Density-based clustering

Jarkko Piiroinen

Master's Thesis

UNIVERSITY OF
EASTERN FINLAND

School of Computing

Computer Science

May 2018

ITÄ-SUOMEN YLIOPISTO, Luonnontieteiden ja metsätieteiden tiedekunta, Joensuu
Tietojenkäsittelytieteen laitos
Tietojenkäsittelytiede

Piiroinen, Jarkko Mikael: Tiheyteen perustuva ryhmittely
Pro gradu -tutkielma, 71 s., ei liitteitä
Pro gradu -tutkielman ohjaaja: Professori Pasi Fränti
Toukokuu 2018

Tiivistelmä: Ryhmittely on eräs ohjaamattoman koneoppimisen muoto, jossa tavoitteena on muodostaa ryhmiä kohteiden samankaltaisuuksien perusteella. Tiheyteen perustuvat ryhmittelymenetelmät pyrkivät ryhmittelemään kohteita hyödyntämällä niiden tiheyttä. Tässä pro gradu -tutkielmassa tarkastellaan kirjallisuudessa usein viitattuja tiheyteen perustuvat menetelmiä ja sitä, miten ne soveltuvat erityisesti sellaisten datajoukkojen ryhmittelyyn, jotka sisältävät sisäkkäisiä eri tiheyksisiä klustereita. Tarkastellut menetelmät eivät sovellu kovin hyvin tällaisten datajoukkojen ryhmittelyyn, sillä ne tyypillisesti ryhmittelevät datajoukot ryhmiin, jotka eivät voi olla sisäkkäin. Lisäksi esitellään perinteiseen keskipistepohjaiseen ryhmittelyyn perustuva menetelmä, jossa klustereiden keskipisteille annetaan painot. Painoja hyödynnetään siten, että kohteiden etäisyydet keskipisteisiin lasketaan kerrottuna kunkin keskipisteen painolla. Näin keskipiste, jolla on matala paino, voi vetää puoleensa datavektoreita kauempaa kuin sellainen, jolla on korkea paino. Tämä puolestaan mahdollistaa sen, että klusterin sisällä olevat tiheämmät aliklusterit voidaan tunnistaa. Menetelmää testataan kokeellisesti *k*-means ja random swap -algoritmien avulla. Johtopäätöksenä on, että menetelmä toimii sisäkkäisiä klustereita sisältäville datajoukoille. Menetelmän haittapuolena on, että keskipisteiden painot on annettava algoritmille erillisenä parametrina.

Avainsanat: ryhmittely, tiheys, tiheyteen perustuvat ryhmittelymenetelmät, painotettu pituus

ACM-luokat (ACM Computing Classification System, 2012 version):

- **Computing methodologies~Cluster analysis**
- Information systems~Clustering

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

Piiroinen, Jarkko Mikael: Density-based clustering
Master's Thesis, 71 p., no appendices
Supervisor of the Master's Thesis: Professor Pasi Fränti
May 2018

Abstract: Clustering is an unsupervised form of machine learning where the objective is to form groups from objects based on their similarity. Density-based methods seek to group objects by utilizing the density of the objects in the clustering process. In this thesis, existing density-based clustering methods commonly cited in literature are examined in terms of their behavior with data sets that contain nested clusters of varying density. The existing methods are not ideal for such data sets, because they typically partition the data into clusters that cannot be nested. An approach based on traditional centroid-based clustering is introduced that assigns a weight to each cluster. The weights are then utilized by calculating the distances from data vectors to centroids by multiplying the distance by the centroid weight. As a result, a centroid with low weight will attract objects from further away than a centroid with higher weight. This allows dense clusters inside larger clusters to be recognized. The method is tested experimentally using the *k*-means and random swap algorithms. The experimental results show that this approach can be used to solve data sets with nested objects of varying densities. The drawback is that the centroid weights must be given as an extra parameter.

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

CR Categories (ACM Computing Classification System, 2012 version):

- **Computing methodologies~Cluster analysis**
- Information systems~Clustering

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
KM	*k*-means, a clustering algorithm
PDF	Probability density function
RS	Random swap, a clustering algorithm
SSE	Sum of squared errors

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 $\{x_1, x_2, \ldots, x_N\}$
C	Set of K cluster centroids $\{c_1, c_2, \ldots, c_K\}$
C_i	Set of data vectors in the cluster in the partition p_i.
P	Set of N cluster indices $\{p_1, p_2, \ldots, p_N\}$
W	Set of centroid weights $\{w_1, w_2, \ldots, w_K\}$
$\mathrm{dens}(i)$	Density of the cluster i represented by the centroid c_i
$\mathrm{d}(p, q)$	Distance between p and q

Contents

1 Introduction

Clustering (or *cluster analysis*) is a form of unsupervised machine learning where the objective is to group objects based on their similarity. (Bishop, 2007; Jain et al., 1999) Organizing the objects into groups allows us to draw conclusions based on the groups that were discovered. (Theodoridis & Koutroumbas, 2009)

Figure 1 shows a data set consisting of points that have been grouped into five clusters. Here, the distance between two points is used as the measure of similarity – the shorter the distance between them is, the more similar they are. The clusters are illustrated by drawing a *convex hull* for each cluster. A convex hull envelops all points belonging to a cluster.

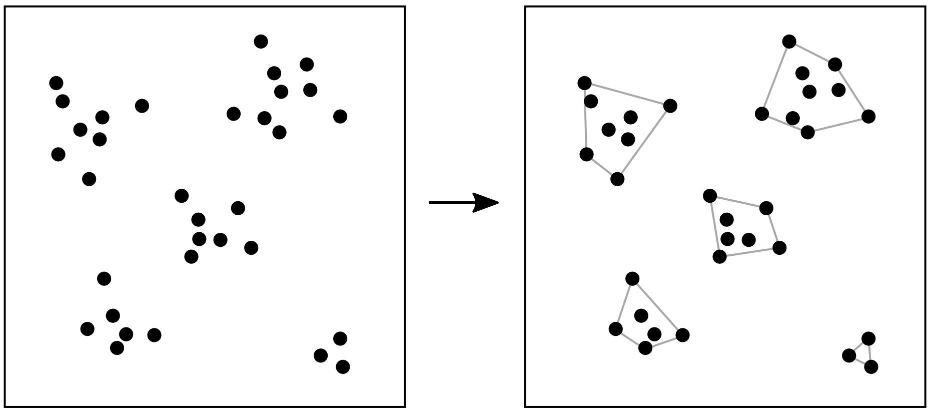


Figure 1. Clustering a set of points set into five groups, illustrated by convex hulls.

Clustering is particularly useful for exploratory data analysis and in machine learning contexts, including data mining and image segmentation. In such contexts, it is often important to make as few assumptions about the data as possible. This is where cluster analysis as a form of unsupervised machine learning is particularly appropriate. (Jain et al., 1999)

Clustering has numerous applications within computer science as well as across many different fields outside computing. In market research, clustering can be used to form groups out of customers that have with similar needs. In bioinformatics, it is utilized to identify gene groups with similar patterns of expression in order to study hereditary diseases. In chemistry, it may be used for classifying chemical compounds. There are countless other applications in fields including astronomy, psychiatry, archeology, and climate science. (Everitt et al., 2011; Kaufman & Rousseeuw, 2005)

In general, cluster analysis provides a powerful set of tools for analyzing data. Data analysis can be roughly divided into two categories: *exploratory* and *confirmatory* data analysis. (Jain et al., 1999; Theodoridis & Koutroumbas, 2009)

In exploratory (or *descriptive*) data analysis, we want to understand and learn more about the data and its structure without any predetermined hypotheses or models. (Jain, 2010) Clustering can therefore be used to describe and gain new insights from the data. It can also be used as an instrument to suggest new hypotheses that can be investigated later by other means.

In confirmatory (or *inferential*) data analysis, we seek to validate an existing hypothesis, model, or assumptions about the data. (Jain, 2010) A clustering result can therefore be analyzed using various statistical techniques, including analysis of variance, linear regression, and principal component analysis.

In addition to suggesting new hypotheses and testing existing ones, clustering may be used to make predictions based on groups. Theodoridis and Koutroumbas (2009) illustrate this with an example of patients with infectious diseases, and their reaction to drug therapy. First, all patients are clustered based on how they react to different medications. For a new patient, we can then determine the best medication by finding the closest cluster.

Clustering can also be used for data reduction. Sometimes a data set can be unnecessarily large for its intended purpose, and representative objects for each cluster would be sufficient. This is illustrated in in Figure 2, where sufficiently large clusters of map markers have been reduced to a single representative object each.

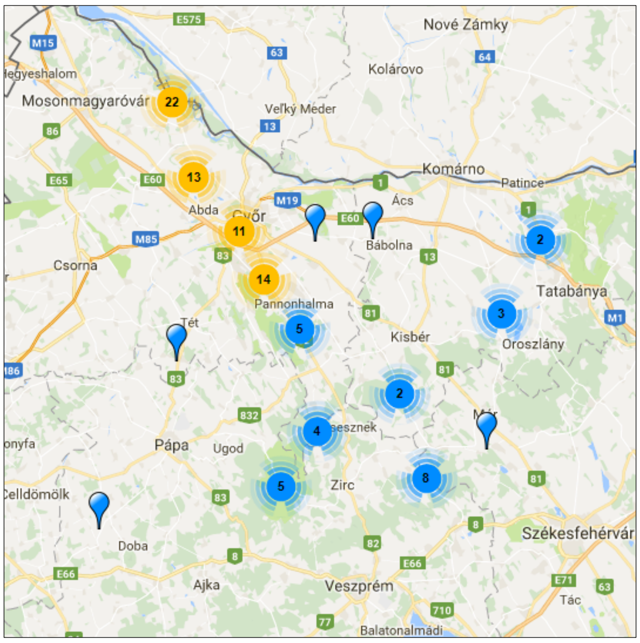


Figure 2. Clustering used to reduce the number of map markers.

Finally, sometimes we are not interested in discovering something new about the data, but rather we want to impose a new structure to it. For example, we might want to divide a country into telephone areas. (Kaufman & Rousseeuw, 2005) In such cases, the partitioning-based clustering methods can be used.

In this thesis, we are going to examine how to cluster data sets that contain nested groups of objects. Figure 3 shows an example with two nested clusters inside a single large cluster. The problem is how to detect each of the three clusters and group the points accordingly.

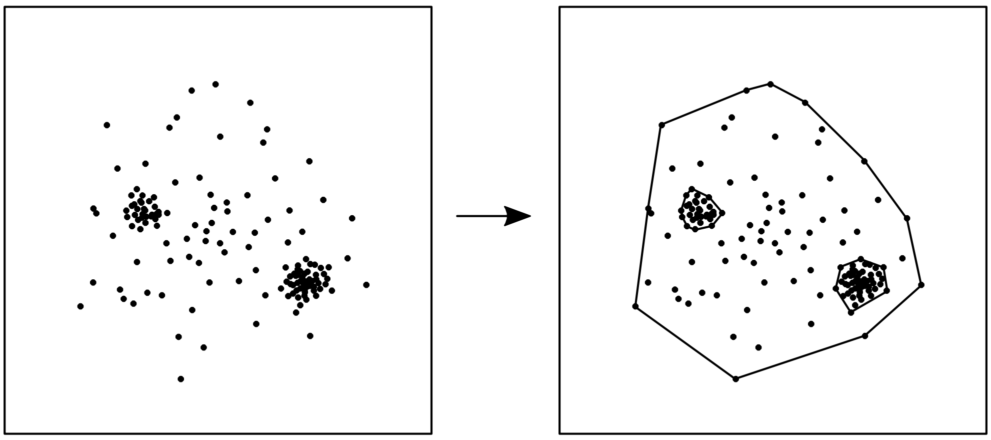


Figure 3. Clustering a data set with nested clusters.

There are several existing methods that can be used to attempt to solve such data sets. The problem is that while they are typically good at solving clusters of arbitrary sizes and shapes, they struggle with nested clusters. Instead, they usually require the groups to be at least somewhat separated from each other. Some of the common algorithms and their behavior with such data sets are examined in chapter 4.

An attempt to solve this problem is introduced in chapter 5. The idea is that cluster centers are given weights, and the weights are utilized when grouping objects: cluster centers that have low weight will then attract points from further away than those with higher weight. As a result, dense groups of objects would be detected inside large, sparse ones.

The experiments in chapter 6 show that this method works, and may have potential uses in any application where the problem is to find groups in data containing many nested clusters.

2 Clustering overview

A typical clustering task consists of several subtasks. Jain et al. (1999) list the following, the first three of which are illustrated in Figure 4:

(1) representation of patterns,
(2) definition of proximity measure,
(3) clustering or grouping,
(4) data abstraction (if needed), and
(5) output assessment (if needed).

Pattern representation refers to how the objects to be clustered will be represented as data structures. It includes the choice of the type, scale and the number of features to be given to the clustering algorithm. It may include *feature selection*, which is identifying the features to be used for clustering. It may also include *feature extraction*, which is creating new features by using transformations of the original features.

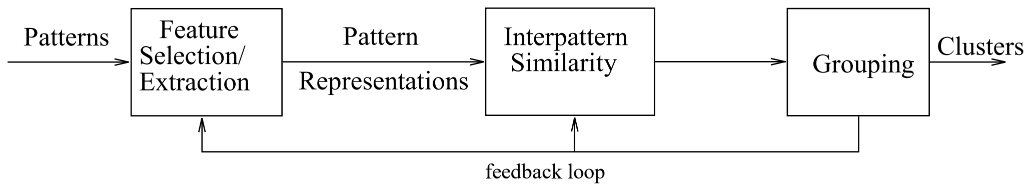


Figure 4. The different stages of clustering (Jain et al., 1999)

Proximity measure (sometimes called *similarity measure* or *distance measure*) refers to how similarity is measured between any two input data objects. Different kinds of measures are discussed further in the subchapter 2.3.

In the *clustering* or *grouping* step, the actual clustering is performed. There are many different methods that can be used in this step. These include *partitioning methods*, which produce a single clustering, and *hierarchical methods* which seek to produce a hierarchy of clusterings. Different clustering methods are discussed further in the subchapter 2.5.

Data abstraction means extracting a representation of the data set that is compact and easy for either a human or the computer (in the case of automatic analysis) to understand and make analyses of. Often this kind representation is in the form of *cluster prototypes*. Cluster prototypes are discussed further in the subchapter 2.1.

Finally, the *assessment* of the output is performed. The output can be assessed both from the data domain point of view (i.e. does the clustering output reveal something useful about the data, and if so, what?) as well as from the point of view of the clustering algorithm (i.e. does the algorithm produce a "good" result). Kaufman & Rousseeuw (2005) suggest that when interpreting the results, it should be based on insight into the original data, as well as having some experience with the clustering algorithm. They also advise trying multiple algorithms with the same data, as cluster analysis is mostly used for exploratory purposes. This makes sense if multiple algorithms are applicable to the data and the choice cannot be narrowed down to a single algorithm. It is also worth noting that different cost functions may produce different clusters for the same input. (Kaufman & Rousseeuw, 2005)

In Figure 4, the clustering process also includes an optional feedback loop. This means that the clustering "output could affect subsequent feature extraction and similarity computations". (Jain et al., 1999)

2.1 Basic concepts

A group of similar objects is called a *cluster*. Coming up with a precise definition of a cluster is difficult, as pointed out by Everitt et al. (2011). Intuitively, however, we can say that objects that are in the same cluster are more similar to each other compared to those in another cluster.

A cluster can also be defined by its internal cohesion or *homogeneity*, and external isolation or *separation*. (Everitt et al., 2011) These characteristics can be witnessed in Figure 5, which shows clusters in three different shapes.

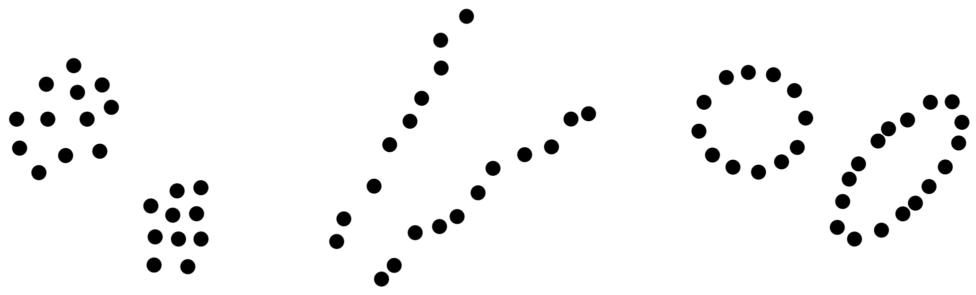


Figure 5. Compact (left), elongated (middle), and spherical clusters (right) (Theodoridis & Koutroumbas, 2009)

Although the three types of shapes are very different to one another, we can clearly see that each cluster is either relatively homogeneous (objects in each cluster are close to each other), or the clusters are separated (relatively distant from each other). However, coming up with precise mathematical definitions for the concepts of homogeneity and separation is difficult. (Everitt et al., 2011)

The chosen proximity measure, attributes, as well as the algorithm for the clustering each affect the clustering result. (Theodoridis & Koutroumbas, 2009) Figure 6 shows how a coarse clustering causes the data set to be grouped into two clusters (surrounded by dashed lines), whereas a finer clustering results in four groups (surrounded by solid lines).

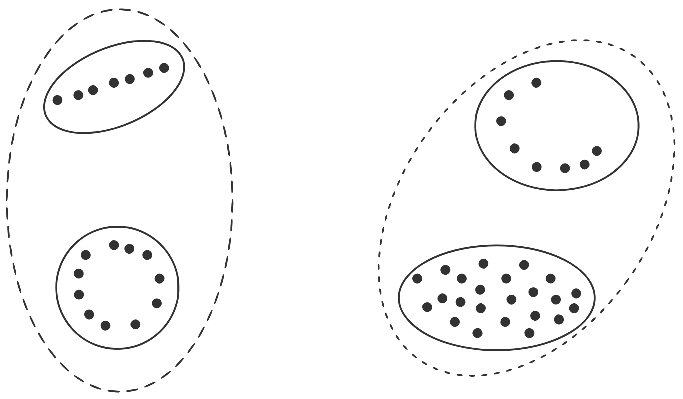


Figure 6. Coarseness of clustering can affect the result. (Theodoridis & Koutroumbas, 2009)

In this thesis, the objects we are clustering will be called *data vectors*. A *data set* $X = \{x_1, x_2, \dots, x_N\}$ is a collection containing N data vectors. Data vectors can be any type of objects, for example colors, animals, or movies. Data vectors have *attributes* that describe them. For example, a movie could have attributes such as title, director, or year of publication. For a point in Euclidean space, we can consider its coordinates to be its attributes.

The *dimensionality* D of a data set is the number of attributes its data vectors have. The data vectors we will look at in this thesis will be mostly two-dimensional points, in which case the value of D is 2. An example of such a data set is shown in Figure 7.

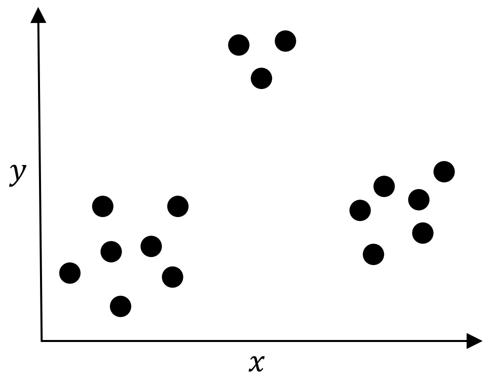


Figure 7. A small, two-dimensional data set with three clusters.

The *prototype* of a cluster is a value that represents that cluster. Its purpose is to characterize the cluster and the objects in it. A prototype can be part of the data set, but does not have to be.

The choice of prototype depends on the nature of the data. For data sets whose attributes are continuous, the prototype is often the *centroid* of the cluster – the mean of all data vectors in the cluster (Figure 8). In the case of categorical data where a mean or centroid is not definable, the prototype could be the *medoid* – the data vector with the shortest distance to the other members in the same cluster. (Everitt et al., 2011)

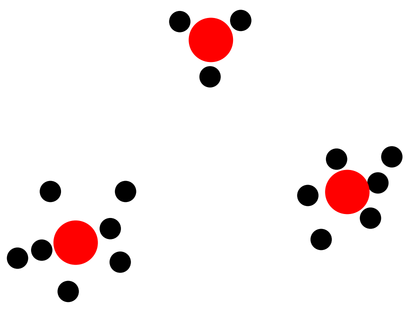


Figure 8. Cluster centroids.

In this thesis, we mostly deal with numerical data sets. Therefore, we use centroids as the cluster prototypes, and the notation $C = \{c_1, c_2, \dots c_K\}$ to denote the collection of centroids for the K clusters in the data set.

The *partitioning* $P = \{p_1, p_2, \dots p_N\}$ of a data set is the mapping of data vectors to prototypes: each data vector is associated with a prototype, and all data vectors that have the same prototype belong to the same *partition* or *cluster*. Figure 9 shows the previous data set partitioned into three clusters.

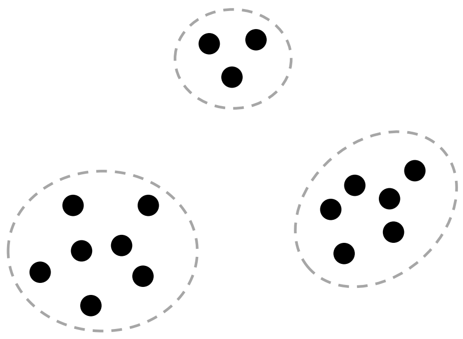


Figure 9. Partitioning of the data set.

Partitions and centroids are related to each other in the sense that it is possible to calculate the one we are missing based on the other. If we know the centroid locations, we can calculate the optimal partitioning by iterating over each data vector and finding its closest centroid. On the other hand, if we know the partitioning, we can calculate the optimal representative (centroid) for each partition by calculating the average of the data vectors in that partition. (Fränti & Kivijärvi, 2000)

2.2 Problem definition

Given only a data set, the clustering problem can be defined in many ways and may include several subproblems. For example, we could be interested in the number of clusters in a data set, where the clusters are located, or how to identify the most effective subset of attributes to be used for clustering. (Jain et al., 1999)

In this thesis, we assume that the number of clusters is known in advance and that our data sets consist of points in Euclidean space. Furthermore, we are interested in the locations of the clusters. Thus, we formulate our clustering problem as follows:

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

In general, this type of clustering problem has been shown to be NP-hard in *D*-dimensional Euclidean space. (Mahajan et al., 2009) Therefore, most algorithms focus on finding approximate solutions to the problem using various heuristics. There are several such algorithms for solving the problem, two of which (*k*-means and random swap) will be described in more detail later.

2.3 Distance measures

To be able to cluster data vectors, we must first define a measure for how similar a data vector is to another. In other words, we need a *distance function* that tells how close or far apart two given data vectors are.

The choice of a distance function depends heavily on the data, particularly the types of the attributes. Common for all distance functions according to Kaufman and Rousseeuw (2005) is that they must satisfy the following criteria:

(1) Distance between any two points is nonnegative
(2) Distance from a point to itself is zero
(3) Distance is symmetric; the order of the points does not matter
(4) Distance satisfies the *triangle inequality*; the distance from x to y to z is never less than the direct distance from x to z.

For *D*-dimensional points in a Euclidean space, an obvious choice for a distance function is the *Euclidean distance* between given two points x_1 and x_2:

$$\mathrm{d}(x_1, x_2) = \sqrt{\sum_{i=1}^{D} \left(x_1^i - x_2^i\right)^2} = \|x_1 - x_2\| \tag{1}$$

Euclidean distance is commonly used when the attributes of the data vectors are coordinates, color components or other numeric types that exist in a Euclidean space. In this thesis, the data sets used are two-dimensional data sets with integer attributes, so unless otherwise stated, Euclidean distance is assumed as the distance function.

Another type of distance function is *edit distance*, often used for strings. This is the number of *edit operations* needed to transform one string into the other. (Ristad &

Yianilos, 1998) A single edit operation may be the insertion, deletion, or substitution of a character. The precise definition of edit distance varies – typically some operations are excluded, and the cost of the operations may vary.

The most common form of edit distance is the *Levenshtein distance*, which allows insertions, deletions or substitutions, each having unit cost. For example, the Levenshtein distance between “depth” and “depend” is 3:

1. depth → dep**e**h (substitute ‘e’ for ‘t’)
2. depeh → depe**n** (substitute ‘n’ for ‘h’)
3. depen → depen**d** (insert ‘d’)

Other popular forms include the *longest common subsequence distance* (LCS) which only allows insertion and deletion operations, and *Hamming distance* where only substitutions are allowed. (Navarro, 2001) The LCS distance between “depth” and “depend” would be 5: substitutions are not allowed, so any characters not shared by the two strings must be deleted first. The Hamming distance for these strings is undefined, since it requires the strings to be of equal length.

Other popular distance functions used in clustering are *Jaccard distance* and *cosine distance*. The former measures the dissimilarity between two sets, while the latter measures the similarity between two data vectors by taking the cosine of the angle between the vectors. (Everitt et al., 2011)

2.4 Objective function

To evaluate the clustering, we need to define an *objective function*. Typically, the objective function is minimized: the lower the value of the objective function, the “better” the clustering.

The choice of objective function is the most important choice in the clustering method. (Fränti & Kivijärvi, 2000) The choice depends on both the data set and the domain of application. For example, if our aim is to minimize the within-cluster variance, we can use *sum of squared errors* (SSE) as the objective function. The SSE is calculated by first taking the squared distance of each data vector to its nearest centroid, then adding them all together:

$$\mathrm{SSE} = \sum_{i=1}^{N} \mathrm{d}\left(x_i, c_{p_i}\right)^2 \tag{2}$$

Here, N is the total number of data vectors in the data set, d is the distance function, x_i is the data vector, p_i is the index of the partition x_i belongs to, and c_{p_i} is the centroid of the partition p_i.

Minimizing SSE has the following properties. For a given partition P, the optimal centroids C can be calculated as follows:

$$c_j = \frac{\sum_{p_i=j} x_i}{\sum_{p_i=j} 1} \tag{3}$$

For given centroids C, the optimal partitioning is the one with the nearest centroid:

$$p_i = \underset{1 \le j \le K}{\arg\min}\, \mathrm{d}\left(x_i, c_{p_i}\right)^2 \tag{4}$$

2.5 Methods of clustering

A vast number of different clustering methods can be found in the literature. (Jain et al., 1999) As a result, there are also several ways of dividing the different approaches into categories. A common categorization has been to divide clustering methods into *hierarchical* and *partitioning* methods (Figure 10). Both divide further into more fine-grained categories. (Fraley & Raftery, 1998; Jain et al., 1999; Kaufman & Rousseeuw, 2005)

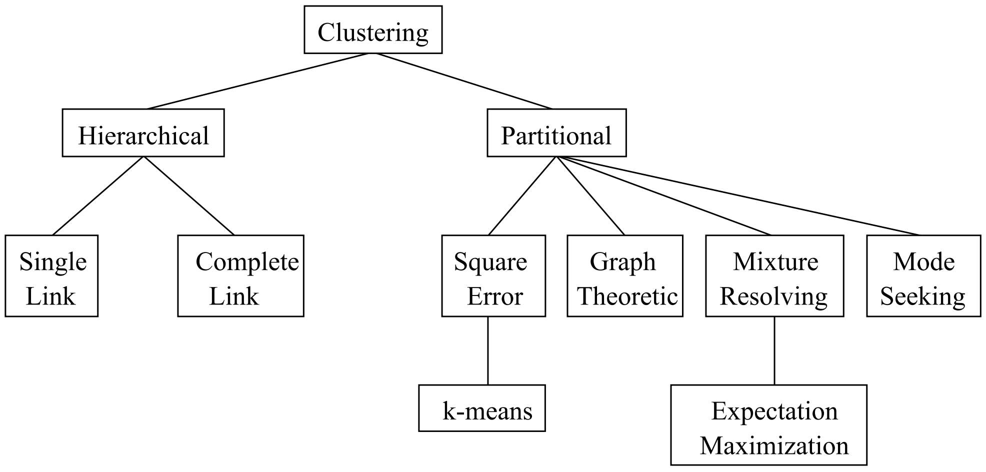


Figure 10. One way of categorizing different clustering methods. (Jain et al., 1999)

The key difference between the two main categories is that hierarchical methods produce “a sequence of partitions, each corresponding to a different number of clusters” (Fraley & Raftery, 1998) by either a divisive (splitting) or agglomerative (merging) approach, whereas partitioning methods start from an initial partitioning and then iteratively move data points from one group to another.

However, this binary division to hierarchical and partitioning methods is not perfect. For example, it is not immediately obvious which category density-based methods would belong to. Estivill-Castro (2002) has criticized such categorizations for being *misleading*, having *blurry boundaries*, and more specifically that the partitioning methods (as categorized by Jain) should more accurately be called *representation-based clustering*.

Fahad et al. (2014) proposed a categorization of clustering from the algorithm designer’s point of view. It attempts to divide the different clustering approaches based on the technical details of each approach, resulting in a tree shown in Figure 11.

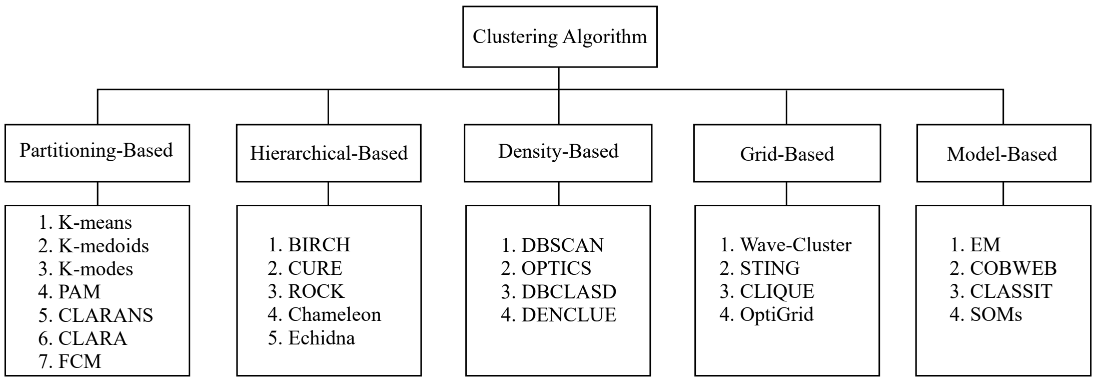


Figure 11. Clustering taxonomy from algorithm designer’s perspective. (Fahad et al., 2014)

This taxonomy considers density-based (among others) algorithms to be its own category. However, it could still be argued that a clustering algorithm can be both density-based and partitioning-based. An example of such a method is introduced in chapter 5.

In this thesis, we simply say that a *clustering method* defines the problem we want to solve. This problem is defined in terms of the objective function. A *clustering algorithm* solves the problem and produces a set of centroids C and the partitioning P as the results.

3 Data sets and algorithms

In this thesis, we will use two different algorithms to perform experiments on three different data sets. These data sets and algorithms are introduced in this chapter.

3.1 Data sets

The data sets are computer-generated and consist of two-dimensional Euclidean points spread across a varying number of Gaussian clusters. Each data set has slightly different characteristics that are meant to examine and illustrate the behavior of the algorithms. Since these data sets are synthetic, their ground truths (true centroid locations) are known. In the following figures, the centroids are marked with red filled circles. A summary of the properties of each data set is shown in Table 1.

Data set 1 contains fifteen Gaussian clusters and 15,000 data vectors in total. (Figure 12) It consists fixed-size clusters, each having the same density. The purpose of this data set is to examine the behavior of the different algorithms when the density of the clusters is constant.

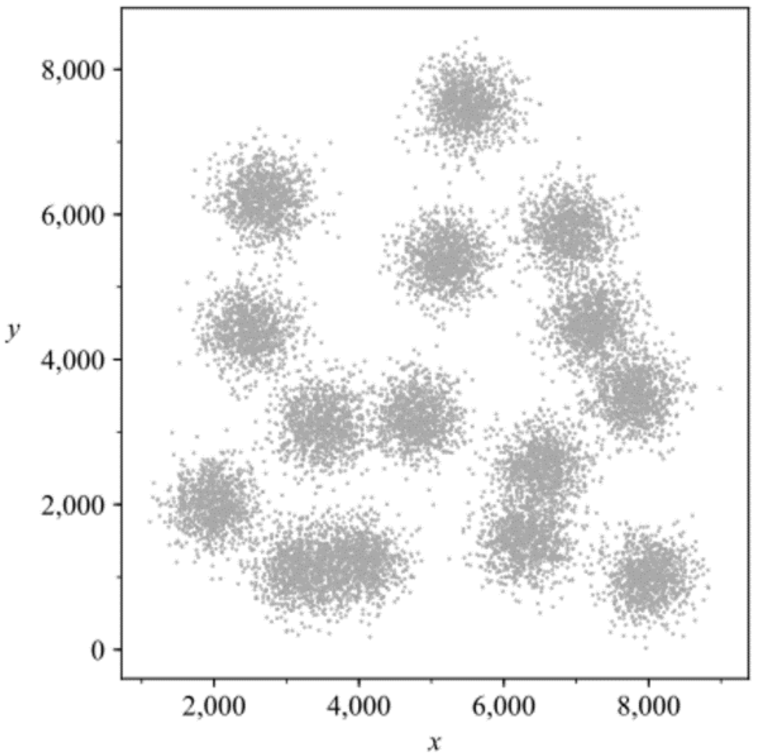


Figure 12. Data set 1 with fifteen clusters of equal density.

Data set 2 contains three Gaussian clusters and 2,250 data vectors in total. (Figure 13) It consists of one dense cluster enclosed by a larger, more widely spread cluster, as well as a third, medium-density cluster separated from the other two. The purpose of this data set is to study how the algorithms deal with some nesting and differences in cluster densities.

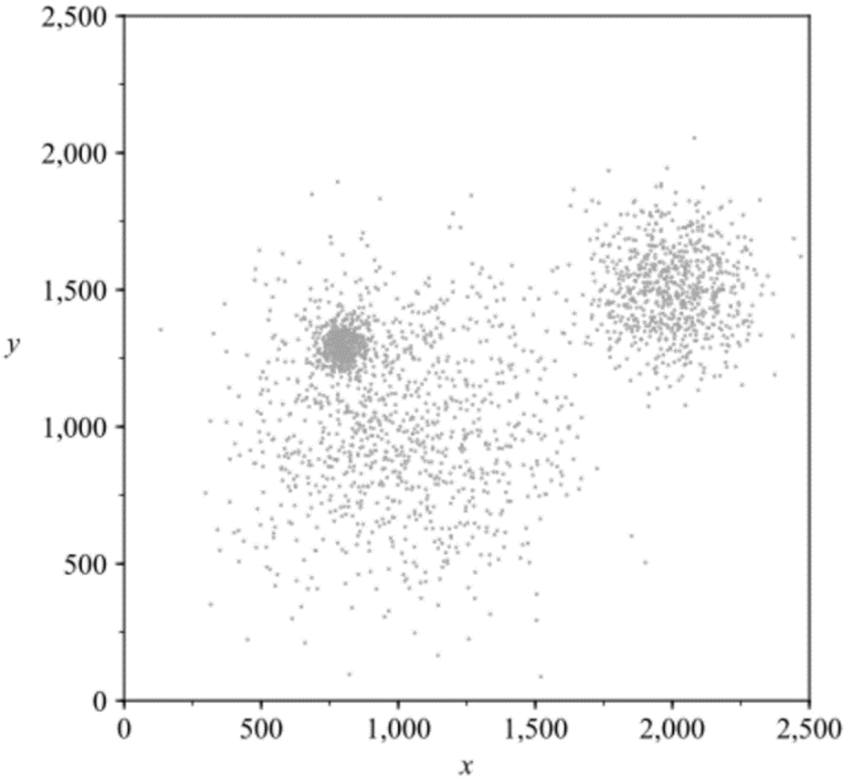


Figure 13. Data set 2 with three clusters and some nesting.

Data set 3 contains six clusters and 5,500 data vectors in total. (Figure 14) It consists of one very widely spread cluster that envelops four more dense clusters, as well as a fourth medium-density clusters that is separated from the other clusters. The purpose of this data set is to show how the algorithms behave with many nested clusters of varying densities and sizes.

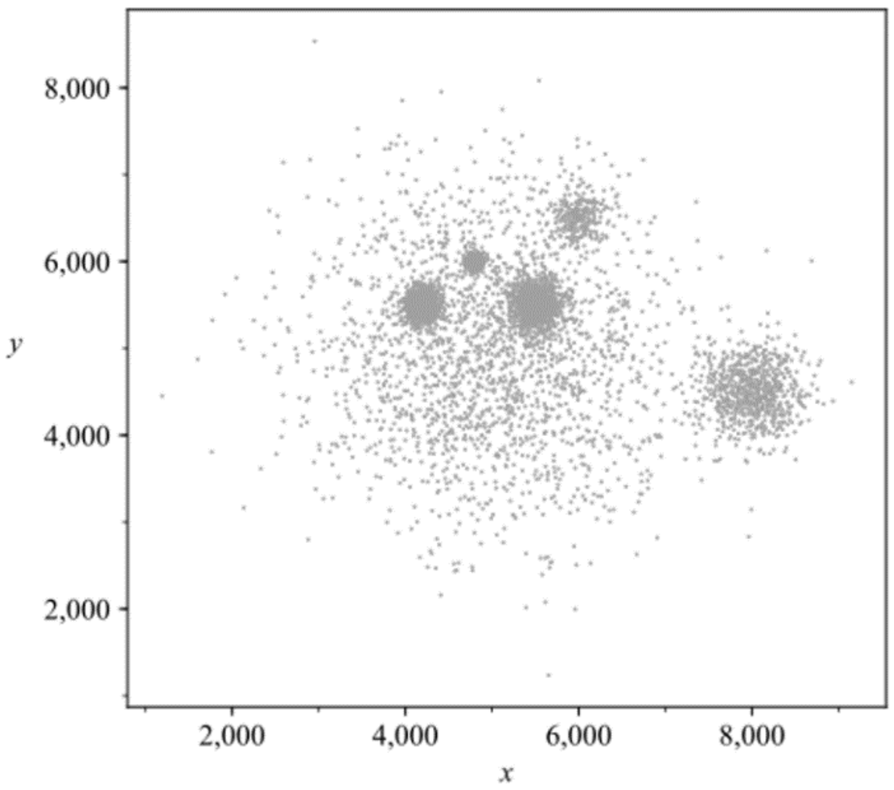


Figure 14. Data set 3 with several nested clusters.

Table 1. Data sets and their properties.

Data set	**Number of clusters** ***K***	**Number of data vectors** ***N***
1	15	15,000
2	3	2,250
3	6	5,500

3.2 Algorithms

Two different algorithms were chosen for this thesis: *k-means* and *random swap*. The algorithms were chosen based on a couple of factors: *k*-means represents an algorithm that is very widely used both as a stand-alone algorithm, and as part of other algorithms. It can also produce reasonable results. Random swap on the other hand represents a well-performing algorithm while still being relatively simple to implement. (Fränti & Kivijärvi, 2000)

3.2.1 *k*-means

The *k*-means algorithm is a simple, iterative, centroid-based clustering algorithm. It was introduced by MacQueen in 1967. Since then, its variations have been used in numerous machine learning applications.

The algorithm is given a data set $X = \{x_1, x_2, \dots, x_N\}$ and the number of clusters K as parameters, and it returns the centroid locations $C = \{c_1, c_2, \dots, c_K\}$ and the partitioning $P = \{p_1, p_2, \dots, p_N\}$ of the data points as the results. (Figure 15)

```
**K-means**$(X, P, C)$: → $(C, P)$
REPEAT
  $C_{\text{prev}} \leftarrow C$
  FOR $i := 1$ TO $N$ DO
    $p_i \leftarrow \text{FindNearestCentroid}(x_i, C)$
  FOR $j := 1$ TO $K$ DO
    $c_j \leftarrow \text{CalculateCentroid}(X, P, j)$
UNTIL $C = C_{\text{prev}}$
```

Figure 15. Pseudo code for the *k*-means algorithm.

The algorithm starts with an initial solution, and iterates until it cannot find any more improvement. It consists of two main steps:

1. Partitioning step
2. Centroid step.

In the partitioning step, the algorithm will assign all data vectors to a partition by finding the nearest centroid for each. The nearest centroid to a given point is found by calculating the distance from the point to all centroids – the nearest centroid is the one that has the lowest distance. (Figure 16)

```
**FindNearestCentroid**$(x, C)$: $\rightarrow i_{\text{nearest}}$
$d_{\text{nearest}} \leftarrow \infty$
FOR $i \coloneqq 1$ TO $K$ DO
  $d \leftarrow \mathrm{d}(x, c_i)$
  IF $d < d_{\text{nearest}}$ THEN
    $d_{\text{nearest}} \leftarrow d$
    $i_{\text{nearest}} \leftarrow i$
RETURN $i_{\text{nearest}}$
```

Figure 16. Pseudo code for finding the nearest centroid.

In the centroid step, the algorithm will calculate the new cluster centroids as the mean vector of each partition calculated in the partitioning step. The mean vector is calculated by summing up the all the vectors in the partition, then dividing the components of the result vector by the number of vectors in the partition. (Figure 17)

```
**CalculateCentroid**$(X, P, j)$: $\rightarrow c$
sum[1..$D$] $\leftarrow 0$
count $\leftarrow 0$
// Calculate the sum of all vectors in the partition
FOR $i \coloneqq 1$ TO $N$ DO
  IF $p_i = j$ THEN
    sum $\leftarrow$ sum $+ x_i$
    count $\leftarrow$ count $+ 1$
// Calculate the mean
$c \leftarrow$ sum / count
RETURN $c$
```

Figure 17. Pseudo code for calculating the cluster centroid.

There are several ways to initialize *k*-means. One of the most common ways is to pick random data vectors as the initial centroids. This is the second initialization method proposed by MacQueen (1967). Other popular techniques are examined and recommended by Celebi et al. (2013).

The advantage of *k*-means is its simplicity: It is easy to implement while providing relatively good results with many data sets. Jain (2010) also includes its "efficiency, and empirical success" as some of the reasons why it has gained such popularity. Like with many centroid-based clustering algorithms, the number of clusters must be given to *k*-means as a parameter. We also assume that SSE is a sensible objective function for the data.

The disadvantage of *k*-means is its high dependency on the initial partitioning. (Fränti & Kivijärvi, 2000; Jain, 2010; Celebi et al., 2013) Therefore, it tends to converge to a local optimum instead of the global one. The clustering quality can be improved by repeating the *k*-means several times. (Kinnunen et al., 2011) Each time, the algorithm is initialized with a different random solution, and the result that provides the smallest objective function value is kept as the final solution.

Figure 18 shows a comparison of a single run of *k*-means, and *k*-means repeated 100 times on data set 1. In the latter case, the best result among all *k*-means runs is retained. With a single run, the algorithm can correctly identify many of the clusters but leaves plenty of room for improvement. The result will be significantly improved after 100 repeats, but at the cost of higher runtime.

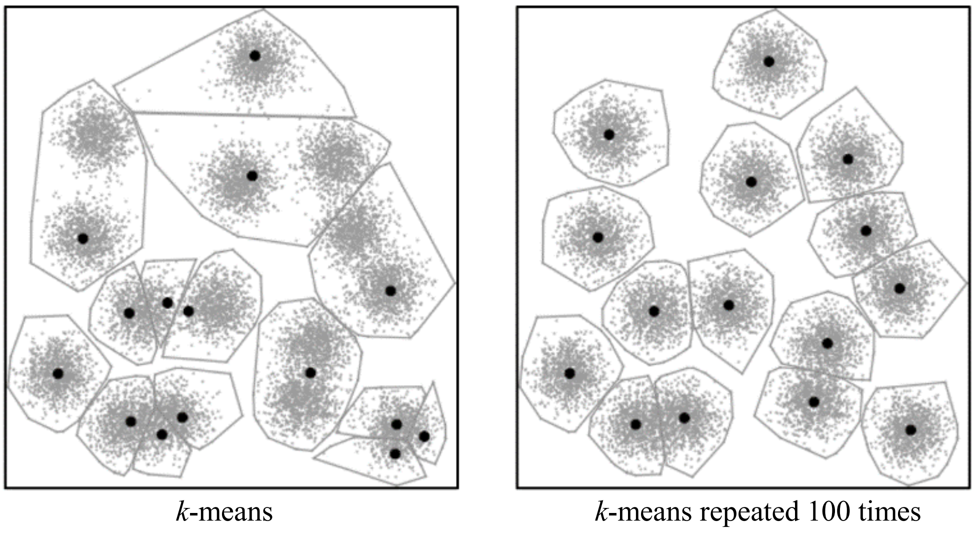


Figure 18. A single run of *k*-means and repeated *k*-means on data set 1.

The *k*-means algorithm suffers from the problems of parametric and partitioning algorithms in general. Specifically, the number of clusters must be known in advance, and the algorithm typically produces clusters that are convex. (Kriegel et al., 2011)

3.2.2 Random swap

Random swap (also known as *randomized local search*) is a swap-based clustering algorithm introduced by Fränti & Kivijärvi (2000). It works by removing a random centroid from the current solution and placing it on a random data vector. (Fränti,

2018) The result is then fine-tuning the result using k-means. If the clustering quality improved after this *trial swap*, the new centroids and/or partitioning is accepted.

Like k-means, random swap is an iterative, centroid-based clustering algorithm. Given a data set $X = \{x_1, x_2, \dots, x_N\}$ and the number of clusters K as parameters, it returns the centroid locations $C = \{c_1, c_2, \dots, c_K\}$ and the partitioning $P = \{p_1, p_2, \dots, p_N\}$ of the data points as the results. (Figure 19)

```
**RandomSwap**$(X, P, C)$: $\rightarrow (C, P)$
REPEAT $T$ TIMES
    $(C_{\text{new}}, j) \leftarrow$ SwapCentroid$(X, C)$
    $P_{\text{new}} \leftarrow$ LocalRepartition$(X, C_{\text{new}}, P, j)$
    $(C_{\text{new}}, P_{\text{new}}) \leftarrow$ K-means$(X, C_{\text{new}}, P_{\text{new}})$
    IF $f(C_{\text{new}}, P_{\text{new}}) < f(C, P)$ THEN
        $(C, P) \leftarrow (C_{\text{new}}, P_{\text{new}})$
RETURN $(C, P)$
```

Figure 19. Pseudo code for the random swap algorithm. (Fränti, 2018)

Like k-means, the algorithm starts with an initial solution (such as using random data points as the initial centroids) and iterates T times. It consists of three main steps:

1. Random swap
2. Local repartition
3. Iteration by k-means.

In the *random swap* step, a centroid is chosen randomly from the current set of centroids and it is moved to the location of a random data vector. (Figure 20) With some luck, this will cause a centroid to move from a *centroid-rich* to a *centroid-poor* area.

```
**SwapCentroid**$(X, C)$: $\rightarrow (C, j)$
$j \leftarrow$ Random$(1, k)$
$i \leftarrow$ Random$(1, N)$
$c_i \leftarrow x_i$
RETURN $(C, j)$
```

Figure 20. Swap centroid to the location of a random data vector.

In the *local repartition* step, all the data vectors allocated to the swapped centroid will be re-allocated to their nearest centroids. (Figure 21) Furthermore, the data vectors near the swapped centroid must also be re-allocated, since it is possible that they will be attracted by the swapped centroid.

```
**LocalRepartition**$(X, C_{\text{new}}, P, j)$: $\rightarrow P$
FOR $i := 1$ TO $N$ DO
  IF $p_i = j$ THEN
    $p_i \leftarrow$ FindNearestCentroid$(x_i, C)$
FOR $i := 1$ TO $N$ DO
  IF $\mathrm{d}(x_i, c_j) < \mathrm{d}(x_i, c_{p_i})$ THEN
    $p_i \leftarrow j$
RETURN $P$
```

Figure 21. Re-allocation of data vectors to centroids.

Finally, *iteration by k-means* is performed. This will fine-tune the result we achieved in the previous steps and will find a new local optimum, given the new configuration of centroids/partitioning. Fränti and Kivijärvi (2000) noted that only two iterations of *k*-means were enough to achieve high quality clustering.

At this point, it is not guaranteed that the result is any better than the clustering *before* the swap (in fact, it could be worse). Therefore, the new candidate centroids/partitioning is accepted only if it produces a better clustering determined by the objective function. For example, if SSE is used as the objective function, then the SSE of the new candidate result must be lower than the SSE before the trial swap. If it is not, the result will be discarded, and a new trial swap will be performed.

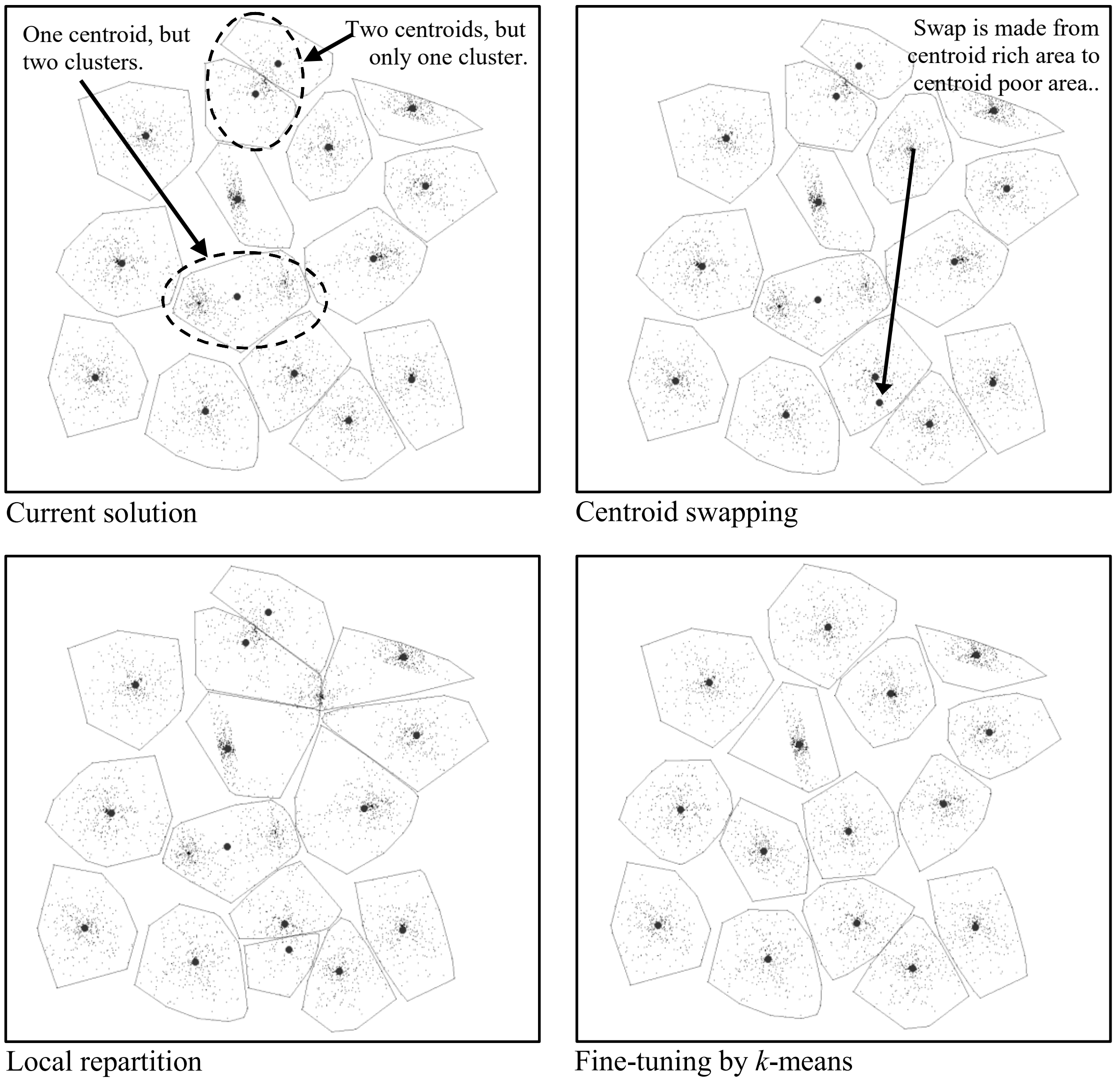

One centroid, but two clusters.
Two centroids, but only one cluster.
Swap is made from centroid rich area to centroid poor area..
Current solution
Centroid swapping
Local repartition
Fine-tuning by *k*-means

Figure 22. Demonstration of centroid swapping. (Fränti, 2018)

The advantage of random swap is that implementing it is easy, and it is also efficient: Fränti and Kivijärvi (2000) showed that the algorithm outperforms *k*-means and gives results that are competitive with more complex techniques such as *genetic algorithms* and *tabu search*. Unlike *k*-means, the algorithm is not dependent on the initialization method: Centroid swapping enables the algorithm to fix situations where *k*-means would get stuck, such as the highlighted regions in Figure 22 (current solution). As a result, random swap can detect the clusters in data set 1, as seen in Figure 23. To achieve similar results with *k*-means, the initial centroids have to be relatively close to the correct locations, or the algorithm has to be repeated multiple times, retaining the best solution over all runs.

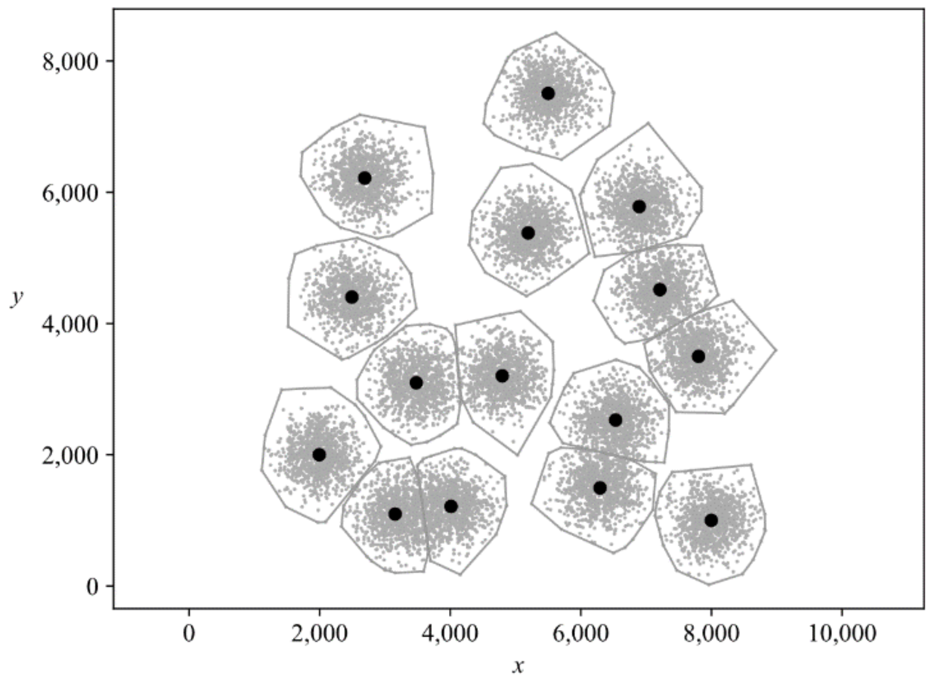


Figure 23. Random swap can solve data set 1.

4 Density-based clustering

Density-based clustering refers to identifying groups of objects that form contiguous, high-density regions in the data space. (Kriegel et al., 2011; Rodriguez & Laio, 2014) Typical for these *density-based clusters* is that they are separated from one another by regions of low density. The low-density regions surrounding the high-density regions are often considered to be *noise*.

The notion of density-based clustering was explored by Jain and Dubes (1988) in what they call *clustering by density-estimation and mode seeking*. In their approach, the high-density regions are called *modes*. Each mode corresponds to a cluster center, and data vectors are assigned to the cluster that has its center nearest to them. Given any data vector x, the number of data vectors inside a small region of space around x is used as an estimate of density.

We can get a rough idea of the density of the distribution of a data set by constructing a histogram. (Ester et al., 1996) This is done by partitioning the data into a grid of non-overlapping cells: Grid cells with high frequency of data vectors are potential cluster centers, and the low-density areas between the clusters correspond to the minima in the histogram. (Jain & Dubes, 1988)

Figure 24 shows a histogram of data set 3. The four clusters nested by the large, widely spread cluster can be clearly recognized because of their high density. Since the density of these clusters is significantly higher than the sparse cluster that nests them, the sparse cluster can be considered background noise. On the other hand, the larger, sparse clusters are also faintly noticeable, since their density is also higher relative to their neighborhood.

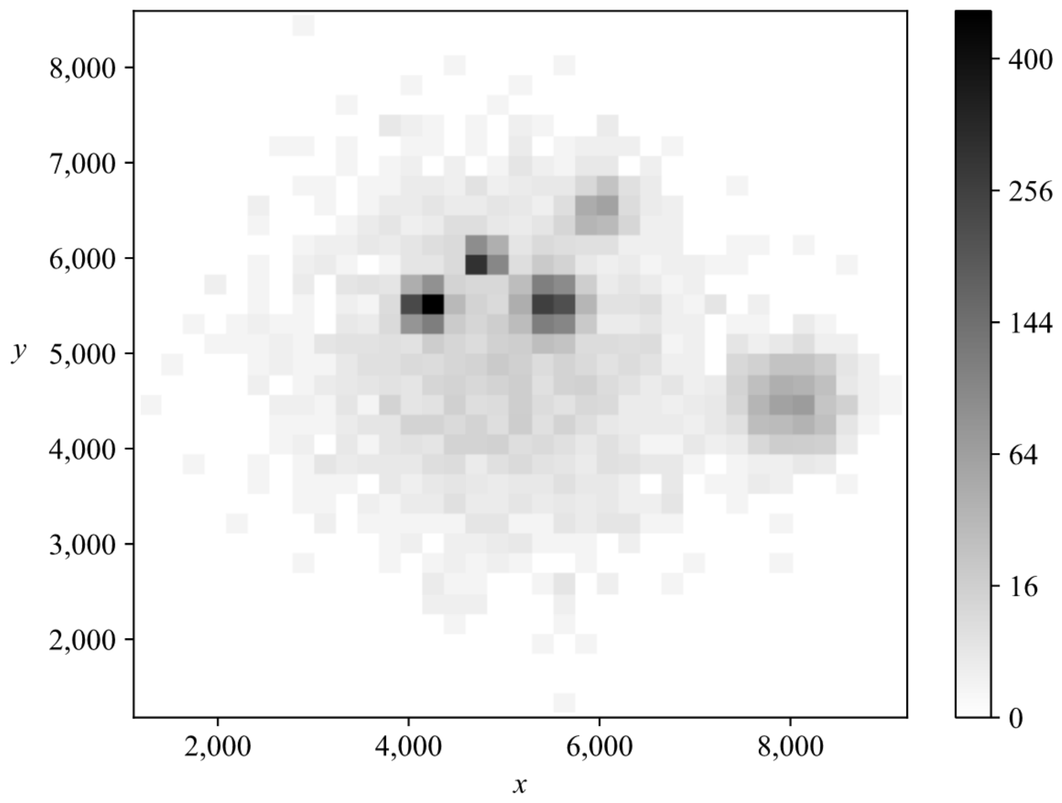


Figure 24. A histogram of data set 3, showing the density of the distribution of the data.

Density-based clustering is often defined to be a *non-parametric* approach to clustering, meaning that the number of clusters does not have to be given as an additional parameter. It usually also does not require assumptions about the density distribution of the data set, or the variance within clusters. (Kriegel et al., 2011) These kinds of methods are typically able to detect clusters that have arbitrary shapes and sizes, since the clusters can grow in any destination driven by density. Figure 25 shows of clusters detected by the density peaks algorithm.

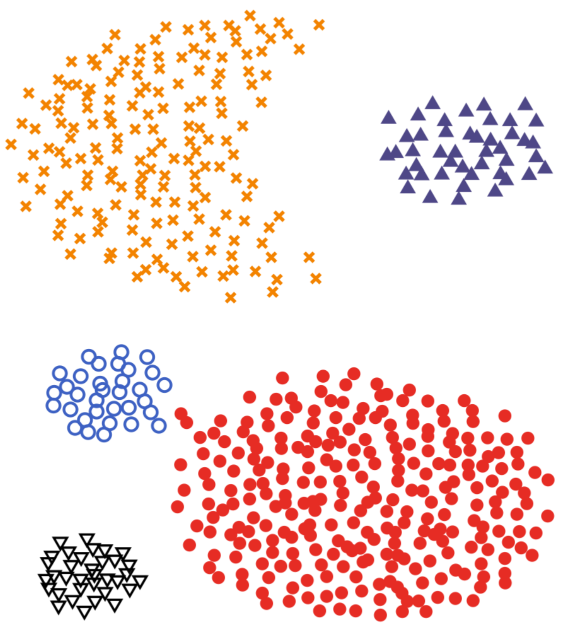


Figure 25. Density peaks can find clusters of arbitrary shapes and sizes. Adapted from Rodriguez & Laio (2014).

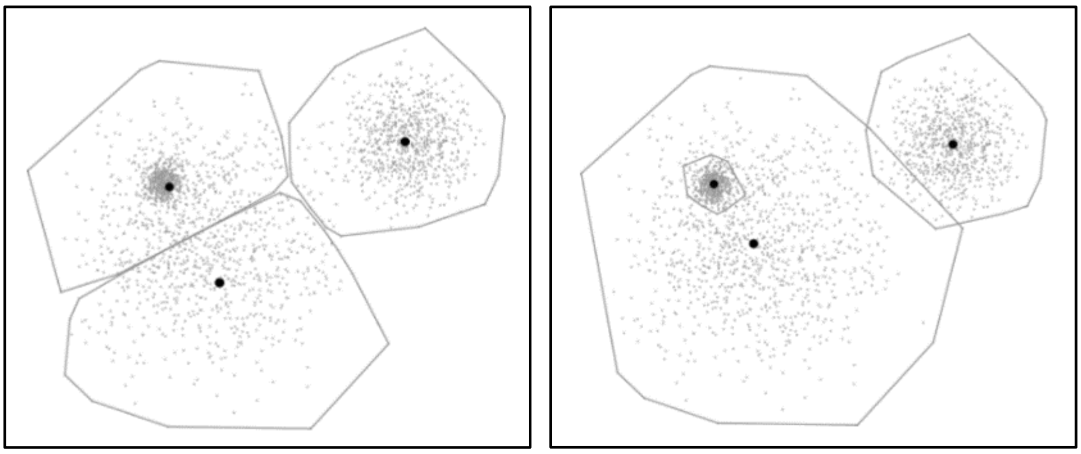
However, density-based clustering methods do not necessarily have to be non-parametric. Melnykov et al. (2014) for instance calls this category of methods *non-parametric clustering* instead of density-based clustering, although they also note that clusters in these methods are traditionally *associated with density bumps or modes*.

In this thesis, we consider density-based methods to be any clustering methods that make use of the density of the data as the primary feature in clustering. As seen in chapter 5, it is possible to utilize density even if we use parametric methods such as *k*-means.

4.1 Motivation

For Euclidean data vectors, partitioning algorithms such as *k*-means and random swap will partition the data space into non-overlapping groups. This follows naturally from the fact that each data vector is assigned to the cluster represented by the nearest centroid. As a result, nested clusters (i.e. clusters within clusters) are generally not detected.

An example of this is shown in Figure 26. When the sum of squared errors (SSE) is used as the object function, random swap can detect the rough locations of the cluster centers. However, the partitioning it produces is far from perfect. Crucially, it fails to group together the points belonging to the small, dense cluster within the larger, sparse cluster. Instead, the large cluster is split roughly in half.

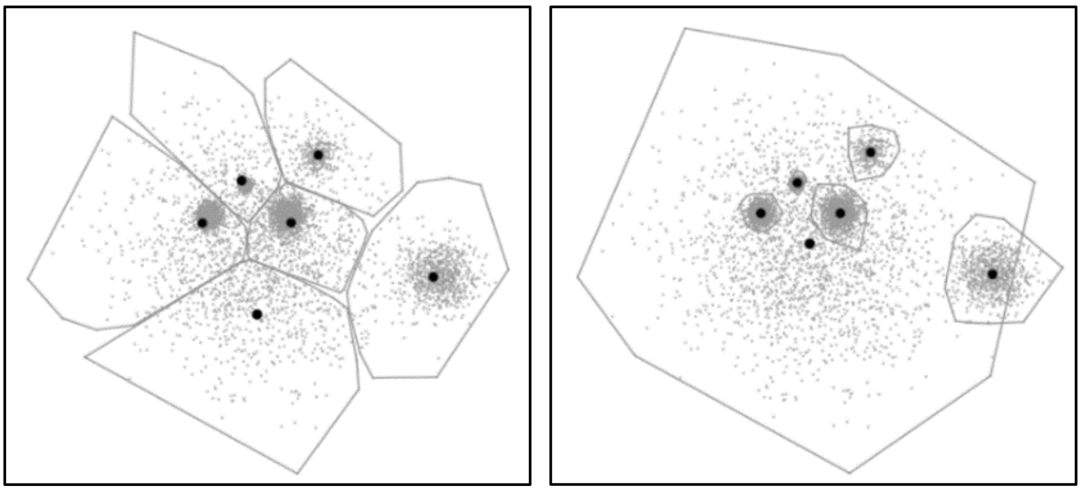


Random swap, SSE as objective function

Ground truth

Figure 26. A clustering of data set 2 using random swap on the left. The true centroid locations and partitioning are shown on the right.

When the number of nested clusters increases, the difference between the ground truth and the partitioning produced by random swap becomes even more emphasized. The partitioning shown in Figure 27 illustrates how the large cluster enveloping all the other clusters has been divided into several non-overlapping sub-clusters.



Random swap

Ground truth

Figure 27. A clustering of data set 3 using random swap on the left. The true centroid locations and partitioning are shown on the right.

Even though the random swap algorithm typically produces results that are comparable to some of the best clustering algorithms (Fränti & Kivijärvi, 2000; Fränti, 2018), these examples show that for data sets with varying cluster densities and nested clusters, a different objective function is needed in order to reach the correct clustering.

4.2 Existing approaches

Numerous density-based methods have been introduced in the literature for various applications. (Kriegel et al., 2014) Common for many of the existing density-based methods is that they calculate density by counting the number of data vectors per volume. The volume is often calculated based on a parameter such as a radius around a data vector or based on the k nearest neighbors (for example their mean distance).

In the following subchapters, some of the commonly cited algorithms are introduced. Their behavior with the density-based data sets 2 and 3 is also examined.

4.2.1 DBSCAN

The most well-known density-based clustering algorithm is the *density based spatial clustering of applications with noise (DBSCAN)* introduced by Ester et al. (1996). It attempts to formalize the previously introduced notion of a cluster having higher density of data vectors than the areas outside the cluster.

The key idea is that for any data vector to belong to a cluster, there must be at least a given number of data vectors within a specified radius. In other words, the density of the neighborhood around the data vector must exceed a given threshold. In general, the shape of the neighborhood depends on the choice of distance function.

The DBSCAN algorithm (Ester et al., 1996; Kriegel et al., 2014) takes two global inputs: the radius r and the minimum number of data vectors k. Given these two values, the *local density* of a data vector is defined as the number of data vectors k_i that lie within the radius r from the data vector x_i. If the local density is greater than the minimum number of points k, then x_i is called a *core point*.

The concepts of density-reachability and density-connectivity are the basis for the notion of cluster. Ester et al. (1996) define these as follows: A data vector p is *directly density-reachable* from another data vector q if the distance between them is shorter than r, and q is a core point. A data vector p is *density-reachable* from a vector q if they are joined by a sequence of data vectors $x_1, \dots x_n, x_1 = q, x_n = p$ such that the data vector x_{i+1} is directly density-reachable from the data vector x_i, see Figure 28. A data vector p is *density-connected* to the vector q if there is an intermediate data vector o from which both p and q are density-reachable. Data vectors that are not core points but are part of a cluster but are called *border points*. Finally, data vectors that do not belong to a cluster are called *noise points*.

Examples of density-reachability and density-connectivity are shown in Figure 28. In (a), the red border point p is density-reachable from the blue core point q. However, the reverse is not true because p itself is not a core point. In (b), the data vectors p and q are density-connected to one another through the data vector o.

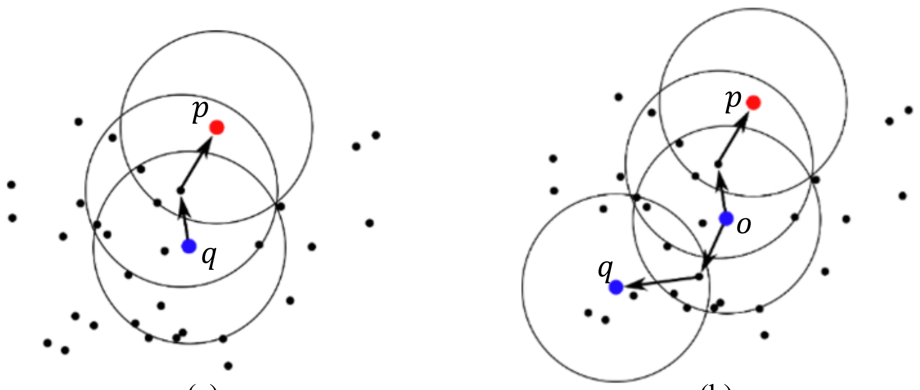


Figure 28. Density-reachability and density-connectivity, adapted from Ester et al. (1996).

Based on these definitions, Ester et al. (1996) define *density-based cluster* as a set of data vectors that satisfy the following two conditions:

(1) All pairs of data vectors belonging to the same cluster are density-connected. (Maximality)
(2) If a data vector x is density-reachable from another that is part of the cluster, then x is also part of the cluster. (Connectivity)

The procedure itself starts at an arbitrary data vector x and calculates all data vectors that are density-reachable from it given the chosen radius r and number of data vectors k. If x is a core point, then the algorithm produces a new cluster assigning to the cluster all data vectors that are density-connected to x. (Kriegel et al., 2011) If x is a border point, DBSCAN will move on to the next data vector in the data set, since no data vector is density-reachable from x.

The original DBSCAN algorithm gained widespread popularity because it was very efficient for large data sets. According to Kotsiantis & Pintelas (2004), DBSCAN outperforms hierarchical and partitioning algorithms. However, this seems to depend on the dimensionality of the data: Kriegel et al. (2011) gives the time complexity of DBSCAN as $O(N^2)$ in the worst case, although they also claim that the average runtime time complexity can be brought down to $O(N \log N)$ if the dimensionality of the data is not too high and R*-trees are used for indexing the data. For comparison, the worst case time complexity of k-means is $O(INK)$, where I is the number of iterations, N is the number of data vectors, and K is the number of clusters. Hierarchical agglomerative clustering can be performed in $O(\tau N^2)$, where τ is a small number that refers to the number of nearest neighbors that need to be updated after merging clusters. (Fränti et al., 2006)

DBSCAN does not require the number of clusters to be given as a parameter because of the way the clusters are formed based on the connectivity of data vectors to each other. The algorithm can generally be used for any data set where a distance function between two data vectors can be defined. Like with non-parametric methods in general, an important advantage is that it can discover clusters of varying sizes and shapes. DBSCAN can also handle outliers because of to its concept of noise.

A disadvantage of DBSCAN is that it struggles with data sets that contain clusters of varying densities. (Ertöz et al., 2003) An example of this is shown in Figure 29. If the chosen distance threshold is too small, all the points belonging to the sparse cluster will be considered noise. If the threshold is too large, all points will be in the same cluster. A possible solution might be to run the algorithm multiple times and removing the already found clusters from the data set after each run.

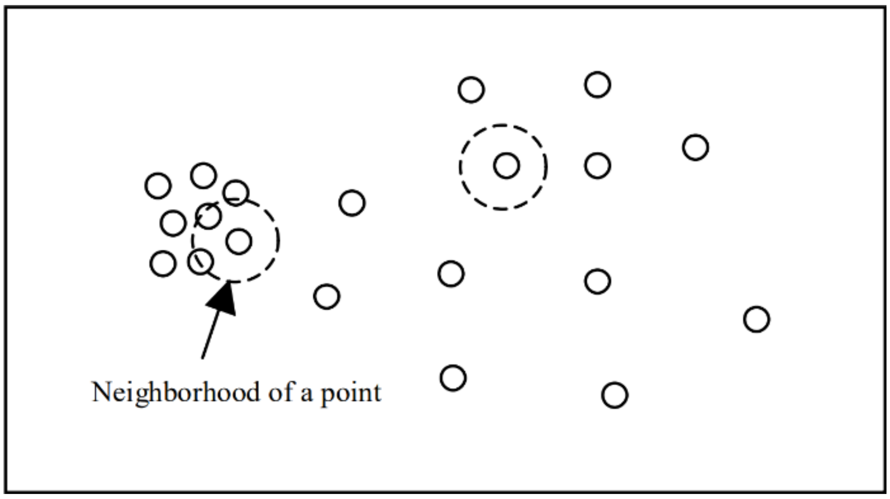


Figure 29. The distance threshold *r* defines the neighborhood of a point. (Ertöz et al., 2003)

High-dimensional data can also make clustering more challenging – distances between data vectors become more uniform, effectively making them seem more similar. (Ertöz et al., 2003) Furthermore, choosing the parameters for the distance and density thresholds requires domain expertise, and the definitions of the algorithm such as density-connectivity and border points are not very simple. A simplified version of the DBSCAN algorithm was proposed by Campello et al. (2013) which gets rid of the concept of border points.

With data sets 2 and 3 as input, DBSCAN cannot solve the correct clusters reliably with small values of k. Example solutions with $k = 5$ and different values of r for each data set are shown in Figure 30 and Figure 31. These were calculated using the DBSCAN implementation from *scikit-learn* (Pedregosa et al., 2011) and by normalizing the attributes before clustering. Since DBSCAN labels noise points separately, they are colored in red and do not belong to any cluster. DBSCAN also does not directly give the centroid locations, so they are calculated based on the partitioning after the clustering is performed.

With small values of r there are too many clusters in the solution. This happens if the distance between adjacent data vectors tends to be higher than r. As a result, you get many clusters containing only a few data vectors, and lots of points labeled as noise.

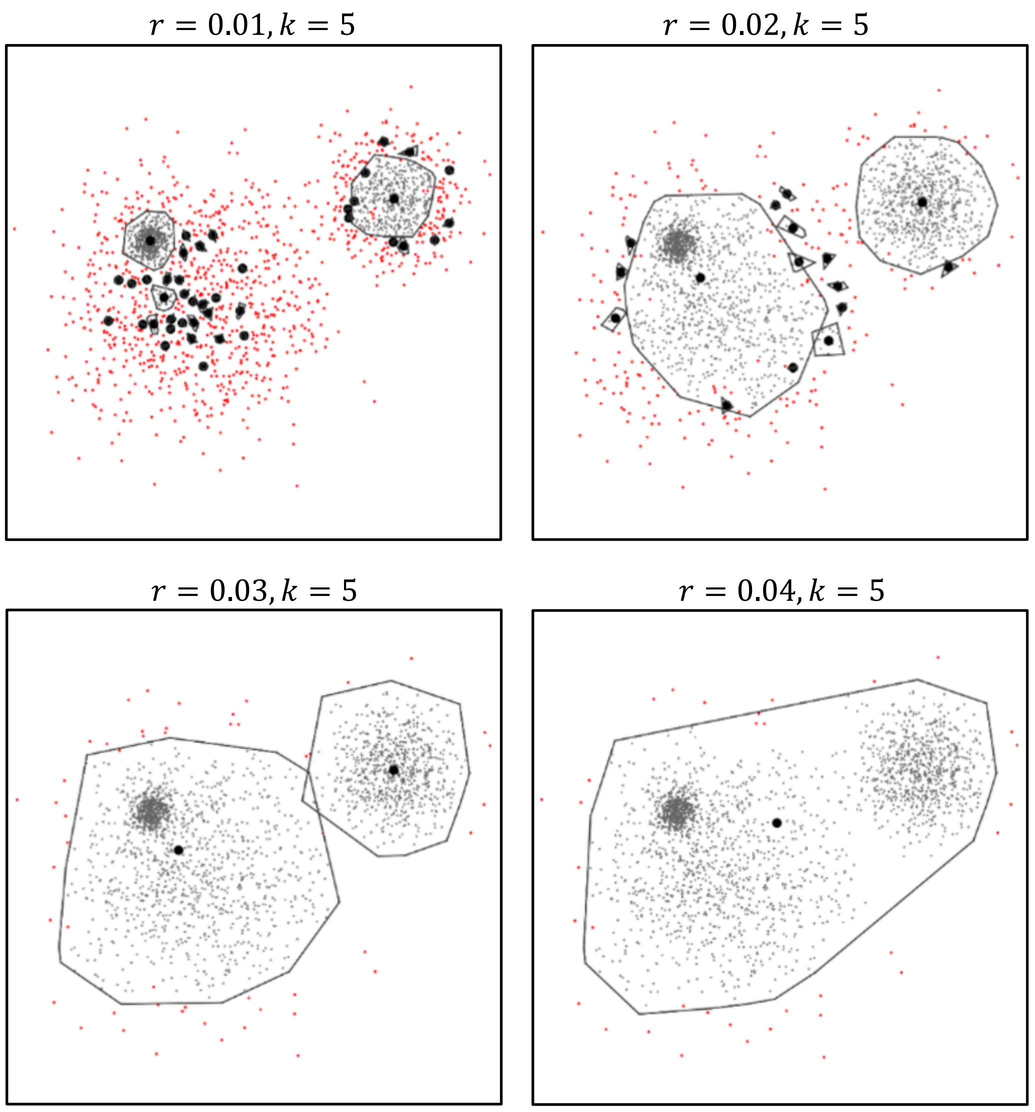


Figure 30. DBSCAN solutions for data set 2.

The best results for data set 2 are achieved when the value of r is about 0.03. However, with this value you already get too few clusters, as the largest and smallest clusters are merged into one. With $r = 0.04$ all three clusters are merged into one.

The results are similar with data set 3. With $r = 0.005$, almost all data vectors belonging to the largest cluster are labeled as noise, and there are numerous clusters containing only a few vectors. However, with these parameters the algorithm does a relatively good job at detecting the dense clusters.

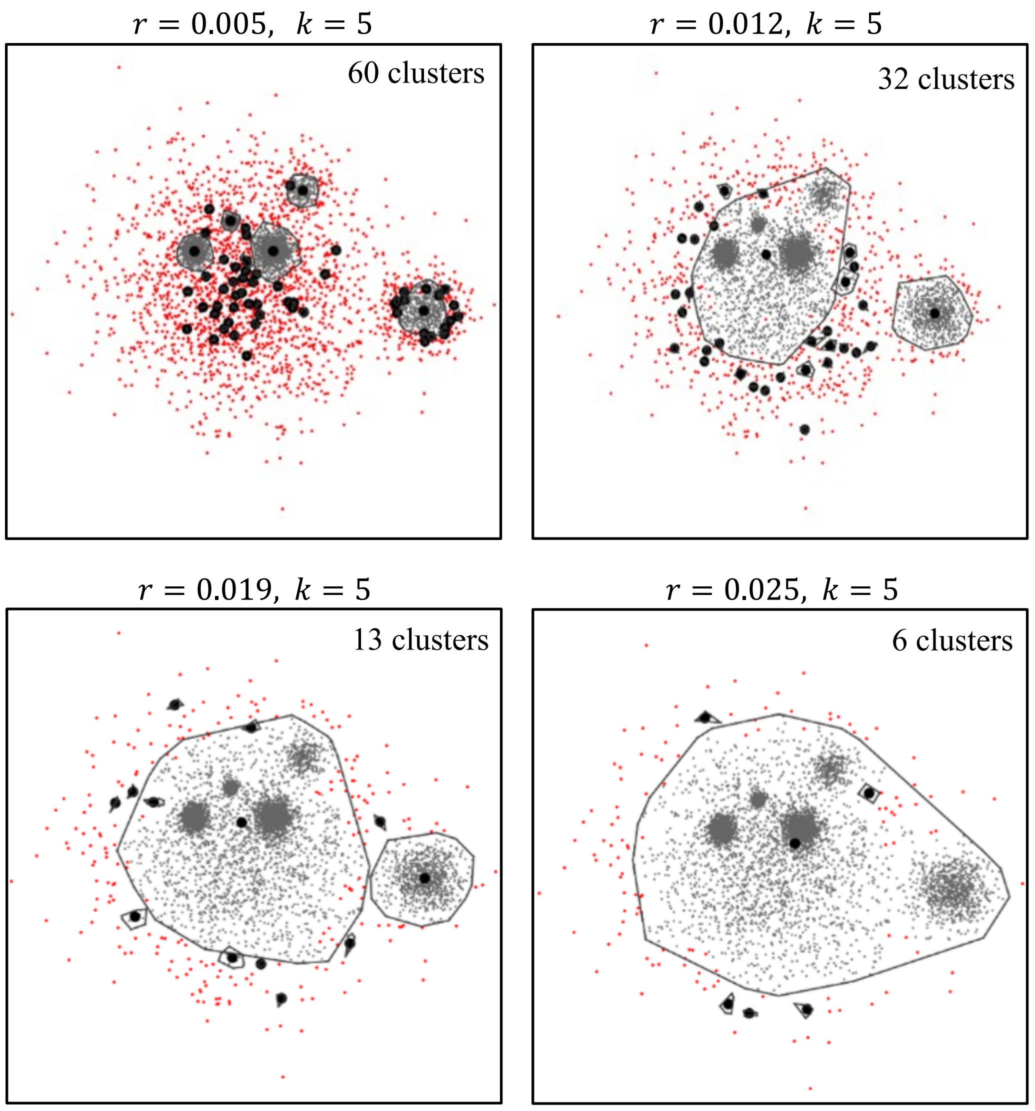


Figure 31. DBSCAN solutions for data set 3.

With higher values of r, the behavior is similar to data set 2: the problem is that many clusters are being merged into one, and too many points are being labeled as noise. With this data set, it is difficult find a value of r that would produce a result that resembles the ground truth. Even with $r = 0.025$ where all the clusters are merged into one, there are still unwanted clusters containing only a few points.

The centers of the clusters can be located more reliably by carefully choosing r and k. Figure 32 shows solutions where both parameters were set high enough in order to not create too many clusters with only few points. This results in lots points in each cluster being labeled as noise but gives very precise locations for most of the cluster centers. The drawback is that the largest clusters in each data set were not recognized but instead labeled as noise.

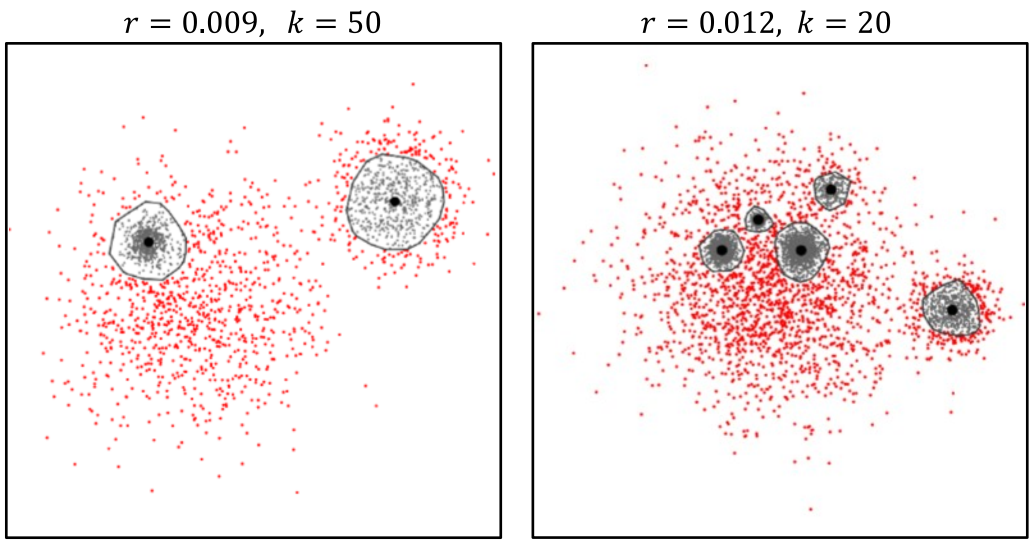


Figure 32. Solutions for data sets 2 and 3 with carefully chosen *r* and *k*.

For these datasets, DBSCAN requires a lot of tweaking of the parameters to get good results. It is possible to detect most of the cluster centers, but the quality of the partitioning must be sacrificed as many points will be labeled as noise. A potential solution to this could be to perform several runs of DBSCAN with different parameters and then combine the results.

4.2.2 Mean shift

Mean shift (Cheng, 1995) is an iterative algorithm that works by shifting each data vector to the average of its nearby data vectors. It is a generalization of the original mean shift procedure introduced by Fukunaga and Hostetler (1975). It is based on the concept of *kernel density estimation* (KDE), which is a non-parametric method for estimating the *probability density function* (PDF) of a random variable. (Comaniciu & Meer, 2002) The input data set is treated as a sample from the PDF. A more precise mathematical formulation of the algorithm can be found in Cheng (1995) and Comaniciu & Meer (2002).

On a higher level, algorithm works as follows. For each data vector, a *window* of radius r is defined, and the mean of the data vectors within the window is calculated. The window is then moved towards the mean by a *mean shift vector*. This vector points in the direction to which density increases the most. (Comaniciu & Meer, 2002)

The above steps are repeated until the mean converges to a stationary point. After each iteration, the mean will shift towards a denser region of the data set, as illustrated in

Figure 33. The end result is that each data vector will converge to a local maximum (mode) of the underlying PDF. The number of modes equals the number of clusters discovered, and each data vector that converged to the same mode belongs to the same cluster.

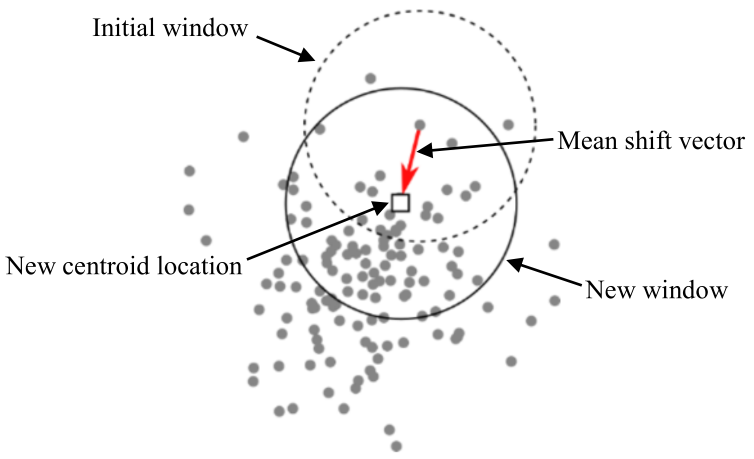


Figure 33. Illustration of the mean shift procedure.

Applications of mean shift in computer science include image segmentation, color space analysis, and face tracking (Comaniciu & Meer, 2002). It has also been applied in other fields such as microbiology. (De Brito et al., 2016)

Like many other density-based algorithms, mean shift does not require the number of clusters to be given as input parameter. This is because the procedure essentially finds the cluster center for each data vector, and the number of unique cluster centers equals to the number of clusters. However, KDE and non-parametric techniques in general may not be suitable for high-dimensional data sets because of the *curse of dimensionality*, which causes all objects to appear dissimilar to each other. (Comaniciu & Meer, 2002; Kriegel et al., 2011).

With data sets with nested clusters, mean shift is unable to find the correct centroid locations reliably. Solutions for data sets 2 and 3 with different values of window radius r are shown in Figure 34 and Figure 35. These were calculated using the mean shift implementation from *scikit-learn* (Pedregosa et al., 2011) and the attributes were normalized before clustering.

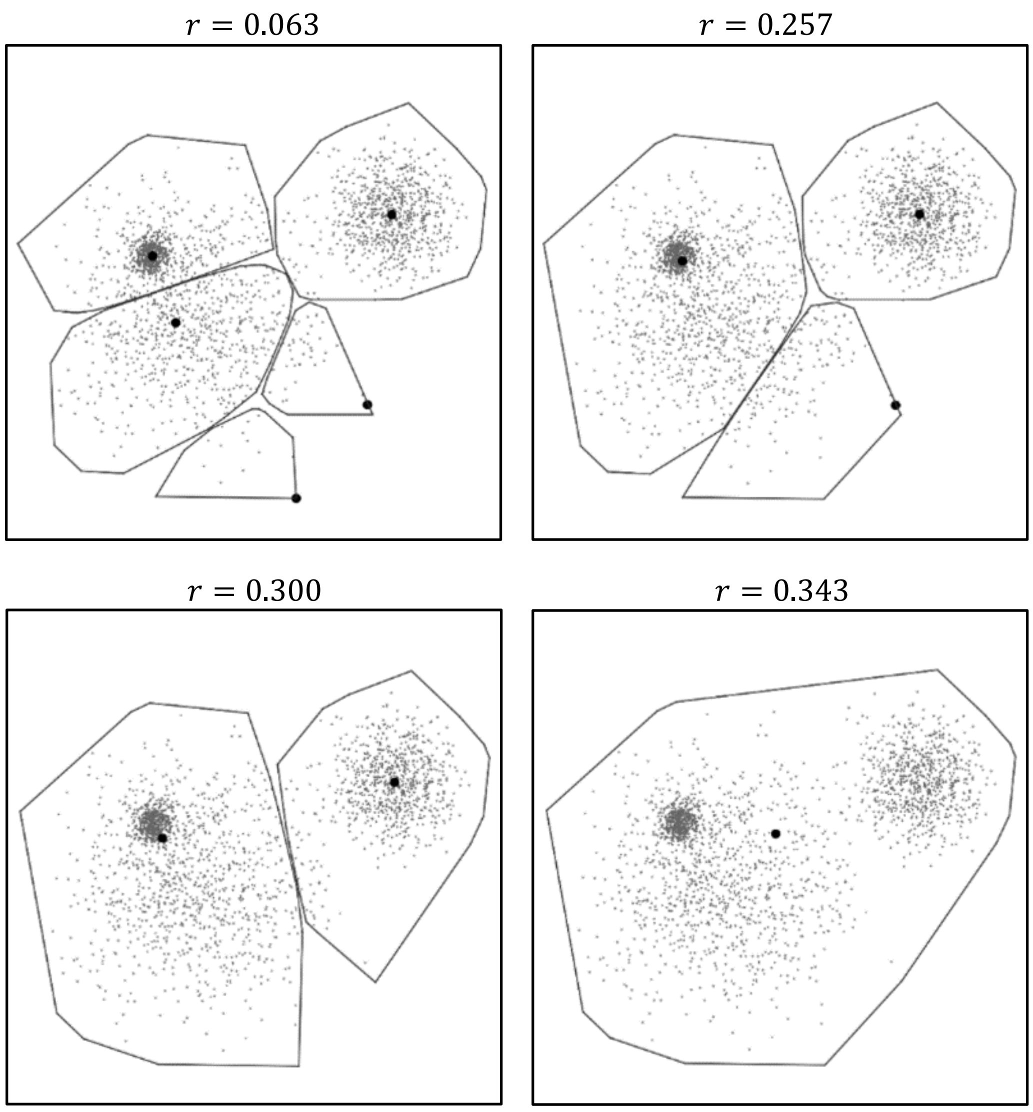


Figure 34. Mean shift solutions for data set 2.

Like partitioning methods, mean shift divides the data set into non-overlapping regions. With small values of r, mean shift tends to split nested clusters in two or more parts. With higher values it tends to merge them into larger clusters. With $r = 0.063$, the two smaller centroids are roughly in the correct locations, but the centroid of the largest one is too low, and there are two extra centroids. As the value of r is increased, the clusters tend to merge together and the centroid locations also get less precise.

With data set 3 the results are similar, as seen in Figure 35. Although many of the cluster centers are correctly placed, there are many extra ones, the partitions are mostly wrong, and the algorithm is unable to detect the largest cluster. With small values of r there are many of extra clusters. However, notable is that with $r = 0.062$, the solution may be considered correct if you take into account only the six largest partitions. As the value of r is increased, more clusters are merged, which again causes the centroid locations to get misplaced.

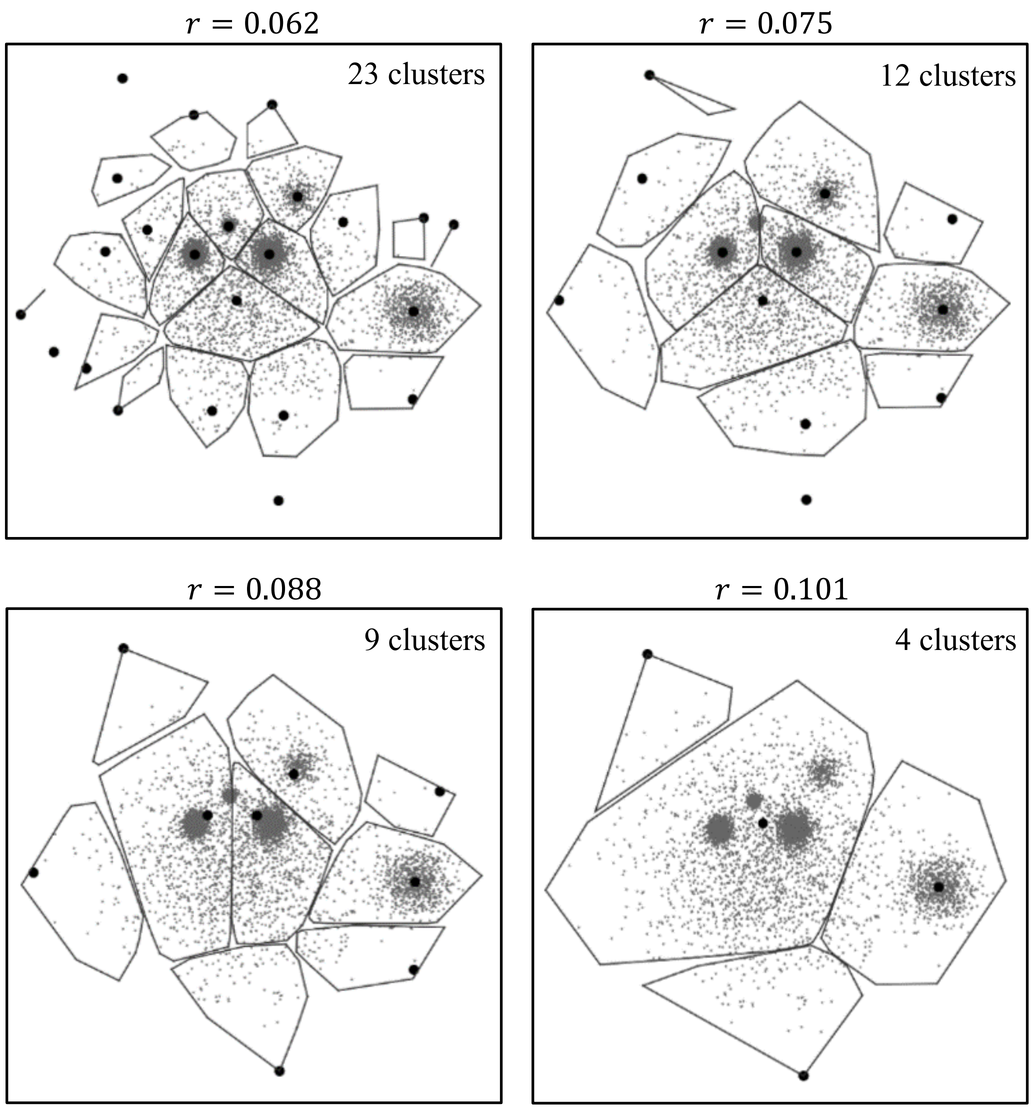


Figure 35. Mean shift solutions for data set 3.

With these data sets, mean shift produces poor results because of the nested clusters. By choosing the parameter r carefully, it is possible to find some of the cluster centers precisely, but many are missed and placed in the wrong places. The algorithm tends to produce solutions that resemble partitioning algorithms such as *k*-means, and as such undesirable for such data sets.

4.2.3 Density peaks

The *density peaks* algorithm was proposed by Rodriguez & Laio (2014). Like density-based methods in general, it is based on the assumptions that cluster centers are surrounded by areas of relatively lower density and are relatively far from data vectors with higher density.

The algorithm works as follows. First, for each data vector x_i its local density and the distance to the nearest data vector of higher density is calculated. The local density d_i for the data vector x_i is the number of data vectors that are closer to it than the radius r. The distance δ_i is the minimum of the distances between x_i and any other data vectors x_j that have higher density.

As an exception, for the data vector x_h with the highest density, $\delta_h = \max(d_i)$ is used. According to Rodriguez and Laio (2014), this is based on the observation that δ_i is significantly larger for data vectors that are local or global maxima, causing data vectors with exceptionally large δ_i to be recognized as cluster centers.

A two-dimensional example of the algorithm in action is shown in Figure 36. The point distribution shows data vectors ranked in order of decreasing density, 1 being the highest and 28 being the lowest density. There are two clusters whose density peaks are points 1 and 10, respectively. The *decision graph* is a plot of δ_i as the function of d_i. It shows how the cluster centers 1 and 10 have significantly higher densities than the rest of the points. Furthermore, the points 26, 27, and 28 also have relatively low local densities and longer distances to the nearest vector of higher density, meaning that they are isolated. These can be considered either as single-point clusters or outliers. (Rodriguez & Laio, 2014)

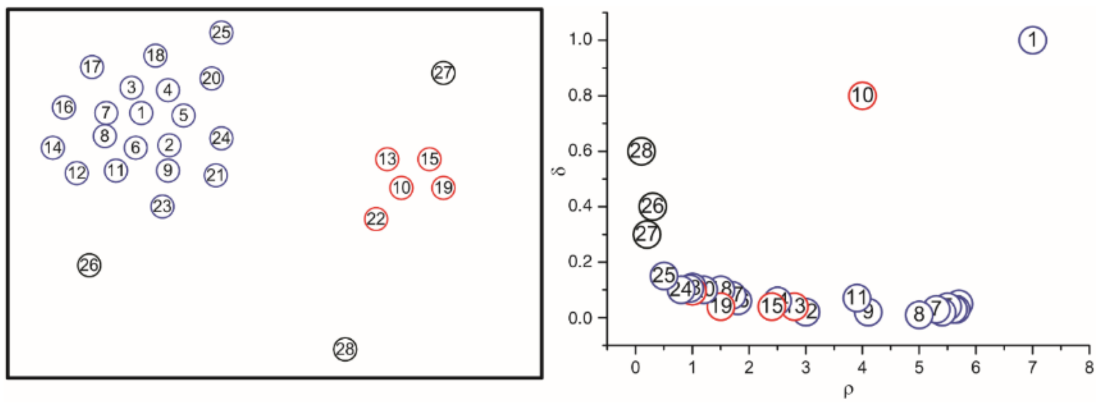


Figure 36. Point distribution (left) and decision graph (right).
Different clusters are shown in different colors. (Rodriguez & Laio, 2014)

After finding the cluster centers, the remaining data vectors are assigned to clusters. This is done for each data vector by assigning it to the same cluster as the nearest data vector of higher density. (Rodriguez & Laio, 2014) The assignment of vectors to clusters is done in a single step, contrary to iterative methods such as DBSCAN or k-means.

The advantages of density peaks are that it does not require the number of clusters to be given as parameter. Instead, the number of clusters is inferred from the peaks in local densities. It is able to find non-spherical clusters and requires only a way of measuring distance between two data vectors. (Rodriguez & Laio, 2014)

A drawback of the algorithm is that for finite samples from randomly generated data vectors, several density peaks may be detected, even though one would expect the density to be uniform. Rodriguez and Laio (2014) claim that these *density ripples* can be distinguished from the decision graph by looking at the distributions of δ_i and d_i: For genuine cluster centers, there are significant gaps in both axes compared to other points, while for density ripples, Rodriguez and Laio observed a continuous distribution following a power law that depends on the dimensionality of the data set. However, if such an analysis works, it would have to be done manually after the clustering has been performed, as the algorithm itself does not do so, and no automatic method for doing such an analysis is proposed.

Additionally, the algorithm requires that the distance between each vector pair is calculated, which becomes a bottleneck with large data sets. (Rodriguez & Laio, 2014) The time complexity according to Xu & Tian (2015) is $O(N^2)$.

With data sets with nested clusters, the algorithm fails to find all the cluster centers. Solutions for data sets 2 and 3 are shown in Figure 37. Like the density-based algorithms introduced so far, density peaks cannot detect the largest cluster either data set. In data set 2, two clusters centers are somewhat precisely detected, but the third one is completely misplaced.

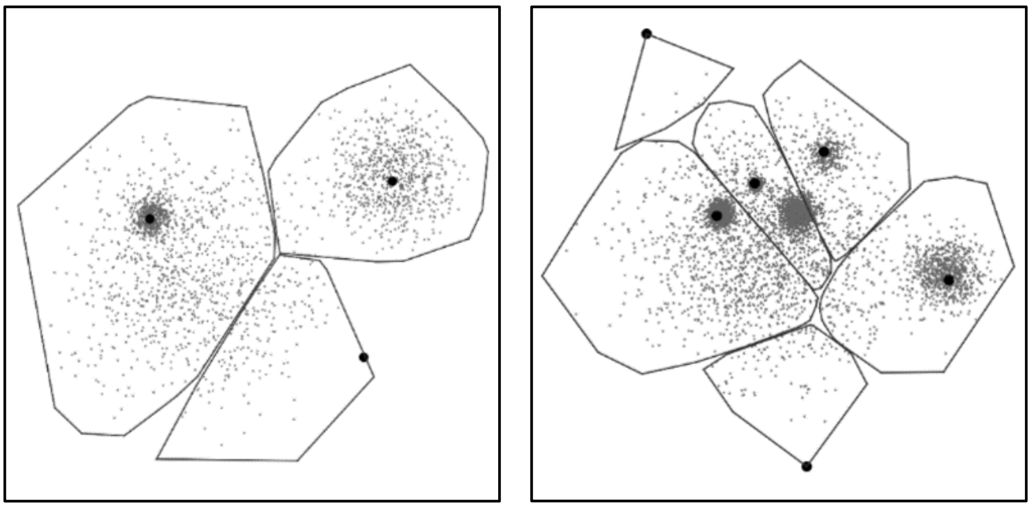


Figure 37. Density peaks solutions for data sets 2 and 3.

In data set 3, four clusters are also somewhat precisely detected, but the largest one and one of the smaller ones are not. The partitions are poor compared to the ground truth. The density peaks algorithm seems better suited for data sets where the clusters are clearly separated.

5 Clustering with weighted centroids

Centroid-based algorithms such as k-means and random swap have trouble detecting the clusters in data sets that have nested clusters or clusters of varying densities. The main problem these algorithms have is that they will always assign data vectors to the cluster whose centroid is nearest. When there are nested clusters, this is typically not desirable.

An example of this is shown in Figure 38, which shows the ground truth centroids and partitioning for a part of data set 2. Ideally, the points surrounded by the convex hull should be assigned to the cluster represented by c_2, and all points outside it should be assigned to c_1. The data vector x should therefore be assigned to the cluster represented by c_1, even though the Euclidean distance to c_2 is shorter.

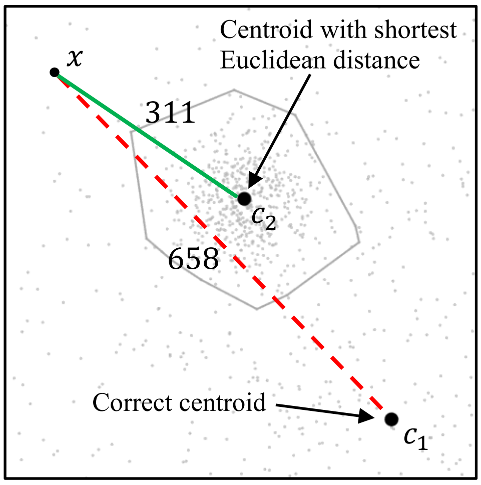


Figure 38. Cropped part of data set 2.

To achieve this kind of result, we introduce weighted centroids. Each centroid c_i will have a *weight* w_i associated with it. The weight of a centroid is related to the density of the cluster: a low weight means the cluster is sparse and a high weight means the cluster is dense with many data vectors concentrated on a small area. The notation $W = \{w_1, w_2, \dots, w_K\}$ is used to denote the collection of centroid weights.

The centroid weights are illustrated in Figure 39 for the ground truth centroids of data set 2. The large, sparse cluster has the lowest weight while the cluster it envelops has a significantly higher weight because of its high density.

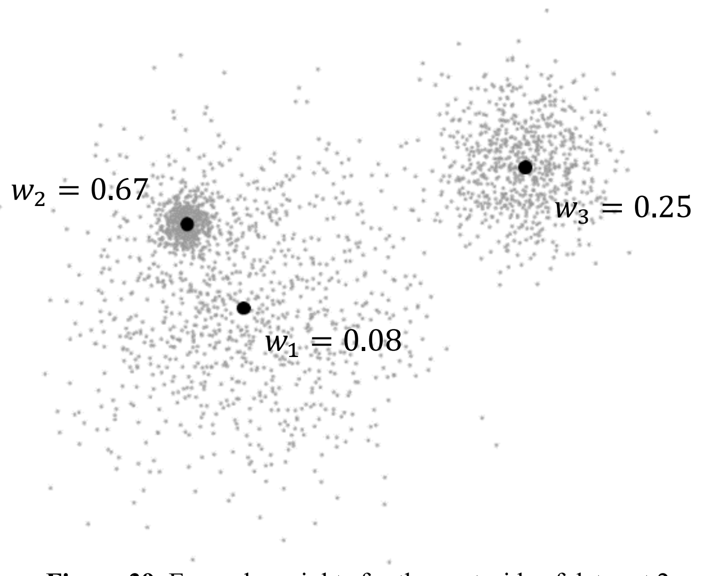


Figure 39. Example weights for the centroids of data set 2.

A similar idea is described by Koyuncu & Jafarkhani (2017). They approach the problem from the point of view that we want to allocate a given number of weighted points optimally to an area, for example by finding the optimal locations for cellular network towers in an area of subscribers. However, the authors do not provide an algorithm or a method for calculating the weights.

5.1 Problem definition

Like in the traditional clustering problem defined in chapter 2.2, we assume that the number of clusters is given, the data sets consist of Euclidean points, and we are interested in the locations of the clusters. Thus, we formulate the clustering problem with weighted centroids as follows:

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

Since the only addition to the clustering problem is the centroid weights, any centroid-based clustering algorithm can be used to solve the problem, if they are modified take centroid weights into account.

5.2 Distance measure

For measuring the distance between a data vectors x and a centroid c_i, the *weighted squared Euclidean distance* is used:

$$\mathrm{d}(x, c_i) = \sqrt{\sum_{i=1}^{D} \left(x^j - c_i^j\right)^2} = w_i \|x - c_i\| \tag{5}$$

During partitioning, this allows data vectors to be assigned to clusters whose centroid is not necessarily the closest by Euclidean distance alone, but the one with the shortest weighted distance. Effectively, this allows nested clusters to exist.

Figure 40 shows both unweighted and weighted distances calculated from the data vector x to centroids c_1 and c_2. On the left, unweighted Euclidean distances are used, resulting in x being assigned to the cluster represented by c_2 since it is the nearest. On the right, weighted Euclidean distances are used. Since the weight of c_2 is very small (0.08), the weighted distance between x and c_2 ends up being smaller than the weighted distance between x and c_1, despite the latter unweighted distance being over twice as large.

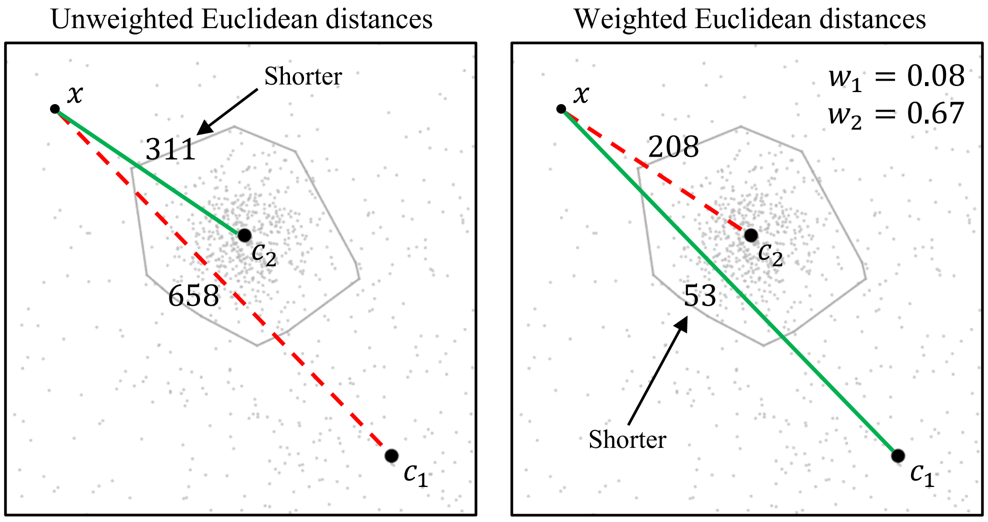


Figure 40. Comparison between unweighted and weighted distances.

5.3 Objective function

With this density-based approach, the objective function to minimize is still the sum of squared errors (SSE) shown in Equation 2 in chapter 2.4. The only thing to note is that squared distance is multiplied by the centroid weight:

$$\text{Weighted SSE} = \sum_{i=1}^{N} \left(w_i \left\| x_i - c_{p_i} \right\|^2 \right) \tag{6}$$

5.4 Estimating cluster density

The density-based clustering method defined in chapter 5.1 requires the centroid weights to be known in advance. In this subchapter, we will do it empirically during the clustering process.

Assuming we know the ground truth partition of the data set, the density of the cluster can be estimated with the *mean distance* between the centroid c_i and the data vectors belonging to this cluster:

$$\text{md}(c_i) = \frac{1}{n_i} \sum_{j=1}^{n_i} \left\| x_j - c_i \right\| \tag{7}$$

The mean distance indicates how far the data vectors are from their centroid, on average. A small mean distance means that the cluster is dense while a long mean distance means that the cluster is sparse.

The mean distances for the three clusters in data set 2 are visualized in Figure 41. The radius of each circle equals to the mean distance for each cluster.

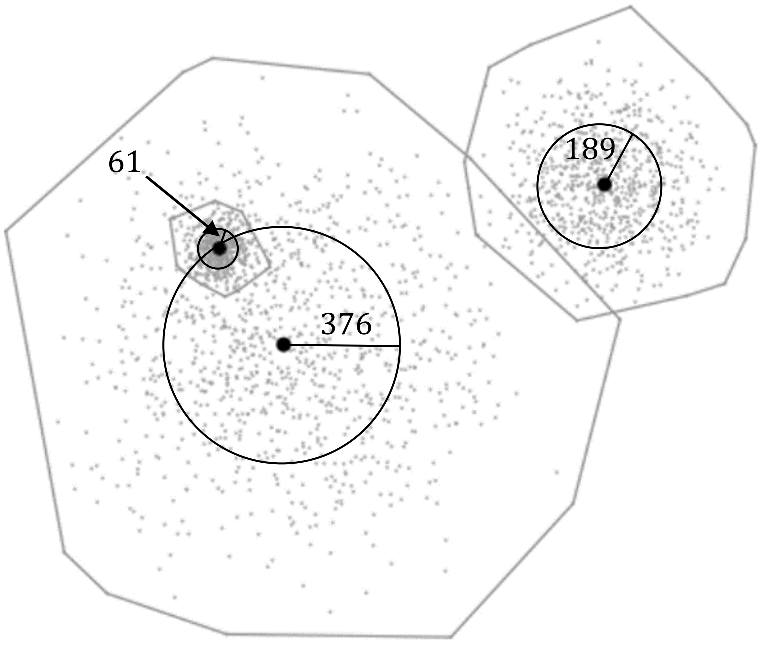


Figure 41. Mean distances from the cluster centroids for data set 2.

Since the mean distance is inversely proportional to the density of the cluster, we define the *density* $\mathrm{dens}(i)$ of the cluster represented by the centroid c_i to be the inverse mean distance:

$$\mathrm{dens}(i) = \frac{1}{\mathrm{md}(c_i)} \tag{8}$$

By this definition, the higher the density, the closer the data vectors are to its centroid, on average. Conversely, a lower density means that the data vectors are on average far away from the cluster centroid.

5.5 Calculating centroid weights

There are many ways in which centroid weights could be defined. In this thesis, the centroid weights are calculated based on the densities of the clusters. We define the *weight* w_i of the centroid c_i as the density of the corresponding cluster divided by the sum of all cluster densities:

$$w_i = \frac{\mathrm{dens}(i)}{\sum_{j=1}^{K} \mathrm{dens}(j)} \tag{9}$$

Dividing the density of the cluster by the sum normalizes the weights to the range $[0, 1]$. This is not strictly required, but makes the weights more human-readable. It also means that all the weights sum up to one:

$$\sum_{i=1}^{K} w_i = \sum_{i=1}^{K} \left(\frac{\text{dens}(i)}{\sum_{j=1}^{K} \text{dens}(j)} \right) = \frac{\sum_{i=1}^{K} \text{dens}(i)}{\sum_{j=1}^{K} \text{dens}(j)} = 1 \tag{10}$$

The process of estimating the cluster densities and the centroid densities is shown in Figure 42. On the left, the density of each cluster is calculated based on the inverse mean distance. On the right, each density is divided by the sum of all densities to get the weight for each centroid.

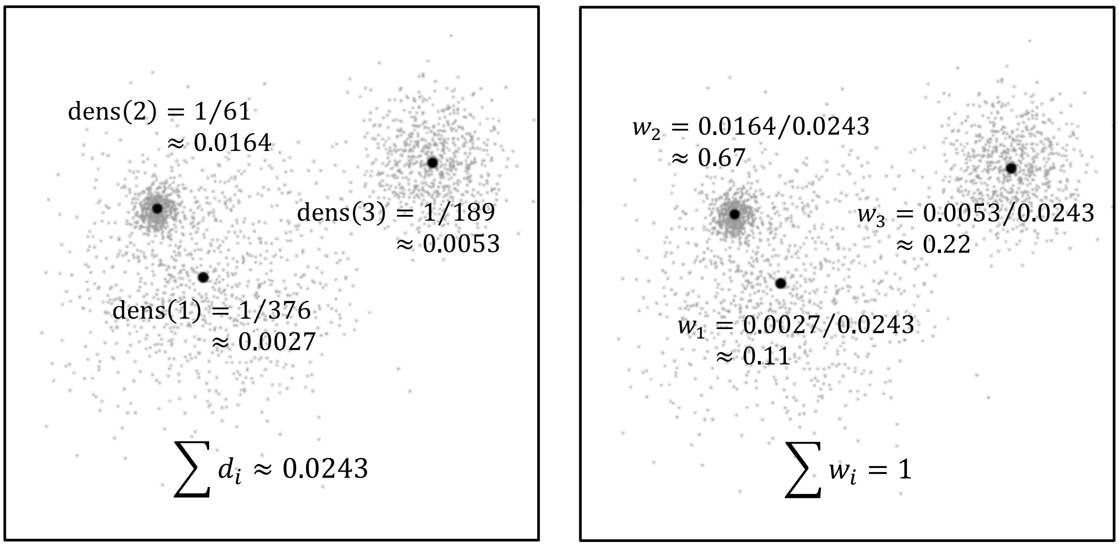


Figure 42. Estimating cluster densities and centroid weights for data set 2.

5.6 Modified algorithms

Following are the *k*-means and random swap algorithms modified to use centroid weights. These are used in the experiments in chapter 6.

5.6.1 *k*-means

The modified *k*-means algorithm (Figure 43) takes a data set $X = \{x_1, x_2, \dots, x_N\}$, centroids weights $W = \{w_1, w_2, \dots, w_K\}$, and the number of clusters K as parameters, and it returns the centroid locations $C = \{c_1, c_2, \dots, c_K\}$ and the partitioning $P = \{p_1, p_2, \dots, p_N\}$ of the data points as the results.

```
**K-means**$(X, W, P, C)$: $\rightarrow (C, P)$
REPEAT
   $C_{\text{prev}} \leftarrow C$
   FOR $i := 1$ TO $N$ DO
      $p_i \leftarrow$ FindNearestCentroid$(x_i, C, W)$
   FOR $j := 1$ TO $K$ DO
      $c_j \leftarrow$ CalculateCentroid$(X, P, j)$
UNTIL $C = C_{\text{prev}}$
```

Figure 43. Pseudo code for the *k*-means algorithm with weighted centroids.

Compared to the original *k*-means algorithm (Figure 15), the centroid weights are given as an additional parameter. Additionally, the weights are utilized when calculating the distances to centroids in the FindNearestCentroid function (Figure 44). The CalculateCentroid function is identical to the original.

```
**FindNearestCentroid**$(x, C, W)$: $\rightarrow i_{\text{nearest}}$
$d_{\text{nearest}} \leftarrow \infty$
FOR $i := 1$ TO $K$ DO
   $d \leftarrow w_i \|x - c_i\|$
   IF $d < d_{\text{nearest}}$ THEN
      $d_{\text{nearest}} \leftarrow d$
      $i_{\text{nearest}} \leftarrow i$
RETURN $i_{\text{nearest}}$
```

Figure 44. Pseudo code for finding the nearest centroid.

5.6.2 Random swap

The modified random swap algorithm (Figure 45) takes a data set $X = \{x_1, x_2, \dots, x_N\}$, the centroids weights $W = \{w_1, w_2, \dots, w_K\}$ and the number of clusters K as parameters, and it returns the centroid locations $C = \{c_1, c_2, \dots, c_K\}$ and the partitioning $P = \{p_1, p_2, \dots, p_N\}$ of the data points as the results.

```
**RandomSwap**$(X, W, P, C)$: $\rightarrow (C, P)$
REPEAT $T$ TIMES
   $(C_{\text{new}}, j) \leftarrow$ SwapCentroid$(X, C)$
   $P_{\text{new}} \leftarrow$ LocalRepartition$(X, W, C_{\text{new}}, P, j)$
   $(C_{\text{new}}, P_{\text{new}}) \leftarrow$ K-means$(X, W, C_{\text{new}}, P_{\text{new}})$
   IF $f(C_{\text{new}}, P_{\text{new}}, W) < f(C, P)$ THEN
      $(C, P) \leftarrow (C_{\text{new}}, P_{\text{new}})$
RETURN $(C, P)$
```

Figure 45. Pseudo code for the random swap algorithm with weights.

The local repartition step (Figure 46) utilizes centroid weights when finding the nearest centroid for each data vector. The random swap step SwapCentroid is identical to the original.

```
**LocalRepartition**$(X, W, C_{\text{new}}, P, j): \to P$
FOR $i := 1$ TO $N$ DO
  IF $p_i = j$ THEN
    $p_i \leftarrow$ FindNearestCentroid$(x_i, C, W)$
FOR $i := 1$ TO $N$ DO
  IF $\|x_i - c_j\| < \|x_i - c_{p_i}\|$ THEN
    $p_i \leftarrow j$
RETURN $P$
```

Figure 46. Re-allocation of data vectors to centroids.

5.7 Problems with calculating weights dynamically

The obvious drawback of the method described in this chapter is that the weights for each cluster centroid have to be known in advance. In practice, this means that a pre-processing step that estimates the centroid weights is needed, and the estimated weights are then given to the clustering algorithm as input. An alternative approach is to try to calculate the weights dynamically during the clustering process.

The idea is to give equal weights for each centroid as the initial weights. Then, an additional *weight calculation step* is performed after the new centroid locations and partitions have been calculated. Figure 47 shows the random swap algorithm modified with the weight calculation step.

```
**WeightAdjustingRandomSwap**$(X, W, P, C): \to (C, P)$
REPEAT $T$ TIMES
  $(C_{\text{new}}, j) \leftarrow$ SwapCentroid$(X, C)$
  $P_{\text{new}} \leftarrow$ LocalRepartition$(X, W, C_{\text{new}}, P, j)$
  $(C_{\text{new}}, P_{\text{new}}) \leftarrow$ K-means$(X, W, C_{\text{new}}, P_{\text{new}})$
  IF $f(C_{\text{new}}, P_{\text{new}}, W) < f(C, P)$ THEN
    $(C, P) \leftarrow (C_{\text{new}}, P_{\text{new}})$
  $W \leftarrow$ CalculateWeights$(X, P, C)$
RETURN $(C, P)$
```

Figure 47. Pseudo code for the random swap algorithm with the weight calculation step.

In this step, the weight for each centroid is calculated based on the corresponding partition. During writing this thesis, the function CalculateWeights was implemented by using a variation of the mean distance based approach introduced earlier in this chapter, where density is defined to be the number of data vectors in a cluster divided by the mean distance of the data vectors from the cluster centroid. Pseudo code for this approach is seen in Figure 48.

```
**CalculateWeights**$(X, P, C): \rightarrow W$
FOR $i := 1$ TO $K$ DO
    $\mathrm{dens}(i) \leftarrow n_i / \mathrm{MeanDistance}(X, p_i, c_i)$
FOR $i := 1$ TO $K$ DO
    $w_i \leftarrow \mathrm{dens}(i) / \sum \mathrm{dens}(j)$
RETURN $W$
```

Figure 48. Pseudo code optimizing the centroid weights.

Figure 49 shows the how the random swap algorithm typically behaves with data set 2 with the additional weight calculation step. On the first iteration, each weight is initialized as 0.33, so the partitioning is the same as that produced by random swap. The weights are then calculated based on each partitioning, giving the weights 0.11, 0.49 and 0.40 for the second iteration.

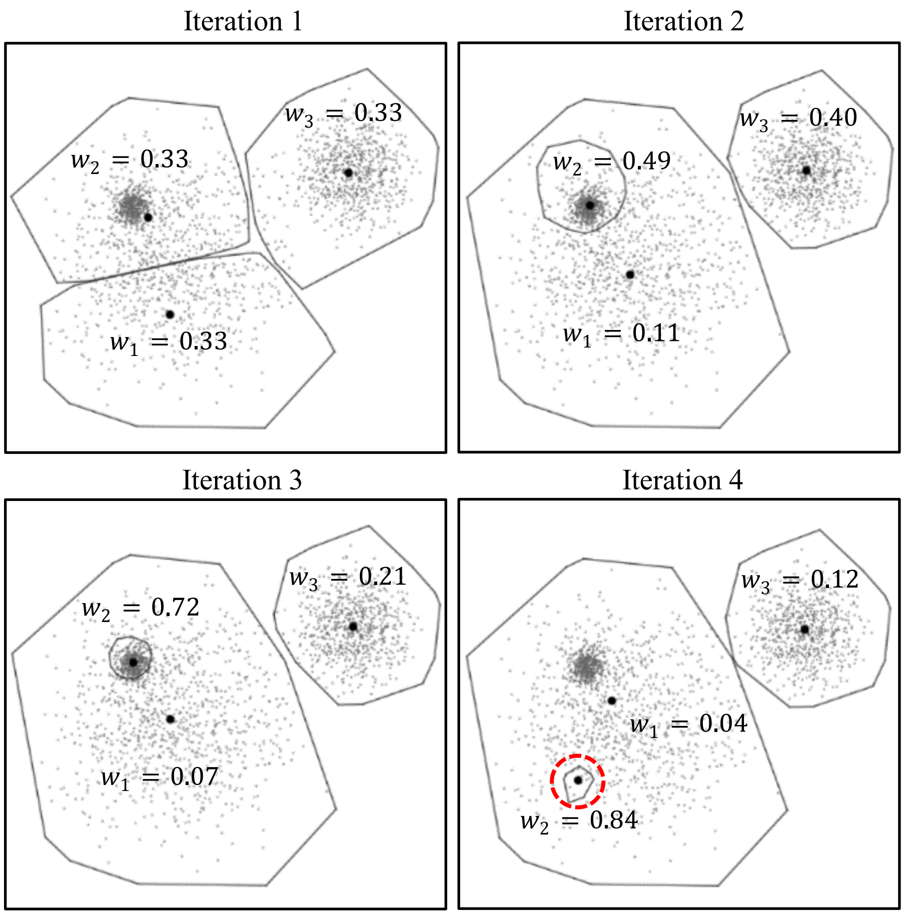


Figure 49. Behavior of random swap with the weight calculation step.

In this example run, notable is that on the fourth iteration, the highlighted centroid was swapped far from its correct location. This seems to happen because the centroid of the large, sparse cluster has such a low weight (0.04) that although the centroid with heavier weight (0.84) is at an incorrect position, the overall weighted SSE is lower than during the previous iteration. Thus, the trial swap is accepted.

In some cases, the algorithm produces the correct result, as seen in Figure 50. However, more common is the situation seen in Figure 51.

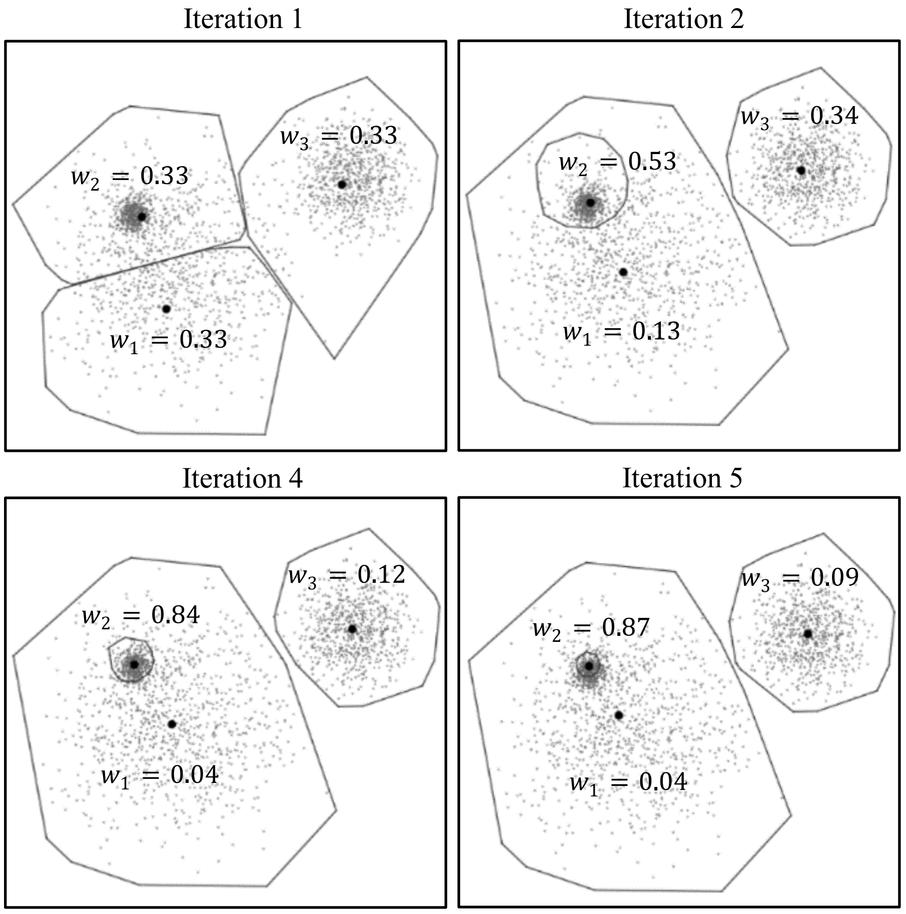


Figure 50. Random swap with weight calculation producing the correct solution.

The behavior of the algorithm is similar with data set 3. Figure 51 shows how it can reach an almost correct solution. One centroid is misplaced, causing the highlighted area to contain two centroids.

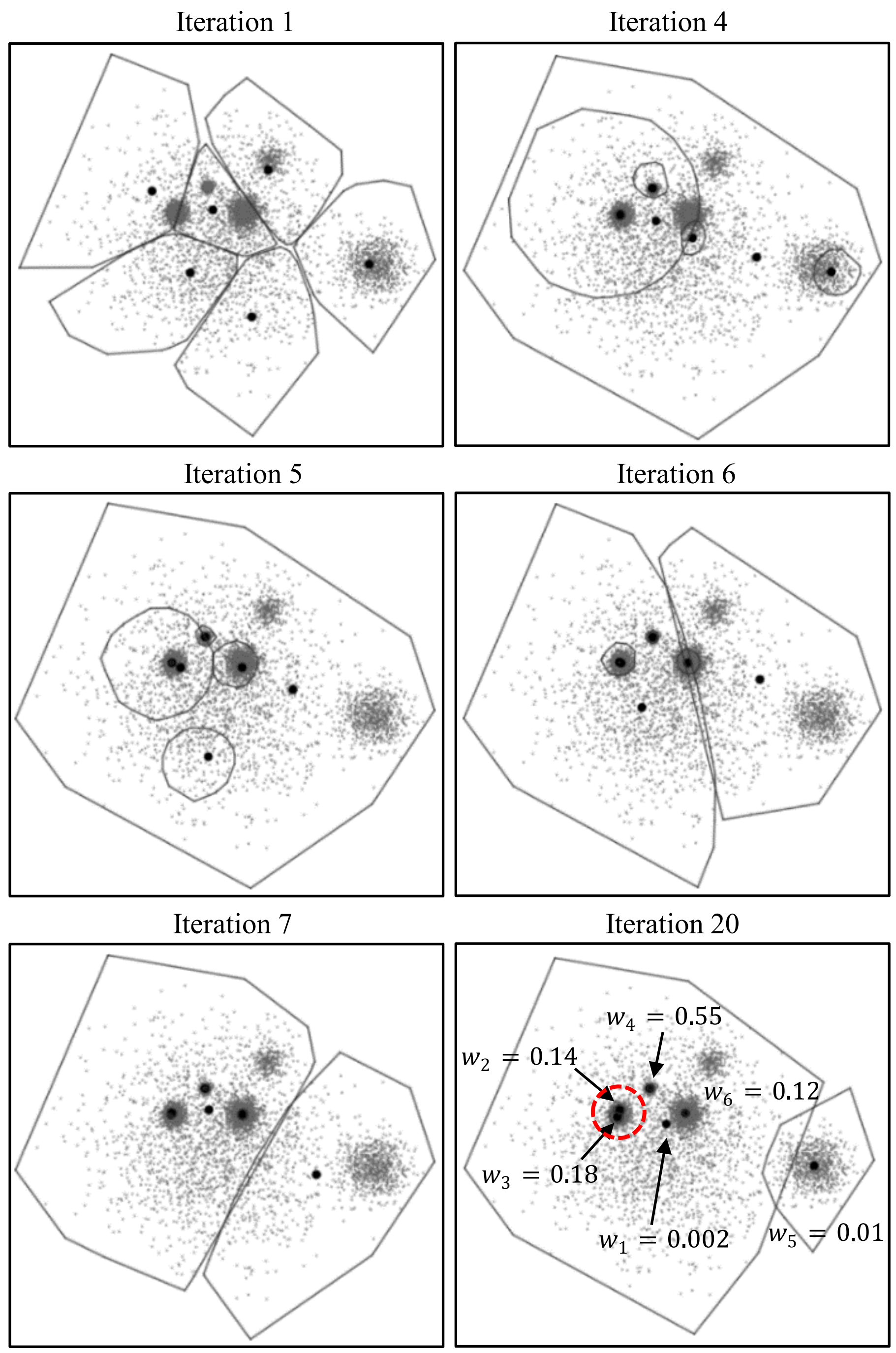


Figure 51. Behavior of random swap with the weight calculation step and data set 3.

After 20 iterations, there was no improvement even after a total of 10,000 random swap iterations. Similar to data set 2, the problem seems to be that once the weights of certain centroids become low enough, it becomes more and more difficult to find locations for them that would result in a lower weighted SSE. For example, even if one of centroids from the centroid-rich area (weights w_2 and w_3) were moved to the cluster in the top-right corner with no centroids, the overall weighted SSE would not improve.

6 Experiments

Two sets of experiments were performed using the data sets and algorithms introduced in chapter 3. The purpose of these experiments is to examine whether the method based on weighted centroids would be suitable for clustering data sets with clusters of varying densities, especially when the data set contains nested clusters.

In the first set of experiments, the *k*-means and random swap algorithms were run with data sets 2 and 3 as input. The sum of squared errors (SSE) was used as the objective function. For the second set of experiments, both algorithms modified to use weighted SSE as the objective function, as described in chapter 5.6.

Since the ground truth of each data set was known, the clustering solution of each run was compared to the ground truth by inspecting the solution visually and by evaluating the clustering quality using two different measures: an external validity indices called *centroid index* (CI) for evaluating cluster-level differences, and *centroid similarity index* (CSI) for evaluating the point-level differences. These will be introduced in the following subchapter.

All programs were written in C and were run on a desktop computer with an Intel Core i5-6600K 3.5 GHz processor and 16 gigabytes of RAM.

6.1 Evaluating clustering quality

To compare the clustering results of different algorithms, we need a way to measure the quality of the clustering. In these experiments, the *centroid index* (CI) and *centroid similarity index* (CSI) were used for this purpose.

6.1.1 Centroid index

Centroid index is a cluster-level similarity measure proposed by Fränti et al. (2014) that estimates the similarity of two clustering solutions based on their centroids. The centroid index for two solutions $C = \{c_1, c_2, \dots, c_{K_1}\}$ and $C' = \{c'_1, c'_2, \dots, c'_{K_2}\}$ is calculated as follows. First, the *nearest neighbor mappings* $C \rightarrow C'$ are constructed:

$$q_i \leftarrow \underset{1 \le j \le K_2}{\arg\min} \left\| c_i, c_j' \right\|^2 \quad \forall\, i \in [1, K_1] \tag{11}$$

A target centroid c_j' is an *orphan* if there are no centroids c_i that consider it the nearest:

$$\mathrm{orphan}(c_j') = \begin{cases} 1, & q_i \neq j \;\forall\, i \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

The dissimilarity between C and C' is then defined as the number of orphans in the target clustering solution:

$$\mathrm{CI}_1(C, C') = \sum_{j=1}^{K_2} \mathrm{orphan}(c_j') \tag{13}$$

This mapping is not symmetric, meaning that in general $\mathrm{CI}_1(C, C') \neq \mathrm{CI}_1(C', C)$. A symmetric variant is defined by Fränti et al. (2014) as follows:

$$\mathrm{CI}(C, C') = \max\{\mathrm{CI}_1(C, C'), \mathrm{CI}_1(C', C)\} \tag{14}$$

The value $\mathrm{CI} = 0$ means that each centroid c_i is mapped to exactly one c_j' and vice versa. In other words, the two solutions have the same global structure. A value of $\mathrm{CI} > 0$ gives the number of clusters that have been allocated differently between the solutions.

Figure 52 shows an illustration of how the centroid index is calculated. The ground truth centroids are shown in blue and the clustering solution is shown in red. Each solution centroid is mapped to the nearest ground truth centroid. The numbers beside the ground truth centroids indicate the number of solution centroids that are mapped to each.

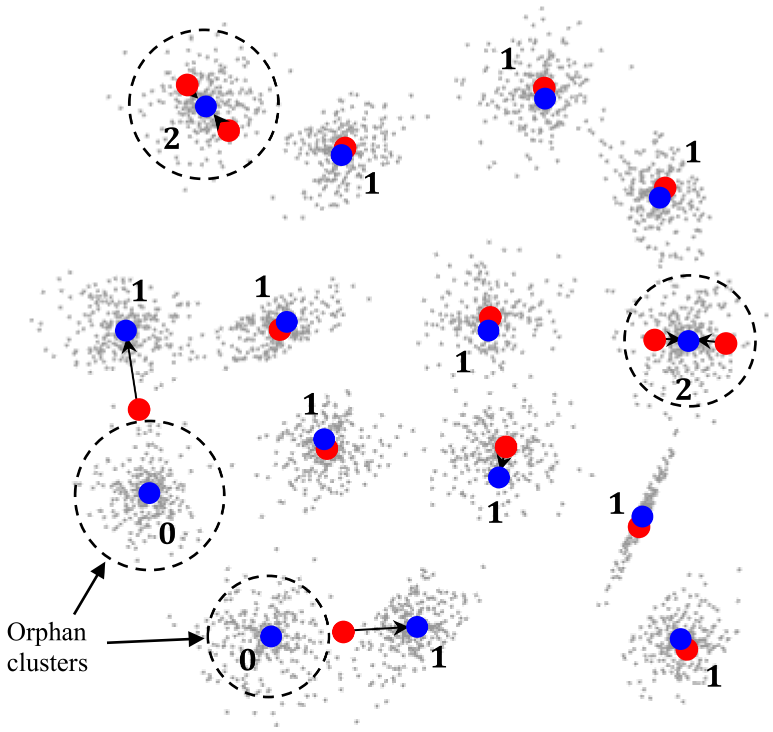


Figure 52. A clustering solution with CI = 2. The blue dots represent the ground truth centroids and the red dots solution centroids.

If a ground truth centroid has no solution centroids mapping to it, it is considered an orphan. The differences in the global clustering structure are highlighted with dotted lines. In the above case, there are two orphan clusters, resulting in CI $= 2$.

6.1.2 Centroid similarity index

Centroid similary index (Fränti et al., 2014) is a generalization of centroid index that is suitable for measuring point-level differences between two solutions. Like with centroid index, first a nearest neighbor mapping is done both ways, from solutions $C \rightarrow C'$ and $C' \rightarrow C$. The distances between each pair of centroids are calculated as Euclidean distances and each centroid in the source set is mapped to the nearest in the target set. For each mapped pair, the number of shared points is then calculated, and the CSI value is then obtained as follows:

$$\mathrm{CSI} = \frac{S_{12} + S_{21}}{2}, \text{where } S_{12} = \frac{\sum_i^{K_1} C_i \cap C_j}{N}, S_{21} = \frac{\sum_j^{K_2} C_j \cap C_i}{N} \tag{15}$$

While CI is used to evaluate the similarity of the global structure of two solutions on cluster-level, CSI is more fine-grained as it measures the similarity at point-level.

$\mathrm{CSI} = 0$ means that the two solutions are completely dissimilar, while $\mathrm{CSI} = 1$ means that the solutions are identical.

6.2 Experimental setup

Experiments were run with data sets 2 and 3, introduced in chapter 3.1. Each experiment consisted of 100 runs. For the random swap algorithm, 1,000 iterations were used. Each run used random data vectors as the initial centroids. For each set of repeats, the following data was recorded:

- Success rate, i.e. the percentage of runs that reached $\mathrm{CI} = 0$
- Average CI value for all runs
- Average CSI value for all runs
- Average number of iterations to reach $\mathrm{CI} = 0$
- Average time to reach $\mathrm{CI} = 0$
- Average SSE or weighted SSE

The centroid weights for the experiments were chosen by using the mean distance based approach described in chapter 5.5. First, the mean distance $\mathrm{md}(c_i)$ between each cluster centroid c_i and the data vectors belonging to that cluster was calculated based on the ground truth of each data set. Then, the density estimate d_i was calculated as the inverse mean distance. Finally, the centroid weights w_i were calculated based on the density.

The calculated values for data set 2 are shown in Table 2, and the values for data set 3 are shown in Table 3. The resulting centroid weights for each data set are shown in Figure 53.

Table 2. Mean distances and weights for data set 2.

Cluster i	**Mean distance** $\mathrm{md}(c_i)$	**Density** $\mathrm{dens}(i)$	**Weight** w_i
1	376	2.66×10^{-2}	0.11
2	61	16.44×10^{-2}	0.67
3	189	5.29×10^{-2}	0.22

Table 3. Mean distances and weights for data set 3.

Cluster i	**Mean distance** $\mathrm{md}(c_i)$	**Density** $\mathrm{dens}(i)$	**Weight** w_i
1	1259	0.79×10^{-2}	0.02
2	64	15.69×10^{-2}	0.42
3	189	5.30×10^{-2}	0.14
4	127	7.88×10^{-2}	0.21
5	188	5.31×10^{-2}	0.14
6	367	2.72×10^{-2}	0.07

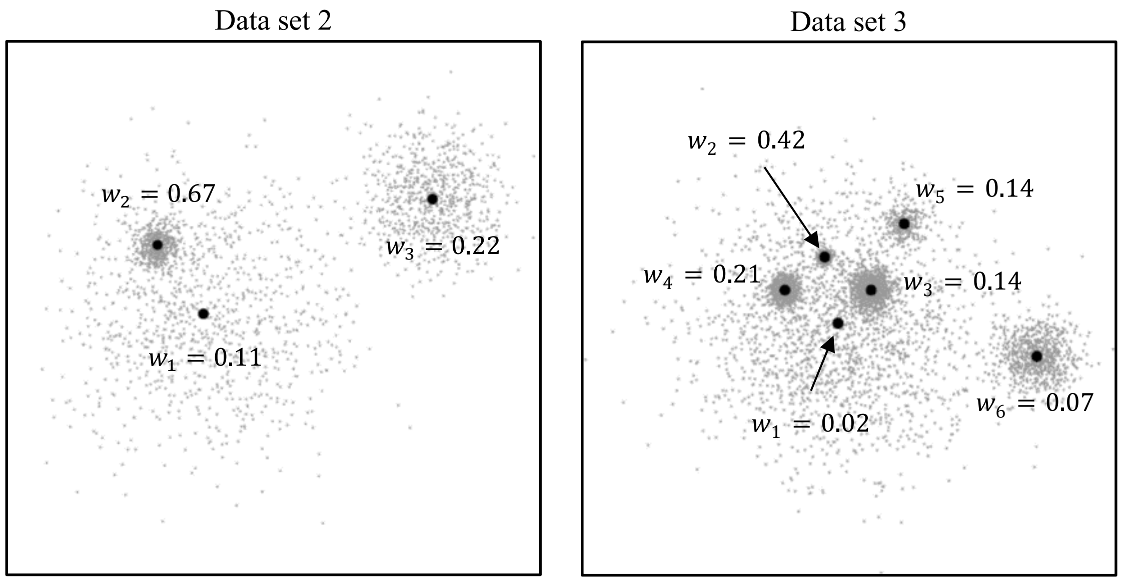


Figure 53. Calculated centroid weights for data sets 2 and 3 based on their ground truth partitions.

6.3 SSE results

For data set 2, the unweighted version of both k-means and random swap reach $\mathrm{CI} = 0$ with a success rate of 100%, see Table 4. In fact, they both arrive at exactly the same solution, as revealed by the SSE. The average CSI is 0.80, which is a mediocre result compared to the ground truth partitioning.

Although the 100% success rate seems like a perfect result, we can see that the partitions are far from the ground truth if we inspect the solution visually. The comparison in Figure 54 illustrates how the centroids are roughly in the correct locations, resulting in $\mathrm{CI} = 0$. The CSI value of 0.80 tells that there is still room for improvement. Specifically, the problem is that both algorithms partition the data set into non-overlapping regions. Such partitions are undesirable in this case, because clusters within other clusters will not be detected. Crucially, the largest cluster has been roughly divided in half instead of enveloping the smallest cluster. Therefore, despite the high success percentage, the solutions cannot be considered good.

For data set 3, we already see some differences in the results, see Table 4. Neither algorithm can reach the correct global clustering structure due to the higher number of nested clusters. The unsuitability of SSE as the objective function is again illustrated by the fact that k-means on average gets CSI value of 0.71 compared to 0.58 of random swap. Random swap is better at optimizing the objective function and therefore reaches a lower SSE, but this also makes the solution diverge further from the ground truth, causing the decrease in CSI.

Typical solutions for both algorithms and data sets are shown in Figure 54. A notable difference between the *k*-means and random swap results is that, on average, random swap produces solutions with lower CSI value. This means that the clusters produced by random swap share fewer data vectors with the ground truth than the clusters produced by random swap, on average. This is because random swap is better at optimizing the objective function, as seen from the average SSE value which is lower for random swap. This in turn shows that SSE is not a good objective function for such data sets, as it does not take into account the centroid weights. In other words, the unsuitable objective function causes random swap to converge towards a solution that is further from the ground truth compared to *k*-means.

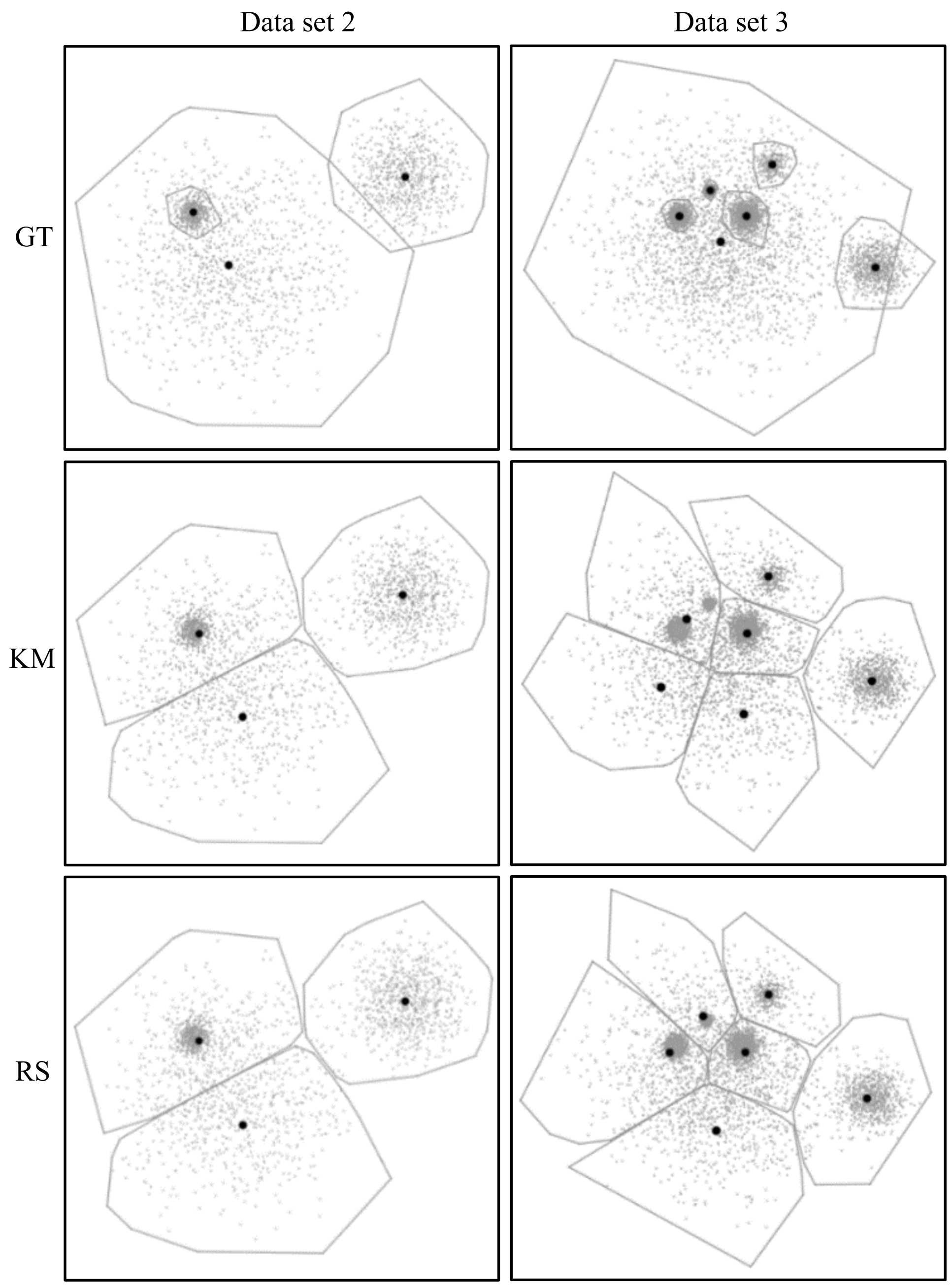


Figure 54. *k*-means and random swap produce similar results for both data sets.

Table 4. Results for data sets 2 and 3 using SSE as the objective function.

Algorithm	Success %	CI	CSI	Iterations to CI = 0	Time to CI = 0 (ms)	SSE ($\times 10^8$)
Data set 2						
k-means	100%	0.00	0.80	4.3	3.1	1.51
Random swap	100%	0.00	0.80	1.7	2.4	1.51
Data set 3						
k-means	0%	1.80	0.71	–	–	1.72
Random swap	0%	1.03	0.58	–	–	1.65

Table 5. Results for data sets 2 and 3 using weighted SSE as the objective function.

Algorithm	Success %	CI	CSI	Iterations to CI = 0	Time to CI = 0 (ms)	SSE ($\times 10^8$)
Data set 2						
k-means	54%	0.47	0.83	1.9	2.5	1.26
Random swap	100%	0.00	0.91	5.6	5.9	2.73
Data set 3						
k-means	5%	1.78	0.70	4.3	8.3	1.26
Random swap	62%	0.38	0.80	71	183.0	1.03

6.4 Weighted SSE results

For data set 2, the weighted version of k-means reaches $\mathrm{CI} = 0$ in 1.9 iterations, on average, with a success rate of 54%, see Table 5. The higher CI value of 0.47 can be explained by two factors: the dependency of k-means on the initial solution, and the differing weights for each centroid. Since k-means is only able to do local optimizations, it cannot solve problems in the global structure of the clustering. Therefore, if a centroid is initialized too far from its globally optimal position, it may not be able to find its way to the correct location, especially if its weight happens to be wrong. The average CSI is 0.83, which is slightly better than with SSE as the objective function. This reveals that although the CI is OK (0.47) compared to the unweighted results (0.00), on average the weighted version of k-means produced a better clustering.

The weighted version of random swap reaches $\mathrm{CI} = 0$ every time with an average CSI value of 0.91. The clustering quality is significantly better compared to using unweighted SSE as the objective function. The solutions produced by random swap are comparable to k-means, but on average the quality is better thanks to random swap's ability to solve problems in the global clustering structure. This is also evidenced by the lower average weighted SSE for random swap. Examples of successful clustering for both algorithms and data sets are shown in Figure 55.

With data set 3, k-means reaches the correct global clustering 5% of the time, on average. Notable here is that k-means can do it when we get lucky with the initial solution and the centroids are located near their globally optimal locations. This allows k-means to perform the necessary local fine-tuning to reach a good clustering. An example of such a solution is shown in Figure 55, which shows a partitioning that is very close to the ground truth.

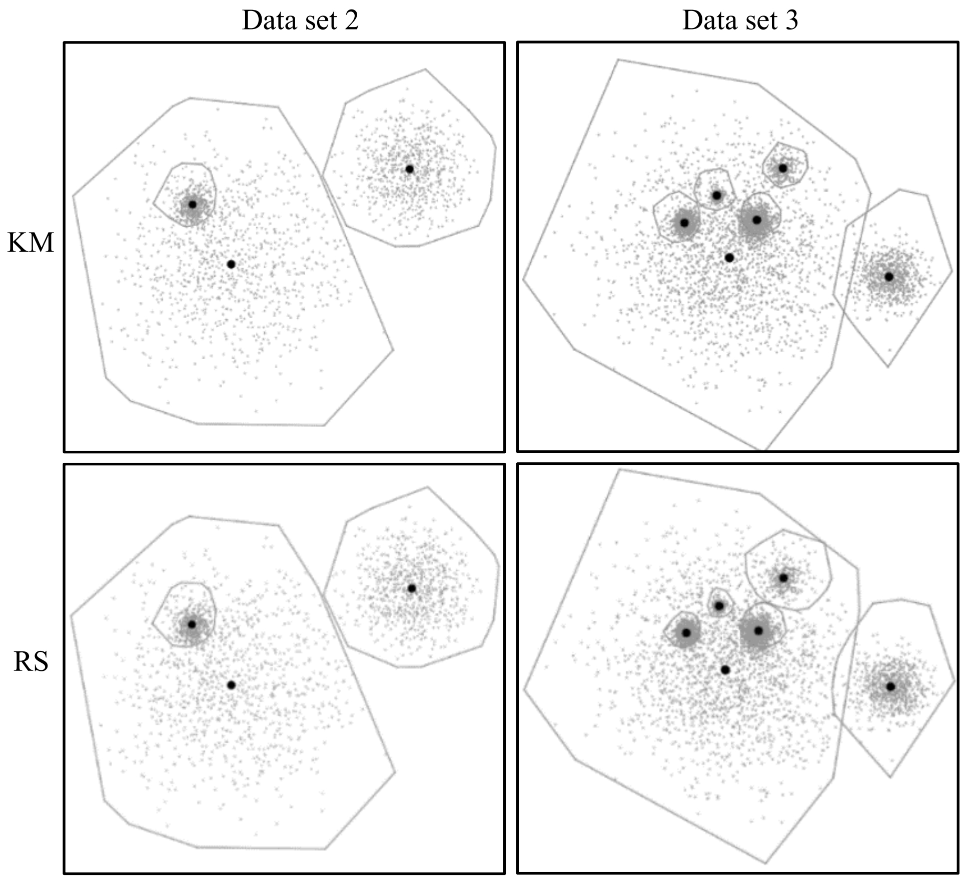


Figure 55. Examples of good solutions by *k*-means and random swap with weighted centroids.

Random swap reaches the correct global clustering structure 62% of the time and has an average CSI of 0.80. This is a significant improvement compared to using the un-weighted SSE as the objective function.

Typical solutions by both algorithms to both data sets are shown in Figure 56. Notable here is that *k*-means is able to find dense regions for many centroids, but the weights are usually wrong. In both cases, the centroids had poor initial positions and *k*-means was unable to find better locations for them.

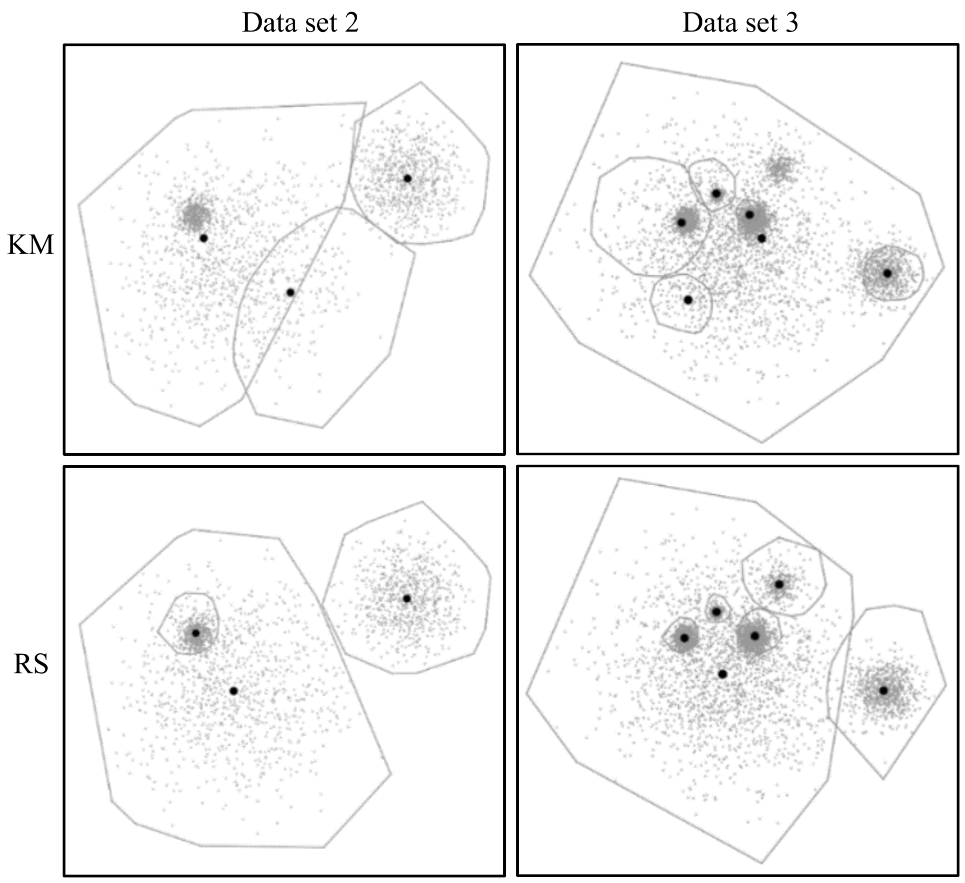


Figure 56. Typical solutions by *k*-means and random swap with weighted centroids.

Note that that for both data set, it is not possible to reach $\mathrm{CSI} = 1$ because they contain overlapping clusters. An example of the overlapped clusters is shown in Figure 57 for data set 2: a lot of the red data vectors belonging to the larger cluster are overlapped with the blue cluster.

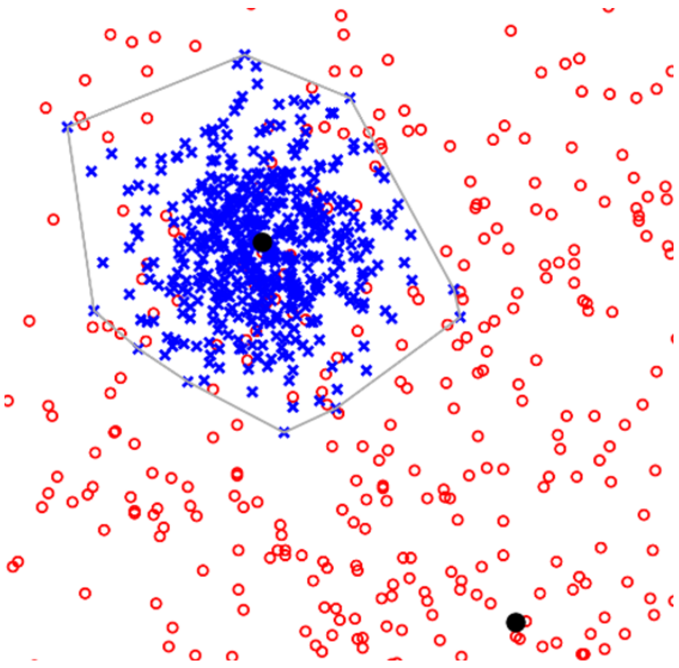


Figure 57. The ground truth clusters in data set 2 overlap.

Partitioning algorithms like k-means and random swap will assign all the points within the convex hull to the blue cluster, thus making it impossible to reach CSI=1 with these data sets.

Finally, notable in all the solutions is that the shape of the nested clusters is slightly distorted. An example of this is seen in Figure 58, where the cluster has a distinct asymmetric shape. This happens when data vectors that fall between the two clusters are more easily attracted to the centroid c_1 with lower weight. On the other hand, for the data vectors behind c_2 the weighted distance to c_1 becomes too long despite its lower weight. As a result, the nested cluster has asymmetric shape.

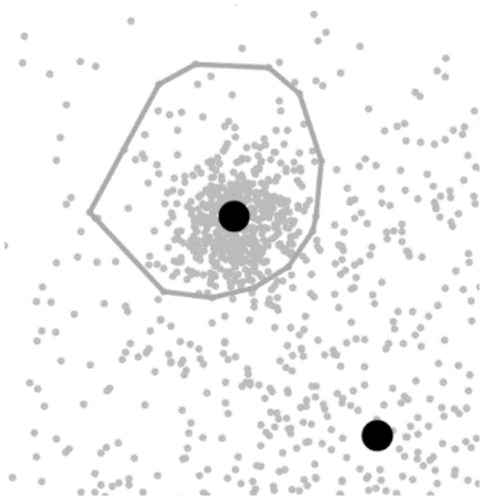


Figure 58. Asymmetry in nested clusters.

7 Conclusion

In this thesis, we have examined different density-based clustering methods for data sets with nested clusters of varying density. Two such data sets were used to evaluate some of the commonly cited algorithms found in literature. Nested clusters were found to be challenging for the existing algorithms. In most cases, the algorithms either did not detect the largest clusters, or simply divided large clusters into non-overlapping regions. In the former case, it may be possible to detect all clusters by doing multiple runs of the algorithm with different inputs, and then combining the results.

As a potential solution, we examined a method based on weighted centroids. We introduced weights for each centroid and used the weighted Euclidean distance to calculate the distance between data vectors and centroids. This way, data vectors are not necessarily assigned to the cluster with the nearest centroid, but the cluster with shortest weighted distance. Centroids with lower weight will therefore attract data vectors from further away than centroids with higher weight. This allows dense clusters inside larger clusters to be detected.

Experiments were performed using the *k*-means and random swap algorithms using both unweighted and weighted centroid approach. The centroid weights were calculated based on the true partitioning of each data set and given to the algorithms as additional parameters. Using unweighted centroids, the results were poor and neither algorithm could solve the correct centroid locations. The partitions also did not resemble the correct clustering. Large clusters were split into non-overlapping partitions instead of nested ones. With the weighted centroids, both algorithms were able to solve the correct centroid locations, with random swap doing so more reliably since it is better at optimizing the objective function. By using weights, the partitions produced by the algorithms also resembled the ground truth more accurately.

The drawback of this method is that the centroid weights must be given as an additional parameter. The partitions it produces are also not perfect. In particular, the shapes of nested clusters are slightly distorted, and the partitioning algorithms do not allow overlapping partitions.

Potential future research could include a more reliable method for calculating the centroid weights dynamically during the clustering process. In this thesis, a method that adds an additional weight calculation step to the clustering process was explored. It seems to work moderately well for simple a data set with not too many nested clusters. The problem with this approach is that it typically results in one or some of the centroids in the data set having low enough weight to attract data vectors from other clusters. For this method to work, a solution is therefore needed to address this problem.

An alternate approach could be to estimate the centroid weights as a pre-processing step before the actual clustering process. A simple way of doing this would be to construct histogram of the data. With two-dimensional data sets, this would essentially be a grid. The modes of the histogram would represent the cluster centers, and the density could be estimated based on how many data vectors fall within a cell. However, choosing the appropriate grid size may be difficult, as it depends on the data.

References

Bishop, C. M. (2007). *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer.

Campello, R. J., Moulavi, D., & Sander, J. (2013). Density-based clustering based on hierarchical density estimates. *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (160–172). https://doi.org/10.1007/978-3-642-37456-2_14

Celebi, M. E., Kingravi, H. A., & Vela, P. A. (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications, 40*(1), 200–210. https://doi.org/10.1016/j.eswa.2012.07.021

Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 17*(8), 790–799. https://doi.org/10.1109/34.400568

Comaniciu, D., & Meer, P. (1999). Mean shift analysis and applications. *Proceedings of the Seventh IEEE International Conference on Computer Vision.* https://doi.org/10.1109/iccv.1999.790416

Comaniciu, D., & Meer, P. (2002). Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24*(5), 603–619. https://doi.org/10.1109/34.1000236

De Brito, D. M., Maracaja-Coutinho, V., de Farias, S. T., Batista, L. V., & do Rêgo, T. G. (2016). A Novel Method to Predict Genomic Islands Based on Mean Shift Clustering Algorithm. *PLOS ONE, 11*(1). https://doi.org/10.0.5.91/journal.pone.0146352

Ertöz, L., Steinbach, M., & Kumar, V. (2003). Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. *Proceedings of the 2003 SIAM International Conference on Data Mining,* 47–58. https://doi.org/10.1137/1.9781611972733.5

Ester, M., Kriegel, H. P., Sander, J., Wimmer, M., & Xu, X. (1998). Incremental clustering for mining in a data warehousing environment. *Proceedings of the 24th*

VLDB Conference (VLDB'98). New York, 323–333. https://doi.org/10.1007/BFb0100982

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996). 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, 226–231.

Estivill-Castro, V. (2002). Why so many clustering algorithms. *ACM SIGKDD Explorations Newsletter, 4*(1), 65–75. https://doi.org/10.1145/568574.568575

Everitt, B. S., Landau, S., Leese, M., Stahl, D. (2011). *Cluster Analysis* (5th ed.). Chichester, West Sussex, UK: Wiley. https://doi.org/10.1002/9780470977811

Fahad, A., Alshatri, N., Tari, Z., Alamri, A., Khalil, I., Zomaya, A. Y., Foufou, S. & Bouras, A. (2014). A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE transactions on emerging topics in computing, 2*(3), 267–279. https://doi.org/10.1109/TETC.2014.2330519

Fraley, C., & Raftery, A. E. (1998). How many clusters? Which clustering method? Answers via model-based cluster analysis. *The Computer Journal, 41*(8), 578–588. https://doi.org/10.1093/comjnl/41.8.578

Fränti, P. (2018). Efficiency of random swap clustering. *Journal of Big Data, 5*(1), 1–29. https://doi.org/10.1186/s40537-018-0122-y

Fränti, P., & Kivijärvi, J. (2000). Randomised local search algorithm for the clustering problem. *Pattern Analysis & Applications, 3*(4), 358–369. https://doi.org/10.1007/s100440070007

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

Fränti, P., Virmajoki, O., & Hautamäki, V. (2006). Fast agglomerative clustering using a k-nearest neighbor graph. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 28*(11), 1875–1881. https://doi.org/10.1109/TPAMI.2006.227

Fukunaga, K., & Hostetler, L. (1975). The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on information theory, 21*(1), 32–40. https://doi.org/10.1109/tit.1975.1055330

Kaufman, L., & Rousseeuw, P. J. (2005). *Finding groups in data: an introduction to cluster analysis*. Hoboken, NJ, USA: Wiley.

Kinnunen, T., Sidoroff, I., Tuononen, M., & Fränti, P. (2011). Comparison of clustering methods: A case study of text-independent speaker modeling. *Pattern Recognition Letters, 32*(13), 1604–1617. https://doi.org/10.1016/j.patrec.2011.06.023

Koyuncu, E., & Jafarkhani, H. (2017). On the minimum average distortion of quantizers with index-dependent distortion measures. *IEEE Transactions on Signal Processing, 65*(17), 4655–4669. https://doi.org/10.1109/TSP.2017.2716899

Kotsiantis, S., & Pintelas, P. (2004). Recent advances in clustering: A brief survey. *WSEAS Transactions on Information Science and Applications, 1*(1), 73–81.

Kriegel, H. P., Kröger, P., Sander, J., & Zimek, A. (2011). Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 1*(3), 231–240. https://doi.org/10.1002/widm.30

Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern recognition letters, 31*(8), 651–666. https://doi.org/10.1016/j.patrec.2009.09.011

Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. Upper Saddle River, NJ, USA: Prentice-Hall, 118–119.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. *ACM computing surveys (CSUR), 31*(3), 264–323. https://doi.org/10.1145/331499.331504

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, 1*(14), 281–297.

Mahajan, M., Nimbhorkar, P., & Varadarajan, K. (2009). The planar k-means problem is NP-hard. *International Workshop on Algorithms and Computation*. Berlin: Springer, 274–285. https://doi.org/10.1007/978-3-642-00202-1_24

Melnykov, V., Michael, S., & Melnykov, I. (2014). Recent developments in model-based clustering with applications. *Partitional Clustering Algorithms.* 1–39. https://doi.org/10.1007/978-3-319-09259-1_1

Navarro, G. (2001). A guided tour to approximate string matching. *ACM computing surveys (CSUR), 33*(1), 31–88. https://doi.org/10.1145/375360.375365

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*(Oct), 2825–2830.

Ristad, E. S., & Yianilos, P. N. (1998). Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 20*(5), 522–532. https://doi.org/10.1109/34.682181

Rodriguez, A., & Laio, A. (2014). Clustering by fast search and find of density peaks. *Science, 306*, 1910–1913. https://doi.org/10.1126/science.1242072

Theodoridis, S. & Koutroumbas, K. (2009). *Pattern recognition* (4th ed.) Burlington, MA, USA: Academic Press.

Xu, D., & Tian, Y. (2015). A comprehensive survey of clustering algorithms. *Annals of Data Science, 2*(2), 165–193. https://doi.org/10.1007/s40745-015-0040-1