
2. External Index

In internal index, the quality of cluster is measured without using any external information. All information is derived only from the existing data in the cluster. Example of internal index is SSE., which does not require ground truth.

External index is used for searching good clusters by comparing similarity between ground truth and the clustering solution. Nearer the solution to the ground truth, better is its quality. Using ground truth, similarity of two clusters is derived. External validation measures can be classified into *pair-counting, information theoretic* and *set-matching* measures [28]. Set matching indices are used as evaluation measures in this thesis.

Set matching indices are classified as point-level or cluster-level. Point level indices study every point in the dataset and consider the intersection of paired clusters in the clustering solution and ground truth. In cluster level indices, the general position of clusters and their centroids is compared with the ground truth instead of comparing every point. In [33], Fränti et al. proposed an external set-matching index measure called *centroid index (CI)*. It is a cluster level index with the assumption that most general information about a certain cluster is contained in its prototype. Comparing centroids (or other prototypes) will determine the goodness of a clustering solution in terms of CI. With CI, solutions are distinguished by measuring differences at the cluster level. Calculation of CI is based on pigeonhole principle. If prototypes of solution A are considered pigeons, and prototypes of solution B as pigeonholes, every prototype in A should have a corresponding one to one mapping in solution B. Given two sets of centroids, CI is calculated as follows.

Let the centroids for two solutions be $C = \{C_1, C_2, \ldots, C_{k1}\}$ and $C` = \{C`_1, C`_2, \ldots, C`_{k2}\}$. Out of these sets, nearest neighbor mappings $C \rightarrow C`$ are shown as follows:

$$q_i \leftarrow \arg\min \|c_i - c`_j\|^2 \; \forall \; i \in [1, k_1] \tag{13}$$

For each $c`_j$, the number of prototypes mapped to it in c_i are calculated. Centroid is an orphan centroid when no centroids in the other solution consider it as nearest centroid. The formula to find if the target centroid is orphan or not is given as:

$$orphan(C`_j) = \begin{cases} 1, & q_i \neq j \; \forall \; i \\ 0, & otherwise \end{cases} \tag{14}$$

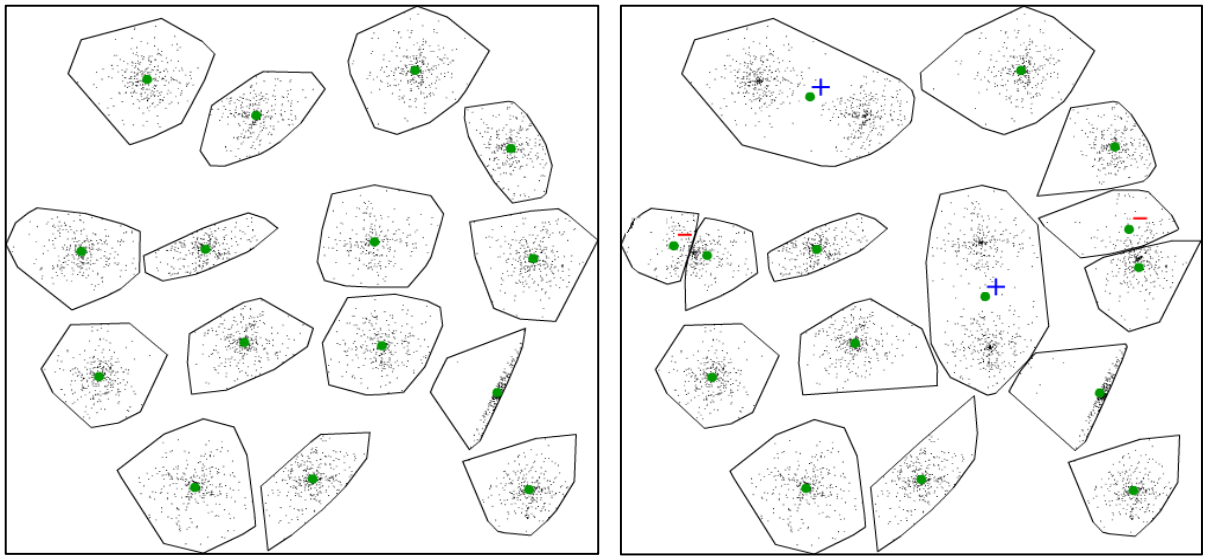
The dissimilarity between C_i and C_j is the number of orphans in the target clustering solution:

$$CI_1(C, C`) = \sum_{j=1}^{k_2} orphan(C`_j) \tag{15}$$

The CI is then calculated as the sum of orphans in each solution.

$$CI(C, C`) = \max\{CI_1(C, C`), CI_1(C`, C)\} \tag{16}$$

One thing however to note here is that, CI is not symmetric. That is, CI (C_i, C_j) ≠ CI (C_j, C_i). In this way, it tells how many clusters in one solution are differently found from clusters in another solution. Use of CI is to compare clustering solution to ground truth and thus CI=0 shows correct clustering when measured by the allocation of clusters. Figure 42 shows CI using two different solutions of the same dataset.



CI = 0 CI = 2

Figure 42: Two different solutions of same dataset.

The centroids represented with minus sign (-) are the orphans, that is, they do not have corresponding prototypes in the first figure. There are two centroids with a minus sign, hence CI for the Figure 42 is 2. Since there are two orphans, it also means that there are two clusters with extra child.

CI gives rough integer values about the cluster level similarity of two clusters since it is based only on prototypes. But if more detailed values of similarity are needed, point level similarity index called centroid *similarity index* (*CSI*) is used. CSI is an extension of CI where number of points shared by corresponding clusters is found and CSI is calculated as:

$$CSI = \frac{S_{12} + S_{21}}{2} \tag{17}$$

Where $S_{12} = \frac{\sum_{i=1}^{K_1} c_i \cap c_j}{2}$ and $S_{21} = \frac{\sum_{j=1}^{K_2} c_j \cap c_i}{2}$

CSI gives more precise similarity value than CSI but lacks intuitive judgment about the centroids which are placed incorrectly. In this thesis, CI and CSI are extensively used in the experiments section.

7.3 Data

This section deals with the input data used for experimentation and deriving conclusions about the density-based clustering algorithm. In total, 14 datasets were used in the experimentation and the list of datasets is given in Table 2.

Table 2: Datasets used in experiments

Dataset Name	Number of clusters	Number of data vectors
n3	3	2250
n6	6	5500
s1	15	5000
s2	15	5000
s3	15	5000
s4	15	5000
a1	20	3000
a2	35	5250
a3	50	7500
b1	100	100000
b2	100	100000
b3	100	100000
dim032	16	1024
dim064	16	1024

The datasets n3 and n6 are synthetic datasets used for primary testing of the algorithm. They are generated by a student of the UEF, machine learning group named Jarkko Piiroinen to test results in his master's thesis [3]. Any change in the algorithm while experimenting was tested first on these two datasets and if the results were correct with n3 and n6, experiments with other datasets were also carried. Dataset n3 has three clusters with one nested cluster and n6 has six clusters with four nested clusters. They are the paragon candidates to test algorithm to detect nested clusters.

Datasets s1, s2, s3 and s4 are synthetic datasets of 2-dimensional data. They are Gaussian clusters with different degrees of overlap. Notable nested clusters are not present in these datasets. These have 5000 vectors and 15 clusters. Datasets a1, a2 and a3 are synthetic 2-dimensional datasets with increasing number of clusters. There are 150 vectors per cluster in this series of dataset. Birch datasets are the most complicated type of datasets. They have 100000 vectors divided into 100 clusters in a complex manner. Birch1 has data spread like a grid, so it is easier to cluster. Birch2 has data points spread in a sinusoidal wave shape. Birch3 has no shape. Data points are spread randomly and there are many potential nested clusters too. Dim032 and dim064 are 32 dimensional and 64 dimensional datasets, respectively. Both have 1024 vectors. Clusters are well separated in both the datasets.

Ground truths for all the datasets are present. All these datasets, except n3, n6 and birch3, are from the clustering basic benchmark [29]. Ground truths include the ground truth centroids and ground truth partitions.

7.4 Results

In the preliminary version of the algorithm [3], it was shown that objective function weighted SSE is much better than normal SSE. Also, it was shown that weighted random swap performs better than weighted k-means. Weighted k-means can detect nested clusters from trivial datasets like n3, but it did not work for complex datasets like n6. So, the basic comparison between k-means and random swap was made. Since it was concluded that weighted random swap works better than weighted k-means, this version of thesis sticks with the experiments with weighted random swap. Also, since it was proved weighted SSE is a better objective function than normal SSE, experiments in this thesis are restricted only to weighted SSE as objective function, though the results obtained in preliminary version of algorithm are also given. In [3], the experiments were carried out only with datasets n3 and n6, while in this version, experiments have been carried out with 15 different datasets. The results obtained by Jarkko Piiroinen in [3] are shown in Table 3.

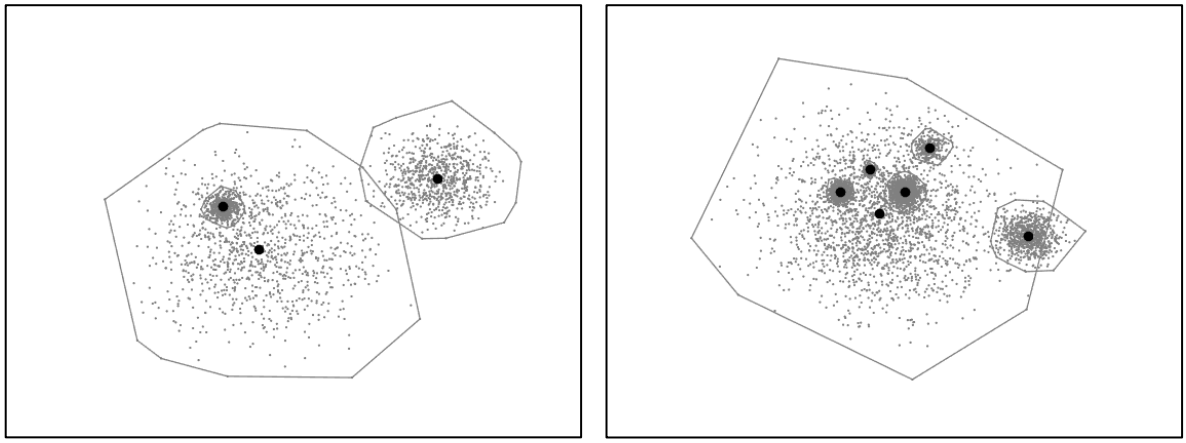
Table 3: Results with k-means and RS when SSE is used as objective function. Adapted from [3], Table 4.

Algorithm	**Success%**	**CI**	**CSI**	**SSE (* 10^8)**
Dataset n3				
K-means	100%	0.00	0.80	1.51
Random Swap	100%	0.00	0.80	1.51
Dataset n6				
K-means	0%	1.80	0.71	1.72
Random Swap	0%	1.03	0.58	1.65

Table 4: Results with k-means and RS when weighted SSE is used as objective function. Adapted from [3], Table 5.

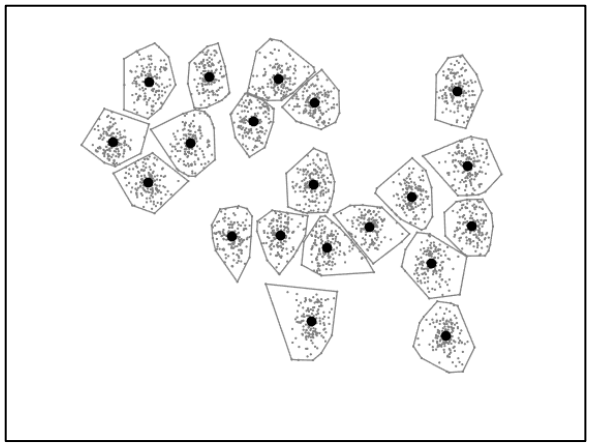
Algorithm	**Success%**	**CI**	**CSI**	**Weighted SSE (* 10^8)**
Dataset n3				
K-means	54%	0.47	0.83	1.26
Random Swap	100%	0.00	0.91	2.73
Dataset n6				
K-means	5%	1.78	0.70	1.26
Random Swap	62%	0.38	0.80	1.03

Table 3 and Table 4 show that weighted SSE performs better as an objective function as compared to normal SSE in this algorithm. Also, random swap is seen to perform better than k-means in most of the cases. Ideally, CI is an integer, but the value of CI in the above tables is in decimal points. This is because the algorithm is run for 100 iterations and CI is noted in every iteration. The CI in 100 runs is averaged to get a single value. The output of experimental results on all the datasets as derived in this version of thesis is given in the following figures.

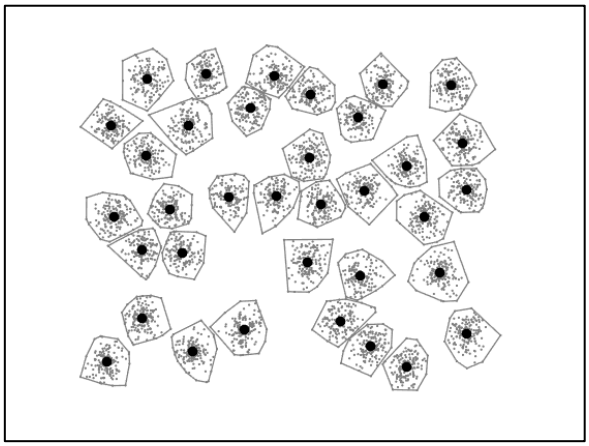


N3 CI = 0 N6 CI = 0

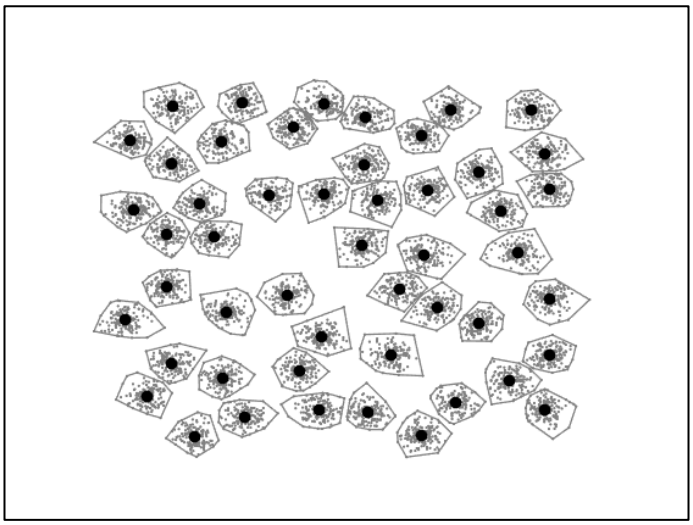
Figure 43: N-series dataset clustering solutions



A1 CI =0

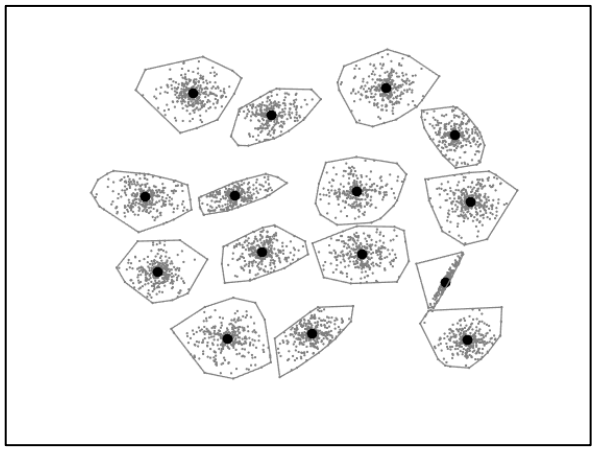


A2 CI = 0

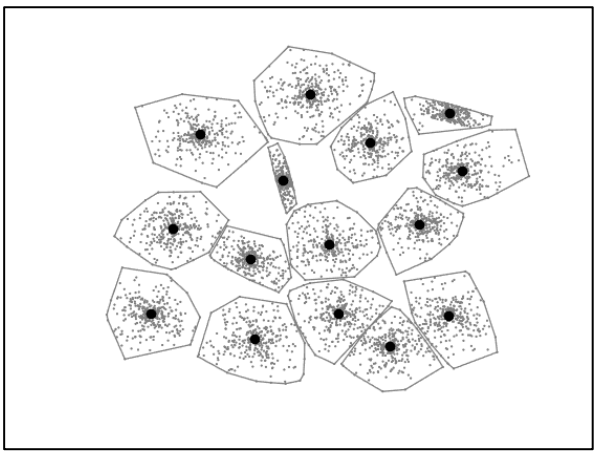


A3 CI = 0

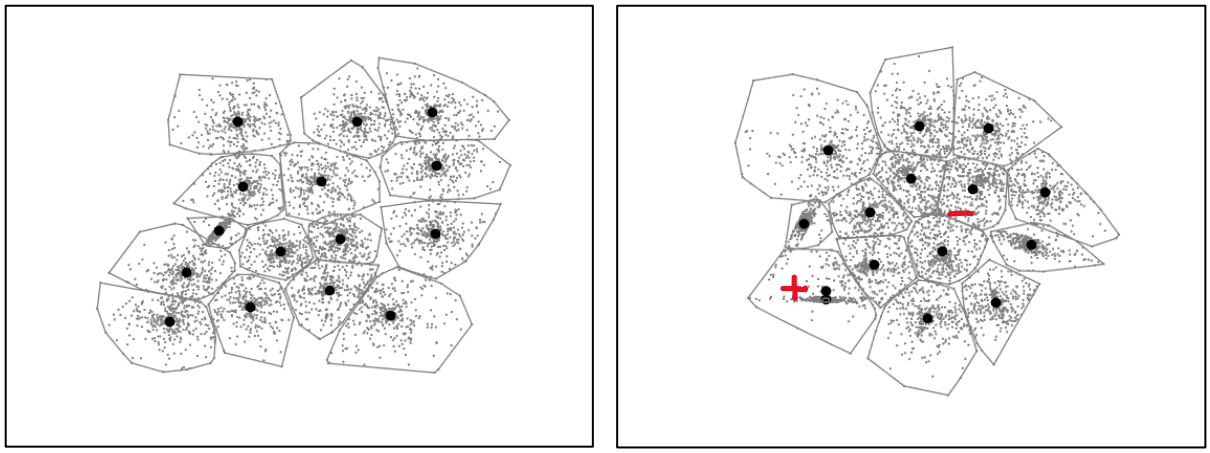
Figure 44: A-series dataset clustering solutions



S1 CI = 0

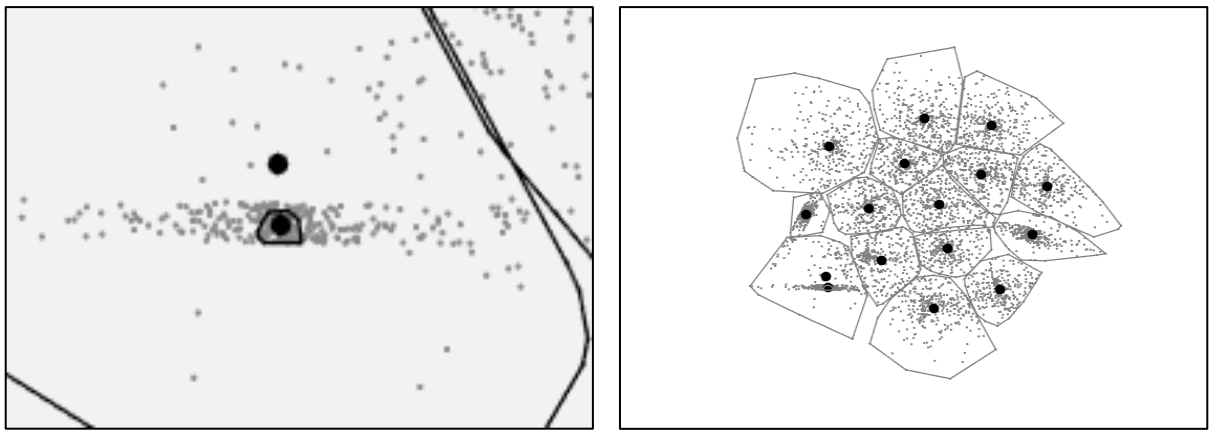


S2 CI = 0



S3 CI = 0 S4 CI = 1

Figure 45: S-series datasets clustering solutions



Nested cluster zoomed S4 with 16 clusters CI = 1

Figure 46: More analysis on S4 dataset

Datasets in S-series and A-series are straight forward, easy to cluster, 2 dimensional datasets. From the experiments, density-based clustering algorithm works well for datasets which do not have nested clusters too. An interesting observation has been made in dataset S4 in Figure 45. In the left bottom cluster, a nested cluster has been detected (showed by '+' sign) and a cluster in the middle part is not detected (showed by '-' sign). The nested cluster detected is very tiny, hence the convex hull cannot be seen. The convex hull of the nested cluster is shown in Figure 46. To confirm the algorithm detects the missing cluster, the experiment was carried with k=16 and then the missing cluster was detected. With 16 clusters, CI values is still 1 because the ground truth has only 15 clusters. Though in ground truth, the nested cluster does not exist, the density-based algorithm is still capable of finding the nested patterns and giving the clusters based on their density values. Datasets n3 and n6 on the other hand have clusters as expected and the nested clusters have been perfectly detected. The results of more complicated birch datasets are as shown in Figure 47.

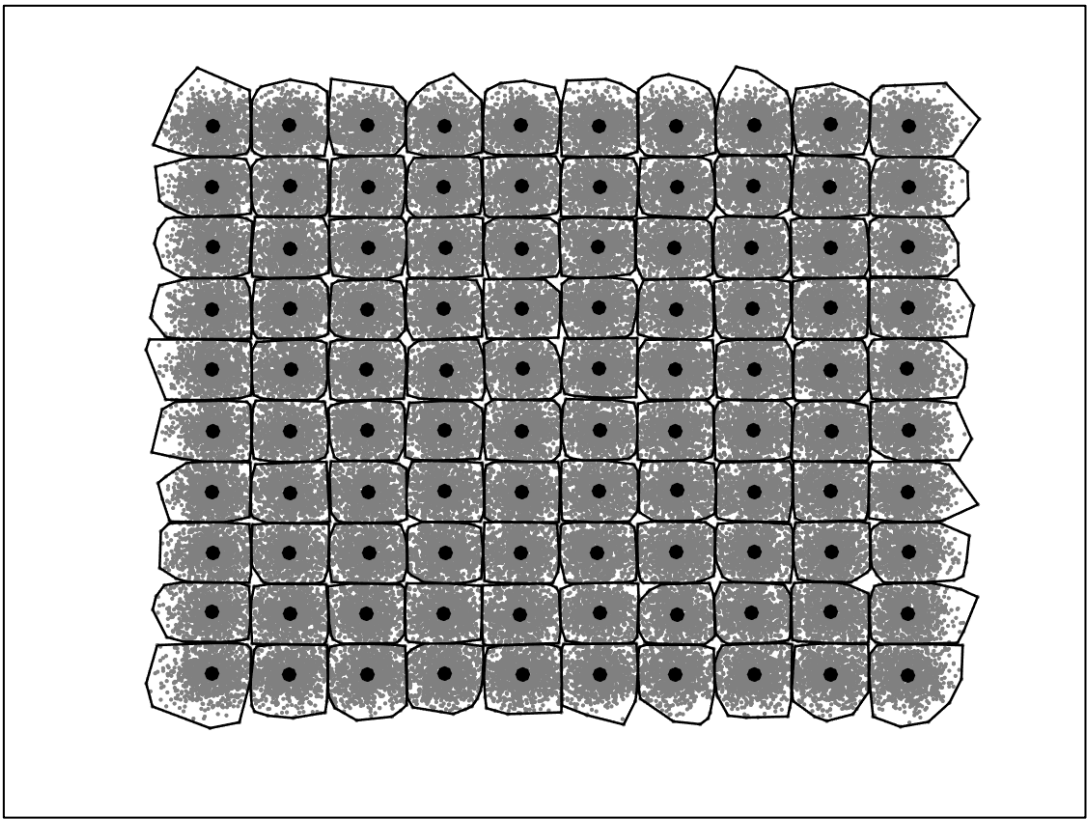


Figure 47: Birch 1 dataset clustering solution

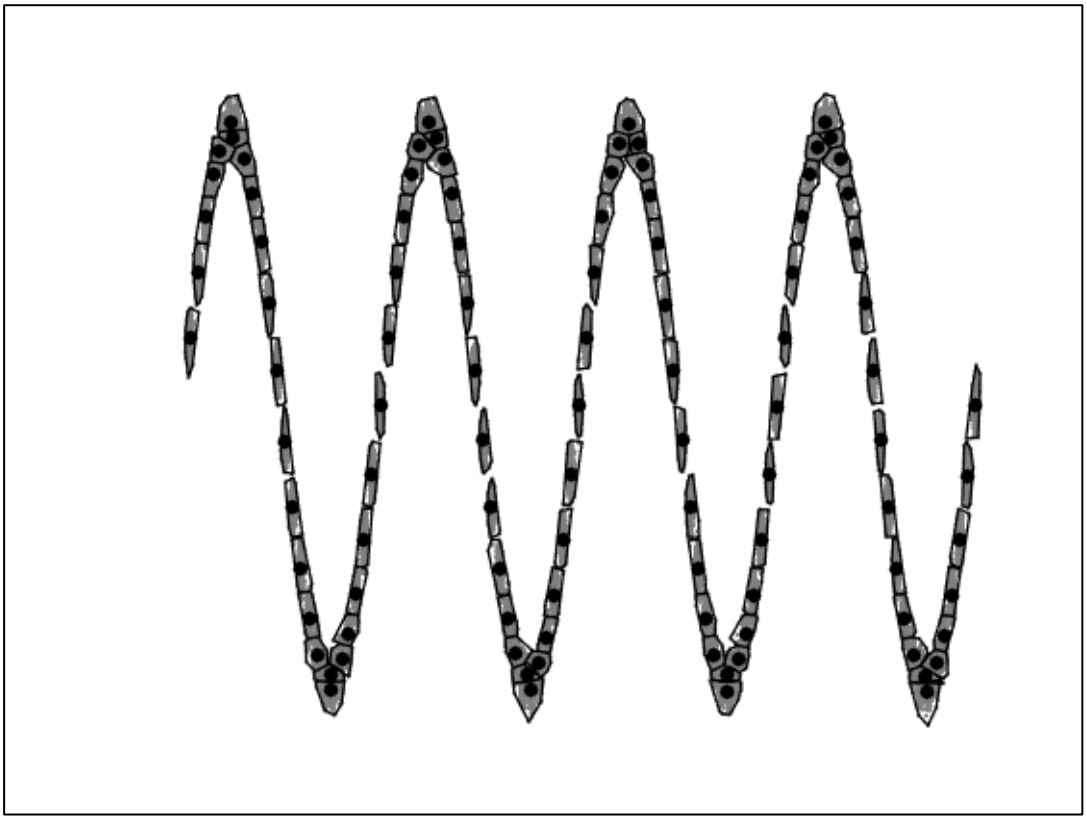


Figure 48: Birch 2 dataset solution

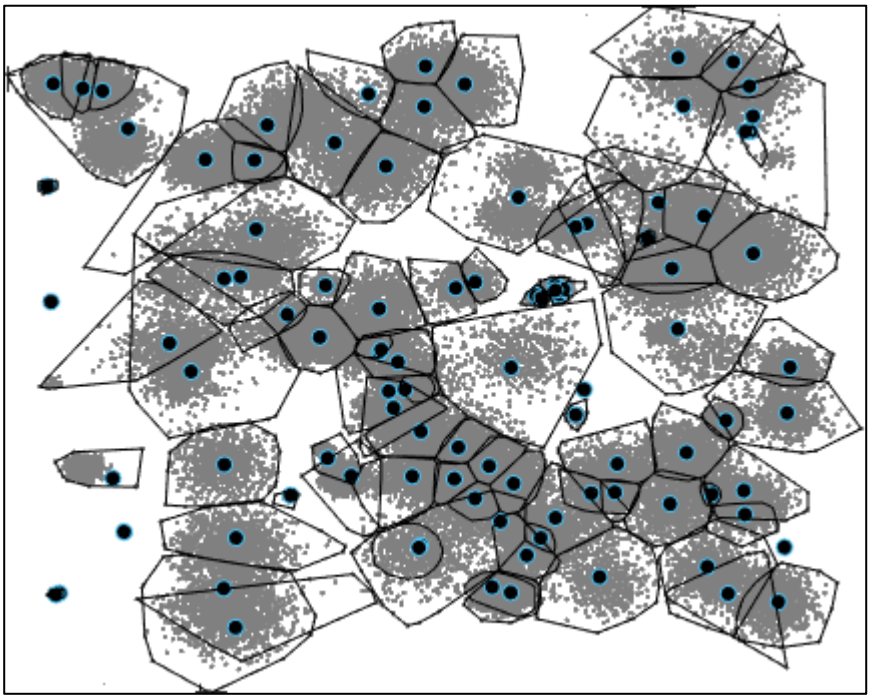


Figure 49: Birch 3 dataset solution

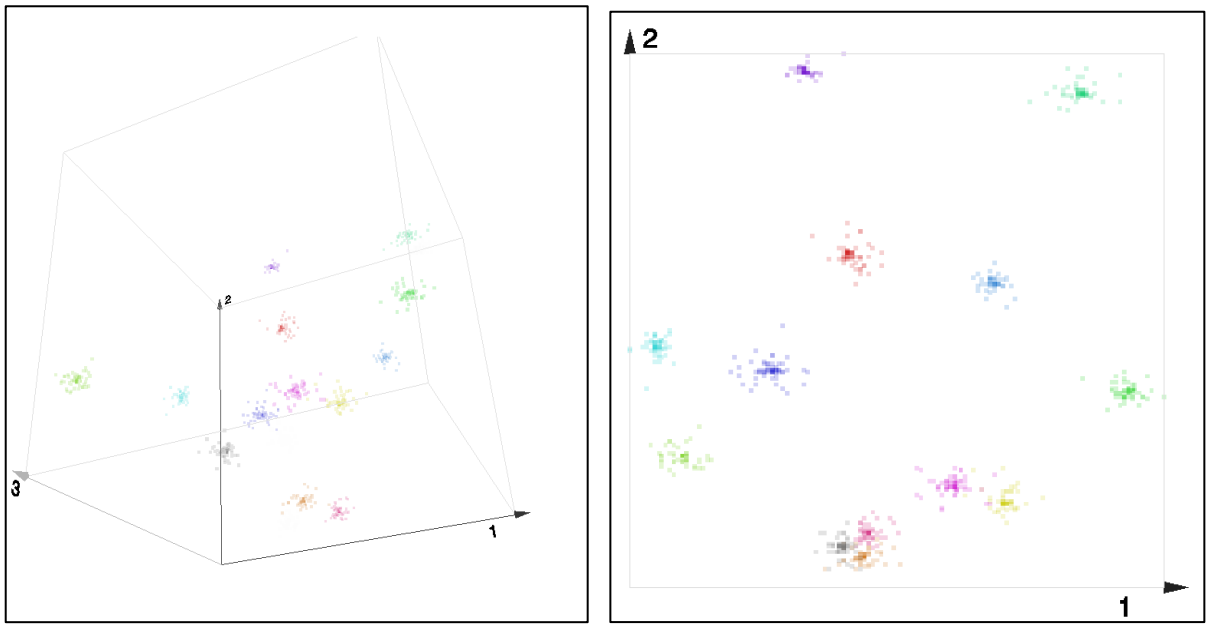


Figure 50: Dim032 dataset clustering results

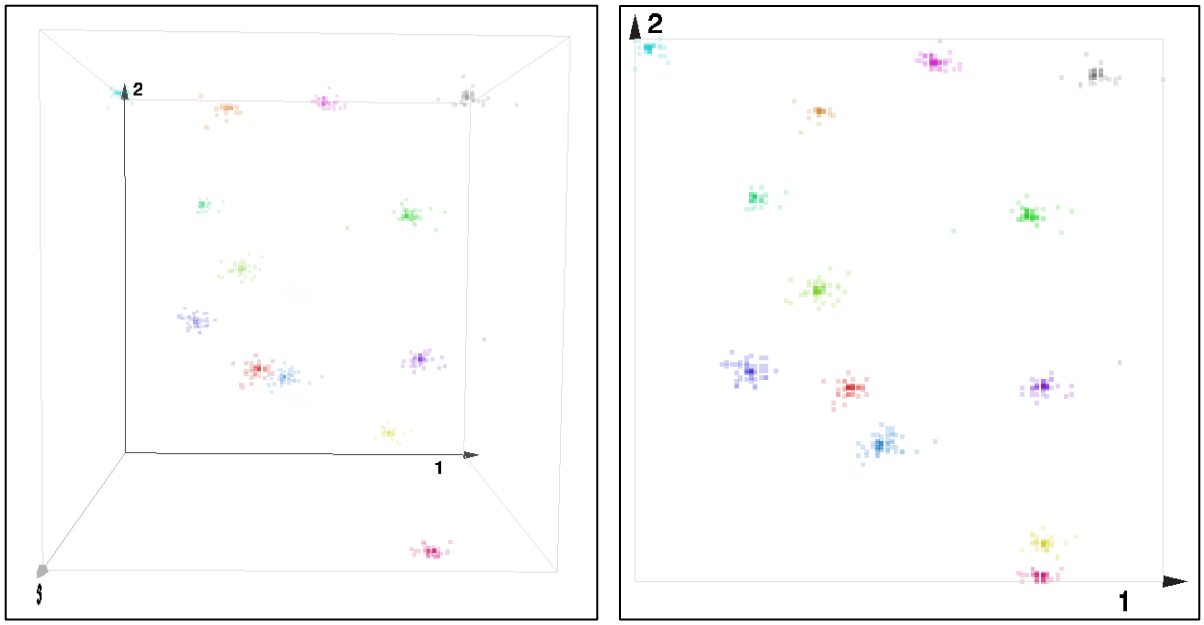


Figure 51: Dim064 dataset clustering results

Figures 50 and 51 show clustering solutions for 32-dimensional and 64-dimensional dataset. The image quality however is not particularly good as there are limitations to show multidimensional data on a 2-d surface. Table 5 gives results for the experiments.

Table 5: Clustering results on all the datasets

Dataset	# Clusters	CI	Success Rate	CSI
j1	3	0	100%	0.99
j2	6	0	100%	0.97
a1	20	0	100%	0.99
a2	35	0	100%	0.99
a3	50	0	100%	0.98
s1	15	0	100%	1.00
s2	15	0	100%	1.00
s3	15	0	100%	0.98
s4	15	1	100%	0.96
b1	100	0	100%	0.98
b2	100	0	100%	0.99
b3	100	7	72%	Not available
dim032	16	0	100%	1
dim064	16	0	100%	1

The algorithm was run on every dataset for 100 number of times. Each time, CSI and CI of the clustering solution was noted. The clustering solution having CI equal to zero was marked as a successful solution. But for datasets s4 and b3, the number of times CI value reached one and seven respectively was counted as a successful solution because one and seven were the lowest CI values respectively on these datasets. The percentage of successful solutions in Table 5 is the success rate. The success rate of getting CI =0 is 100% for all the datasets except for dataset b3 due to its complex structure and overlapping of data and for dataset s4 due to detection of a nested cluster. Multidimensional datasets are clustered perfectly as the clusters are well separated from each other. The CSI value for s4 is a bit low due to detection of a nested cluster in the dataset. The clustering has been successful for all the runs for all datasets except b3. Datasets n3 and n6 have CI=0 but CSI is extremely near one. It is not exactly one because of the reason explained below.

In dataset n3, both inner and the outer clusters are formed as Gaussian clusters. The area covered by nested cluster also has some points from the outer cluster. In ground truth solution, those points are assigned to the outer cluster. But once the algorithm runs, the points inside nested cluster which belong to outer cluster in ground truth, are assigned to the inner cluster. Therefore, even if the clustering solution is not wrong, CSI does not reach exactly to 1 in such cases. But CI is always 0.

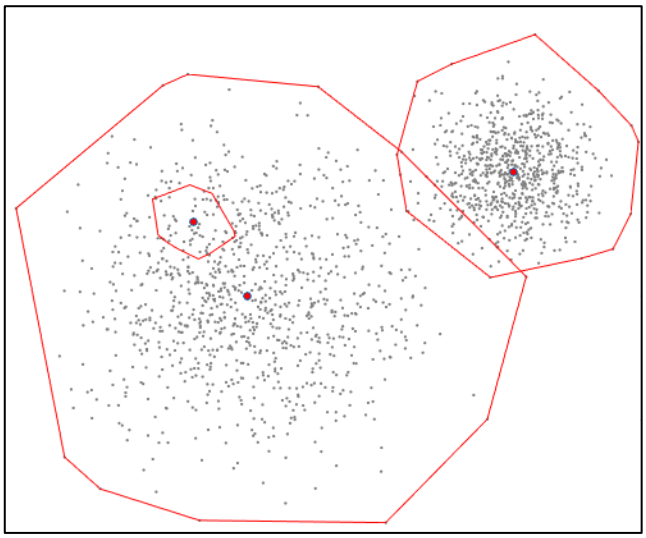


Figure 52: Dataset without data from nested cluster.

Figure 52 makes these arguments clear. It has only the location of nested cluster without actual vectors from the nested cluster. The points inside the nested cluster are the points belonging to outer cluster in ground truth. So, these points get assigned to the nested cluster after clustering algorithm is applied and this is the reason CSI does not reach exactly to 1.0.

Table 6.1 and 6.2 show the CI values of a few other density-based clustering algorithms like density peaks, fast density peaks. CI values in Table 6.1 and 6.2 are averaged over 100 executions.

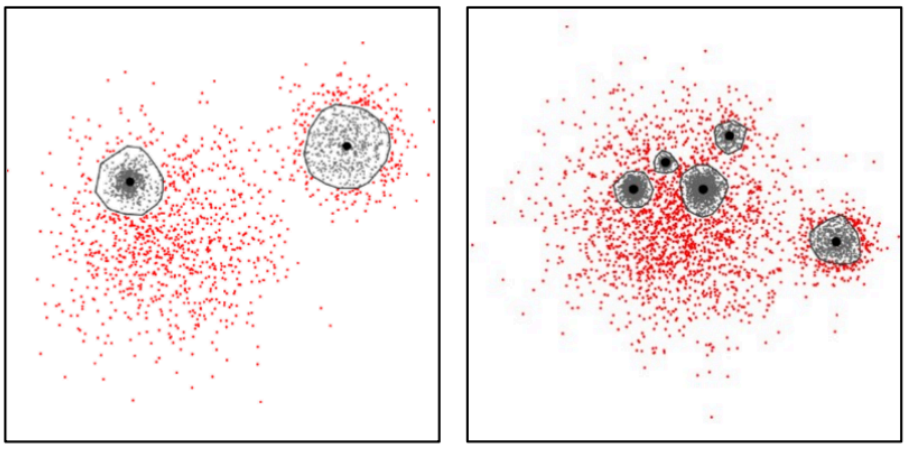
Table 6.1: CI values using various clustering algorithms on benchmark datasets

Dataset	n3	n6	a1	a2	a3
DBSCAN	0.90	0.60	0.00	0.00	0.00
Density Peaks	1.20	2.40	0.00	0.00	0.00
Fast Density Peaks	1.16	2.23	0.00	0.00	0.00
Jarkko version weighted RS	0.00	0.38	2.37	0.00	0.00
Weighted RS	0.00	0.00	0.00	0.00	0.00

Table 6.2: CI values using various clustering algorithms on benchmark datasets

Dataset	s1	s2	s3	s4	b1	b2
DBSCAN	0.00	0.00	0.00	0.00	0.10	0.00
Density Peaks	0.00	0.00	0.00	0.00	0.00	0.00
Fast Density Peaks	0.00	0.00	0.00	0.00	0.00	0.00
Jarkko version weighted RS	0.00	0.00	0.00	0.00	0.40	4.67
Weighted RS	0.00	0.00	0.00	1.00	0.00	0.00

For DBSCAN, choosing the values of ε and *minPts* is challenging. The ground truths for available data have the correct number of clusters, but the number of clusters in DBSCAN changes with its input parameters. This increases the difficulty in comparing clustering solutions of DBSCAN and with the ground truths. The DBSCAN was run with varying values of ε and *minPts* and the solutions with best CI value were chosen to be compared with the ground truths. We got solutions where CI reached 0 but this came at the expense of poor partitioning. Many points were labelled as noise. This means extremely poor value of CSI. Figure 53 shows the clustering solutions in DBSCAN where CI reached 0 for datasets n3 and n6.



ε = 0.009, minPts = 50 *ε = 0.012, minPts = 20*

Figure 53: Clustering solutions for n3 and n6 with carefully chosen values of ε and *minPts*. Adapted from [3].

Datasets of series A, S and birch1, birch2 do not have nested clusters. Most of the clustering algorithms work well with them. Main challenge is to get satisfactory results for datasets n3, n6 and birch3 where the datasets include complex nested clusters. CI values for density peaks and fast density peaks are adapted from [30]. Solutions obtained in density peaks algorithm are

shown in Figure 54. Density peak and fast density peak algorithms have almost similar performances. They however do not detect nested clusters.

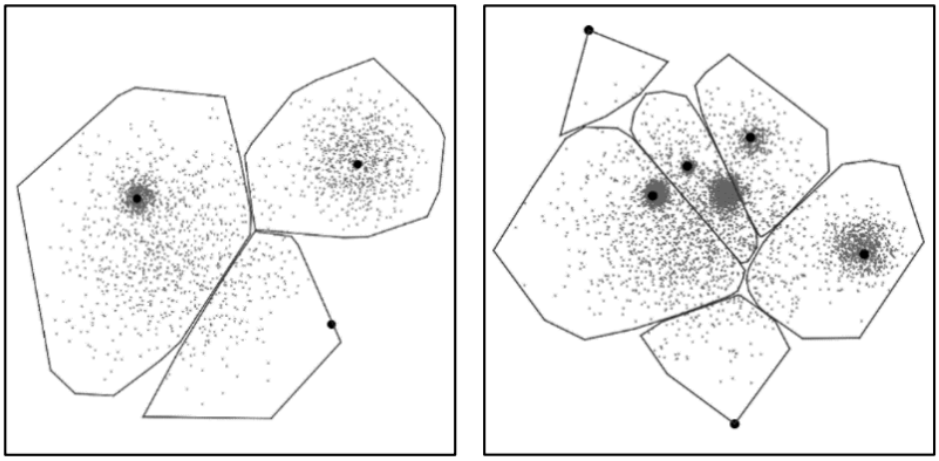


Figure 54: Clustering solutions for j1 and j2 using density peaks algorithm. Adapted from [3]

8. Conclusion

This thesis gives a detailed explanation on a unique density-based clustering algorithm. The algorithm uses weighted clustering. It is both centroids based as well as density based. It is shown that the algorithm works well even in case of non-nested clusters. Detecting nested clusters has been a challenging task and this work simplifies the same. Applications of density-based clustering can include server load balancing, balancing load on telephone networks, finding similarity between objects and applications in recommendation systems. For instance, cricket is a sport which is viewed by millions in countries like India. This puts tremendous amount of network load on video streaming servers in India. Let a person in China want to watch the same cricket match. There are very few cricket fans in China. Geographically, the person sitting in China is closer to streaming server in India, but through this algorithm, he might get directed to a streaming server in Australia where there is a bit lesser load than Indian server. In this case, traffic means the density of the cluster. Thus, it is an effective algorithm in simplifying density-based applications in machine learning.

One potential drawback of this algorithm is the weight initialization technique. Here, weights are initialized first in such a way that every cluster is assigned equal weight. Hence, first iteration of the algorithm works like a normal random swap algorithm. A better centroid weight initialization technique is a potential future work. Grid based approach is worth a try in this case. This would help in estimating the centroid weight as a pre-processing step and not as a part of actual clustering. Determining the grid size is a challenging process.

References

[1] Theodoridis, S. & Koutroumbas, Konstantinos. (2003). Pattern Recognition (2nd edition). 10.1016/B978-1-59749-272-0.X0001-2

[2] Rezaei, M., & Franti, P. (2016). Set matching measures for external cluster validity. *IEEE transactions on knowledge and data engineering*, *28*(8), 2173–2186. doi: 10.1109/tkde.2016.2551240

[3] Piiroinen, Jarkko (2019). Density-based clustering. MSc thesis, University of Eastern Finland. cs.uef.fi/sipu/pub/MSc_JarkkoPiiroinen.pdf

[4] Cao, K., Musa, I., Liu, J., & Zhang, Y. (2017). An adaptive density clustering algorithm for massive data. *2017 13th international conference on natural computation, fuzzy systems, and knowledge discovery (ICNC-FSKD)*. doi: 10.1109/fskd.2017.8393022

[5] El-sonbaty, Yasser & Ismail, Mohamed & Farouk, M. (2004). An efficient density-based clustering algorithm for large databases. 673- 677. 10.1109/ICTAI.2004.27.

[6] Pandit, S., & Gupta, S. (2011). A comparative study on distance measuring approaches for clustering. *International journal of research in computer science*, *2*(1), 29–31. doi: 10.7815/ijorcs.21.2011.011

[7] Gali, N., Mariescu-Istodor, R., Hostettler, D., & Fränti, P. (2019). Framework for syntactic string similarity measures. *Expert Systems with Applications*, *129*, 169–185. doi: 10.1016/j.eswa.2019.03.048

[8] Zhong, C., Miao, D., & Fränti, P. (2011). Minimum spanning tree-based split-and-merge: A hierarchical clustering method. *Information Sciences*, *181*(16), 3397–3410. doi: 10.1016/j.ins.2011.04.013

[9] Bano, Saima & Khan, Naeem. (2018). A Survey of Data Clustering Methods. International Journal of Advanced Science and Technology. 113. 10.14257/ijast.2018.113.14.

[10] Roy, Swarup & Bhattacharyya, Dhruba K. (2005). An Approach to Find Embedded Clusters Using Density Based Techniques. Lecture Notes in Computer Science. 3816. 523-535. 10.1007/11604655_59.

[11] Khan, Mohammad Mahmudur Rahman & Siddique, Md. Abu & Arif, Rezoana & Oishe, Mahjabin. (2018). ADBSCAN: Adaptive Density-Based Spatial Clustering of Applications with Noise for Identifying Clusters with Varying Densities. 10.1109/CEEICT.2018.8628138.

[12] Rezaei, M., & Franti, P. (2018). Real-Time Clustering of Large Geo-Referenced Data for Visualizing on Map. Advances in Electrical and Computer Engineering, 18(4), 63–74. doi: 10.4316/aece.2018.04008

[13] Fränti, P., & Sieranoja, S. (2019). How much can k-means be improved by using better initialization and repeats? *Pattern Recognition*, *93*, 95–112. doi: 10.1016/j.patcog.2019.04.014

[14] Gonzalez, T. F. (1985). Clustering to minimize the maximum inter-cluster distance. *Theoretical Computer Science*, *38*, 293–306. doi: 10.1016/0304-3975(85)90224-5

[15] Khan, S. S., & Ahmad, A. (2004). Cluster center initialization algorithm for k-means clustering. *Pattern recognition letters*, *25*(11), 1293–1302. doi: 10.1016/j.patrec.2004.04.007

[16] Kumara, A., Bharadwaj, H. S., & Ramaiah, N. S. (2019). A Survey on k-means algorithm centroid initialization. *SSRN Electronic Journal*. doi: 10.2139/ssrn.3372643

[17] Arthur, David & Vassilvitskii, Sergei. (2007). K-Means++: The Advantages of Careful Seeding. Proc. of the Annu. ACM-SIAM Symp. on Discrete Algorithms. 8. 1027-1035. 10.1145/1283383.1283494.

[18] Bezdek, James & Pal, Nikhil. (1995). Cluster Validation with Generalized Dunn's Indices. *Proceedings 1995 second New Zealand international two-stream conference on artificial neural networks and expert systems*. doi: 10.1109/annes.1995.499469

[19] Fränti, P. and J. Kivijärvi, "Random swapping technique for improving clustering in unsupervised classification", 11th Scandinavian conf. on image analysis (SCIA'99), Kangerlussuaq, Greenland, 407-413, 1999.

[20] Fränti, P. (2018). Efficiency of random swap clustering. *Journal of big data*, *5*(1). doi: 10.1186/s40537-018-0122-y

[21] Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei (1996). Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama M. (eds.). A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231.

[22] Sander, Jörg; Ester, Martin; Kriegel, Hans-Peter; Xu, Xiaowei (1998). "Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications". Data mining and knowledge discovery. Berlin: Springer-Verlag. 2 (2): 169 194. doi:10.1023/A:1009745219419

[23] Houle, M. E., Kriegel, H.-P., Kröger, P., Schubert, E., & Zimek, A. (2010). Can shared-neighbor distances defeat the curse of dimensionality? *Lecture notes in computer science scientific and statistical database management*, 482–500. doi: 10.1007/978-3-642-13818-8_34

[24] Singh, P., & Meshram, P. A. (2017). Survey of density-based clustering algorithms and its variants. *2017 International conference on inventive computing and informatics (ICICI)*. doi: 10.1109/icici.2017.8365272

[25] Zhao, Q., Shi, Y., Liu, Q., & Fränti, P. (2015). A grid-growing clustering algorithm for geo-spatial data. *Pattern Recognition Letters*, *53*, 77–84. doi: 10.1016/j.patrec.2014.09.017

[26] Rodriguez, A., & Laio, A. (2014). Clustering by fast search and find of density peaks. *Science*, *344*(6191), 1492–1496. doi: 10.1126/science.1242072

[27] S, N., Kashyap, M., & Bhattacharya, M. (2016). A variant of DBSCAN algorithm to find embedded and nested adjacent clusters. *2016 3rd International conference on signal processing and integrated networks (SPIN)*. doi: 10.1109/spin.2016.7566744

[28] M. Rezaei and P. Franti, "Set Matching Measures for External Cluster Validity" in IEEE Transactions on Knowledge & Data Engineering, vol. 28, no. 08, pp. 2173-2186, 2016. doi: 10.1109/TKDE.2016.2551240

[29] Fränti, P., & Sieranoja, S. (2018). K-means properties on six clustering benchmark datasets. *Applied intelligence*, *48*(12), 4743–4759. doi: 10.1007/s10489-018-1238-7

[30] Sieranoja, S., & Fränti, P. (2019). Fast and general density peaks clustering. *Pattern recognition letters*, *128*, 551–558. doi: 10.1016/j.patrec.2019.10.019

[31] Fränti, P., Kaukoranta, T., Shen, D.-F., & Chang, K.-S. (2000). Fast and memory efficient implementation of the exact PNN. *IEEE transactions on image processing*, *9*(5), 773–777. doi: 10.1109/83.841516

[32] Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, *31*(8), 651–666. doi: 10.1016/j.patrec.2009.09.011

[33] Fränti, P., Rezaei, M., & Zhao, Q. (2014). Centroid index: Cluster level similarity measure. Pattern Recognition, 47(9), 3034–3045. https://doi.org/10.1016/j.patcog.2014.03.017