Institute for Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany

Master Thesis

# Evaluation of Prediction Mechanisms of Parameters for Data Mining Algorithms

Kinda Loutfi

| | |
|---|---|
| **Study Program:** | Computer Science |
| **Examiner:** | PD Dr. rer. nat. habil. Holger Schwarz |
| **Advisors:** | Manuel Fritz, M.Sc.<br>Dipl.-Inf. Michael Behringer |
| **Start date:** | 01.11.2017 |
| **End date:** | 01.05.2018 |

## Abstract

Extracting knowledge and useful information from huge amount of data is one of the biggest issues currently in the world of computer science. Data mining is one essential step in the knowledge discovery process due to the importance of its contribution in extracting this knowledge.

One of the most famous mining techniques is Clustering, which is a widely used approach in data mining. Clustering is the process of partitioning a group of objects into smaller sets called clusters, in which similar objects are assigned to the same cluster, and dissimilar objects are assigned to different clusters.

K-Means, DBSCAN, and OPTICS are three of the most popular clustering algorithms which are used to group similar data into clusters. Each of these algorithms requires input parameters. The difficulty of knowing these input parameters in advance is the flaw of these algorithms. Many previous approaches were introduced which provide a prediction of these parameters. However, different problems emerged while using these approaches, such as a long overall runtime to achieve predicted parameter values, since the clustering algorithm is applied multiple times with varying parameters to identify the best parameter configuration.

This thesis introduces a new approach for predicting the input parameters. The proposed approach is called PROD (**P**osition-Based P**r**ediction Using V**o**ronoi **D**iagrams). PROD overcomes the problems which emerged in previous approaches. The prediction of the input parameters is performed by this new approach through using space partitioning approaches and subsequently exploiting their properties.

PROD is evaluated on various data sets. Regarding K-Means algorithm; PROD provides a prediction of number of clusters as well as a prediction of the initial centroids. The experimental results unveil, that PROD is (a) more accurate and (b) faster by a factor of 26.5 in contrast to previously best existing approaches. Additionally, the results of the evaluations show that it can be used either as a prediction approach for the amount of clusters or as a standalone clustering algorithm.

Despite the effectiveness of the prediction made by PROD with regard to the K-Means parameters, the results of the experiments show that this novel approach needs further improvement to make the prediction of DBSCAN and OPTICS parameters work. More clearly, changing some aspects of PROD might lead to a better prediction with regard to these two algorithms.

# Table of Contents

1. Introduction ........ 1
2. Preliminaries ........ 4
   - 2.1 Data Mining ........ 4
   - 2.2 Clustering Technique ........ 4
     - 2.2.1 K-Means ........ 5
     - 2.2.2 DBSCAN ........ 6
     - 2.2.3 OPTICS ........ 7
   - 2.3 Space Partitioning ........ 9
     - Voronoi Diagram and Delaunay Tessellation ........ 9
   - 2.4 Sampling Algorithms ........ 11
     - 2.4.1 Random Sampling Method Without Replacement ........ 11
     - 2.4.2 Systematic Sampling Method Without Replacement ........ 12
3. Related Work ........ 13
   - 3.1 Previous Prediction Methods for Density-Based Clustering Algorithms ........ 13
     - 3.1.1 Determining DBSCAN Parameters ........ 13
     - 3.1.2 Determining OPTICS Parameters ........ 14
   - 3.2 Previous Prediction Methods for K-Means Algorithm ........ 15
     - 3.2.1 Enhanced K-Means Clustering Algorithm Using a Heuristic Approach ........ 15
     - 3.2.2 Improved K-Means Algorithm Based on the Clustering Reliability Analysis ........ 15
     - 3.2.3 The Global K-Means Clustering Algorithm ........ 16
     - 3.2.4 The Elbow Method ........ 17
     - 3.2.5 The Information-Theoretic Approach ........ 17
     - 3.2.6 The Silhouette Method ........ 18
4. Position-Based Prediction Using Voronoi Diagrams ........ 19
   - 4.1 Overview ........ 19
   - 4.2 Sampling ........ 21
     - 4.2.1 Position-Based Sampling Approach ........ 22
     - 4.2.2 Complexity of Position-Based Sampling ........ 24
   - 4.3 Space Partitioning ........ 24
   - 4.4 Fine Tuning ........ 24
     - 4.4.1 Scattering the Data Points ........ 25
     - 4.4.2 Inspection of Voronoi Regions ........ 25
     - 4.4.3 Merging Voronoi Regions ........ 26
     - 4.4.4 Complexity of the Fine Tuning ........ 26
   - 4.5 Predicting Clustering Parameters ........ 27

4.5.1 Predicting Input Parameter for K-Means Algorithm .....27
4.5.2 Predicting Input Parameters for Density-Based Clustering Algorithms .....27
4.5.3 Complexity of the Prediction .....27
4.6 Identifying Outliers .....28
4.7 The Overall Complexity of PROD .....29
5. Implementation .....30
5.1 Class Diagram .....30
5.1.1 The *kdtnearest* Package .....31
5.1.2 The *sampling* Package .....31
5.1.3 The *spacePartitioning* Package .....32
5.1.4 The *fineTuning* Package .....33
5.1.5 The *prediction* Package .....34
5.1.6 The *com.mavenproject.predictingparams* Package .....35
5.2 Libraries .....36
5.2.1 Open Source Implementation of Nearest-Neighbor Search .....37
5.2.2 Voronoi Diagram Library .....37
5.2.3 Mining Algorithms Library .....38
5.2.4 Implementation of Previous Prediction Approaches .....39
5.3 Python Visualization .....40
5.4 Data Sets .....41
6. Evaluation .....43
6.1 Sampling Evaluation .....43
6.2 Space Partitioning Evaluation .....46
6.3 Fine Tuning Evaluation .....47
6.3.1 Evaluation of Different Settings of the Threshold Distance .....47
6.3.2 Evaluation of the Required Time .....55
6.4 Evaluation of the Overall PROD Approach .....56
6.4.1 Evaluation of Predicting K-Means Parameter .....56
6.4.2 Evaluation of Predicting DBSCAN and OPTICS Parameters .....69
7. Conclusion .....77
8. Outlook .....80
9. References .....82
10. Appendix .....86

# 1. Introduction

The world currently is in the big data era. Data is the basis on which users are performing their studies and analyses as it gives tremendous value for these analyses. This fact is leading to a rapid increase in the number of different available data sets [1]. Moreover, internet and automated data collection tools are also contributing, on one hand, in increasing the number of data sets stored in databases and in other data repositories, and on the other hand, in growing the size of each of the stored data sets. This huge amount of data is beneficial but it has one main issue, which is lacking of knowledge.

To overcome this issue, enriching the world with the required knowledge requires analyzing this huge amount of data sets in order to extract the necessary knowledge, which can lead to a better understanding of the data. However, analyzing such huge amount of data is cumbersome to be performed manually by the analysts. Hence, the analysts need help in performing such an analytical process. More specifically, a data analytic process, which guides and helps the analysts, is required in order to ease the process of eliciting knowledge from the available data sets.

There exist multiple reference process models, which guide the analysis process. For instance, one important reference process model is *Knowledge Discovery in Databases (KDD)* [2]. The KDD process model consists of the following main steps: (1) Data Selection such that this step includes selecting the application-relevant target data sets and focusing on the data on which the discovery will be performed, (2) Preprocessing and cleaning in which data reliability is enhanced through handling missing values and removing noise from data, (3) Data Transformation in which the existing data is transformed to match the needs of the subsequent mining step better, (4) Data Mining such that this step includes: choosing the mining technique, choosing the mining algorithms, applying these mining algorithms, and generating the models, (5) Evaluation and interpretation of the generated models and patterns, and finally (6) Using the discovered knowledge by applying the generated models. Similarly, there are other reference process models, e.g., CRISP-DM [41], which have similar steps as the KDD process.

There are no exact guidelines on how to perform each of these steps. Whereas, the analysts need to explore the solution space along the analytics process in order to find proper options. Having prior domain knowledge about the specific data set, as well as, prior technical knowledge about the valid analytical options; might help in reducing the solution space. However, having the domain and technical knowledge in advance is very difficult. One analysis on data and analytics made by Gartner indicates that by the year 2020; more than 40 percent of all tasks made by data scientists will be automated [3]. Furthermore, a research vice president at Gartner said, "*The key to simplicity is the automation of tasks that are repetitive, manual intensive and don't require deep data science expertise*" [3]. Obviously, this means that the technical knowledge needed by the analysts in order to perform data science tasks will not be needed as much as before. Although the latter fact will help the analysts in performing the analytical process without the need of having a technical knowledge; it is still of importance, for reducing the solution space, to have a domain knowledge about the specific context of the analysis. Without having any prior knowledge; it will be a very difficult process to be performed by the analysts due to the huge increasing amount of data sets.

Data mining plays an essential role in the knowledge discovery process [4]. The importance of this step comes from the fact that the models, which are used to deliver new necessary knowledge, are generated in this step. Data mining uses machine learning algorithms, i.e., mining algorithms, in order to deliver new knowledge. Mining algorithms are applied on data sets from which the analyst needs to extract some hidden knowledge. In general, mining algorithms require input parameters in order to be executed. These parameters should be chosen by the user and they are specific and dependent for each data set in order to obtain solid results. This fact means that the analysts need, for each data set, to explore the solution space of all the possible input parameters, which is a very time-consuming process especially for big data sets. However, having a prior domain knowledge about the specific data set, can reduce the solution space of the possible valid input parameters, i.e., will ease the exploration process made by the analysts. For example, if the analyst knows exactly, how many clusters are expected in a dataset, then he knows exactly what the input parameters should be. Since having this knowledge in advance by the analysts cannot be guaranteed; another approach for reducing the solution space is required.

In general, machine learning is mainly divided into two main parts: either (1) supervised learning, or (2) unsupervised learning. In supervised learning, e.g., Classification, a prediction for some expected target value is performed. More clearly, predictive models are generated and applied on the data set. Having the result containing the predicted target value; this result can be compared to the actual expected target value in order to assess the validity of the result. On the other hand, in unsupervised learning, e.g., Clustering, no prediction is made and no target value is known in advance. More specifically, descriptive models are generated and applied to the data set. These descriptive models focus on the structure and interconnectedness of the data. The latter means that the validity of result can only be assessed by the analysts and by their domain knowledge.

The fact that the validity of the result predicted by the predictive models, in the supervised learning, can be assessed through comparing it to the actual target value, lead to the ability of performing this assessment automatically. In fact, for supervised machine learning tasks, the problem of finding good algorithms and good parameters is tackled by using so-called Black-Box Optimization techniques [42]. One concrete instantiation of a Black-Box Optimization technique is the SMBO method (Sequential Model-Based Optimization) [43]. Furthermore, an implementation of the SMBO method called Auto-WEKA [6, 43], which is an extension provided by the machine learning platform WEKA [5]. Auto-WEKA helps analysts by *automatically* searching and choosing the best parameters setting which yields to the best result, i.e., a model which achieves the best match between predicted class and expected class. These Black-Box Optimization techniques help the analysts in exploring the solution space for the supervised machine learning tasks, i.e., it makes the exploration much easier. However, Black-Box Optimization techniques have some downsides. For example, one important downside is that they often require huge amount of time to be performed [43].

On the other hand, in case of unsupervised learning, having such automated tool for assessing the different results of applying the algorithms with different parameters settings, is not achievable since there are no previous expectations about the result. More clearly, the analysts still need to manually explore the solution space of valid input parameters for unsupervised learning. This fact means that, in unsupervised learning, the solution space exploration is much harder to be performed by analysts.

It is of huge benefit to the analysts to be guided while exploring the solution space, as well as, to be provided with a reduced solution space which ease the exploration process. In other words, helping the analysts through providing a prediction for the specific input parameters that are required by the mining algorithms for each data set. Currently, there exist some heuristics and best practices for predicting input parameters for mining algorithms [15, 16, 19, 20, 23, 24, 25, 26, 27]. However, some of these heuristics have some issues. One of the most important issues for some of these heuristics is that they require applying the original mining algorithm multiple times in order to perform the prediction of the parameters [20, 23, 24, 25, 26, 27]. Furthermore, some heuristics are not feasible for big data due to their high complexity [24, 25].

Consequently, it is of importance to find a new prediction approach which is able to overcome the previous problems from which the previous prediction approaches are suffering. More clearly, this new approach should be able to: (1) correctly predict input parameters in a reasonable time, i.e., feasible for big data, and (2) produce the prediction without the need of applying the original mining algorithm multiple times.

Space partitioning approaches [37], e.g., Voronoi diagrams [7], are promising methodologies to be used in this new prediction approach. More specifically, space partitioning algorithms have different properties which will help to overcome the previously mentioned problems. Some of the properties of space partitioning algorithms are: (1) they have reasonable time complexity which make them feasible for big data, and (2) they can be used to perform the prediction instead of applying the mining algorithm multiple times.

Our goal is to find a new approach that performs a prediction for the parameters using one of the space partitioning approaches, e.g., Voronoi diagrams. Furthermore, since the main issue, as described before, lies in the difficulty of exploring the solution space in unsupervised learning, hence, this new approach will be focusing on reducing the explored solution space for the unsupervised machine learning tasks. Since clustering is a famous unsupervised learning technique; therefore, the most common three clustering algorithms: K-Means, DBSCAN, and OPTICS are chosen. More precisely, the goal of the new prediction approach is to predict the input parameters required by these three chosen clustering algorithms, using Voronoi diagrams. This prediction helps in reducing the solution space of the input parameters required by the clustering algorithm for each data set.

The subsequent sections are organized as follows: In Section 2, the preliminaries of each of the used concepts and approaches: Data Mining, Clustering Techniques, Space Partitioning approaches, and Sampling approaches are introduced. Section 3 describes related work in which some of the previous approaches for predicting input parameters of mining algorithms are described. Later on, Section 4 introduces the Position-Based Prediction Using Voronoi Diagrams (PROD) approach. Followed by Section 5 which describes the experimental setup of this proposed approach. Furthermore, an evaluation of PROD is performed and described in Section 6. Section 7 concludes the discussion, and Section 8 refers to an outlook about the possible future work.

# 2. Preliminaries

In order to understand the workflow of the new prediction approach; it is necessary as a first step to understand the basics of some concepts and approaches used in building this new prediction approach. Therefore, this section introduces the preliminaries about each of these concepts and approaches. More specifically, Subsection 2.1 describes Data Mining, Subsection 2.2 introduces Clustering Techniques, Subsection 2.3 shows Space Partitioning approaches, and in Subsection 2.4, Sampling approaches are introduced.

## 2.1 Data Mining

Data mining is one substantial and valuable step in the Knowledge Discovery Process [4]. Having huge amount of data; it is of huge importance and of tremendous benefit to discover useful information and knowledge from this data. The importance of the data mining step arises from the need of discovering this knowledge. In data mining, huge amount of data contained in large data sets is analyzed in order to discover patterns that can be later interpreted and evaluated to generate new knowledge. This analysis is done without any prior knowledge about the data and without any expectations about the result of this analysis.

Discovering new knowledge is done through using different mining techniques and statistical approaches in order to generate models which deliver this new knowledge. There exist multiple mining techniques, and the most common four techniques are: Association Rule Discovery, Clustering, Classification, and Regression. Each of these techniques is either: predictive, e.g., Classification and Regression, or descriptive, e.g., Clustering and Association Rule Discovery. Each mining technique has different algorithms which realize it. For instance, the most common algorithms which fall under clustering technique are K-Means, DBSCAN, and OPTICS.

Most mining algorithms require a set of input parameters in order to be executed, and the usefulness of the result of the algorithm depends on the quality of these parameters such that wrong parameters lead to mining results which are not desired by the analysts. The main issue is that the quality of the algorithm and parameters cannot be estimated until the algorithm is executed completely. Hence, the analysts need to iterate over multiple combinations of algorithms and parameters in order to decide for the best combination; which is very time-consuming, especially when considering an increasing amount of data to analyze. This issue was partially solved for supervised learning, e.g., Classification, through providing a technique that is able to explore the solution space automatically. However, regarding unsupervised learning, e.g., Clustering, the exploration of the solution space is still a highly difficult process for an analyst to perform manually. Thus, it is of importance to focus on solving the problem of the difficulty of this task for unsupervised learning. In Subsection 2.2, an explanation of Clustering Techniques, which are considered as unsupervised learning, will be provided and examples of the most famous clustering algorithms will be introduced.

## 2.2 Clustering Technique

One of the mining techniques used to discover new knowledge is clustering [8]. Clustering is one technique used in unsupervised learning, in order to generate descriptive models that describe the structure and relations in the data set. Given $n$ items in the space, the goal of clustering is to group similar data objects into $k$ clusters. These $k$ clusters have two main properties: (1) the items contained in one cluster are similar, i.e., maximizing the intra-cluster similarity, and (2) the items contained in different clusters are different, i.e., minimizing the inter-cluster similarity.

Clustering techniques can be mainly realized by five different clustering methods [8]. These methods are: Partitioning Methods, Density-Based Methods, Hierarchical Methods, Grid-Based Methods, and Model-Based Methods. Each of these five main methods of clustering has many algorithms which realize it. For instance, the most common algorithm used for the partition-based clustering is K-Means [9]. Moreover, regarding the density-based clustering methods; the two most famous algorithms used are DBSCAN [13, 14, 15] and OPTICS [16]. Each of these clustering algorithms requires, for each data set, different input parameters in order to perform the clustering of this data set. Since these three clustering algorithms are the most commonly used algorithms, hence, the focus will be on them in the subsequent sections.

### 2.2.1 K-Means

K-Means is a partitioning-based clustering algorithm [9, 44]. This algorithm partitions a set of data items $S$ into $k$ clusters such that each cluster has a centroid (central point) in which this centroid is the mean of all items belonging to this cluster. The main idea of K-Means is minimizing the Sum of Squared Error (SSE), which measures the total squared Euclidian distance of data items to their centroid, while performing the partitioning [9, 44].

The algorithm starts with choosing random centroids, and then multiple iterations will be performed until no entities are changed, which means that the minimum SSE is achieved. In each iteration, the Euclidean distance is calculated between each data item and each centroid in order to accordingly assign this item to the cluster of the nearest centroid. Then, the centroid of each cluster is updated through calculating the mean of all data items which were assigned to this cluster. Algorithm 1 presents this described algorithm in pseudo code.

K-Means algorithm has a high popularity because of many advantages. One of the most important reason for its widely usage is its linear complexity. The complexity of $i$ iterations in which $n$ data items are partitioned into $k$ clusters is $O(i * n * k)$ [44]. This fact makes the algorithm very efficient to be performed even on very large data sets. Other advantages of this algorithm are the simplicity and the speed of convergence.

However, as any other algorithm, K-Means has some disadvantages. One critical disadvantage is its sensitivity to the selection of both the initial centroids and the number of clusters $k$ since these selections can affect the result. Many methods were proposed to solve the problem of the initial selection of the centroids [10, 11]. The selection of $k$ is very crucial; however, it is not trivial and it needs a prior knowledge by the analyst. Furthermore, this algorithm is very sensitive to outliers because one outlier can dramatically increase the squared error. Another disadvantage of K-Means is its limitation to numeric data only; however, this limitation was solved by *Huang* who introduced the K-prototypes algorithm which is based on K-Means algorithm but can be used to cluster categorical data in addition [12].

```
Input: S, k
Output: clusters
Randomly choose $k$ cluster centroids.
repeat
    Calculate Euclidean distances and assign data items to the nearest centroid.
    Re-calculate cluster centroids according to the constructed clusters.
until The clusters' centroids do not change anymore
```

*Algorithm 1.* K-Means Algorithm.

### 2.2.2 DBSCAN

DBSCAN (**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise) is a density-based clustering algorithm which is able to discover clusters of arbitrary and complex shapes [13, 14, 15]. DBSCAN, in contrast to K-Means, can handle noise, i.e., outlier data points, which means that it does not necessarily group all data points into clusters. Furthermore, using spatial access methods; this algorithm is also efficient for large spatial databases. DBSCAN algorithm requires two input parameters: $\varepsilon$ and *MinPts*, and works in a two-phase approach in order to find clusters. The parameter $\varepsilon$ defines a distance from a data point to another data point in the space. More specifically, the *ε-Neighborhood* of a data point p defines a neighborhood that contains all other data points which have a distance to *p* less or equal *ε*. Furthermore, the second input parameter for the DBSCAN is called *MinPts*. This parameter defines the number of data points that should be contained in the *ε-Neighborhood* of a data point. More clearly, a data point *p* should have in its *ε-Neighborhood* at least *MinPts* data points in order to form a cluster.

Each cluster, resulted from applying DBSCAN algorithm, is considered to have two types of points: border points and core points. DBSCAN is mainly based on three concepts: (1) The *ε-Neighborhood* of a point *p* which consists of all points that have a distance from *p* less than a threshold *ε*, i.e., $N_\varepsilon(p) = \{q \in D \mid dist(p,q) < \varepsilon\}$, (2) The Direct Density-Reachability such that a point *p* is directly density-reachable from point *q* if *q* is a core point and *p* belongs to the *ε-Neighborhood* of *q*, and (3) The Density-Reachability. More specifically, a point *p* is density-reachable from point *q*, if there is a chain of points $o_1, \dots, o_n \mid o_1 = q \; and \;\; o_n = p$ such that $q$ is a core point and each $o_{i+1}$ is directly density-reachable from $o_i$, where $1 \le i \le n$. Generally, a border point has less number of points in its *ε-Neighborhood* than a core point. More precisely, in order for a point $p$ to be a core point; it has to have number of points in its *ε-Neighborhood* which is larger than a threshold *MinPts*, i.e., $|N_\varepsilon(p)| > MinPts$.

DBSCAN algorithm considers that a point $q$ belongs to a cluster $C$ if there is a core point $p \in C$ such that $q$ is density-reachable from $p$. More generally, a cluster $C$ is defined as the set of points which are density-reachable from a core point *p* in *C*. DBSCAN algorithm, using the two parameters *ε* and *MinPts*, works in a two-phase approach in order to find clusters. Firstly, the *ε-Neighborhood* of each point $p$ is checked to determine whether $p$ is a core point or not. In case $p$ satisfies the core point condition; a new cluster will be created. Secondly, all points which are density-reachable from $p$ are identified and added to the cluster containing $p$. The identification of the density-reachable points from $p$ is done iteratively through following a procedure consisting of three steps: (1) collecting the directly density-reachable points from $p$, (2) checking the *ε-Neighborhood* of each of these collected points, (3) in case any of these points identified as a core point; its *ε-Neighborhood* will also be added to the cluster and will be checked in the next step. This procedure is repeated as long as there are new points that can be added to the current cluster. When no new points can be added to the current cluster; DBSCAN identifies a new core point from the space and creates a new cluster as described above.

The $\varepsilon$-Neighborhood of a data point is considered to be relatively small; and retrieving this $\varepsilon$-Neighborhood can be performed by spatial access methods like R*-trees which have, in the worst case, a height of $O(logn)$ for *n* points. Having *n* data points, this yields that the average runtime complexity of retrieving the $\varepsilon$-Neighborhood of a point is $O(logn)$ and the average runtime complexity of DBSCAN is $O(n * logn)$ since for each data point the algorithm retrieves the $\varepsilon$-Neighborhood at most once.

The DBSCAN algorithm has several pros which make it superior when comparing it with other algorithms. One important strength point of this algorithm is that it can easily determine which data points should be identified as outliers, in other words, it can identify data points that do not belong to any of the clusters. Moreover, another important advantage of DBSCAN is its efficiency on large spatial databases. More specifically, using spatial access methods, DBSCAN has almost linear increase in computing time relatively to the total number of data points [14]. More precisely, applying DBSCAN algorithm for spatial databases and using spatial access methods has an average runtime complexity of $O(n * logn)$ [15]. On the other hand, for other databases; DBSCAN algorithm has a runtime complexity of $O(n^2)$ in worst case, which is one disadvantage of this algorithm. Furthermore, DBSCAN has another important disadvantage which is the inability to differentiate between clusters located closely to each other but have different densities.

### 2.2.3 OPTICS

OPTICS (**O**rdering **P**oints **T**o **I**dentify the **C**lustering **S**tructure) is a density-based cluster-ordering algorithm which does not explicitly generate clustering of the data set, in contrast to other density-based algorithms, whereas it generates an augmented ordering of this data set showing its density-based clustering structure [16]. The generated information by OPTICS is sufficient to construct density-based clusters corresponding to a wide range of parameter settings. In other words, OPTICS can find clusters with varying densities in a single pass in contrast to DBSCAN which finds clusters only with the same density in a single pass.

Similar to DBSCAN, OPTICS can handle outlier data points. Furthermore, OPTICS algorithm is based on the same concepts as DBSCAN, however, it only requires one input parameter which is the *MinPts*. This algorithm starts with a *generating distance* $\varepsilon$, which is the radius of the neighborhood, and it produces information about each clustering level starting from the minimum possible radius up to this $\varepsilon$. The value of $\varepsilon$ represents the density of the clusters, more specifically, lower values of $\varepsilon$ corresponds to higher densities as the neighborhood is smaller and *MinPts* should exist in this smaller neighborhood which yields to a cluster of higher density. Figure 1 shows one important observation in the density-based cluster-ordering which is for a specific value of *MinPts*; density-based clusters corresponding to higher density, e.g. Clusters *C1* and *C2*, are contained completely in density-based clusters corresponding to lower density, e.g., Cluster *C*.

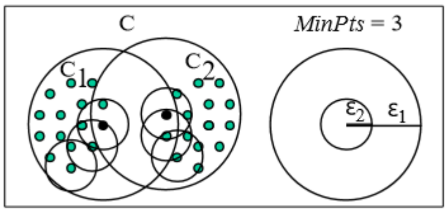


*Figure 1. Nested Density-Based Clusters [16].*

The OPTICS algorithm assigns two values: (1) the *core-distance* and (2) the *reachability-distance* with each point in the data set. These two values can be used, by DBSCAN algorithm, to construct the clusters and assign cluster memberships for each point. Given the parameter *MinPts* and a point $p$ with its $\varepsilon$-neighborhood $N_{\varepsilon}(p)$; the core-distance of $p$ is the smallest distance $\varepsilon'$ between $p$ and another point $q \in N_{\varepsilon}(p)$ in which $p$ is a core point with respect to $\varepsilon'$, and in case $p$ is not a core point; the core-distance is UNDEFINED. Moreover, the reachability-distance of $p$ with respect to another point $o$ is either the smallest distance in which $p$ is directly density-reachable from $o$ if $o$ is a core point, or UNDEFINED if $o$ is not a core point.

In order to produce these two values; OPTICS algorithm process in the following steps. For each point $p$ in the data set; if it is not processed yet, the algorithm retrieves its $\varepsilon$-neighborhood, sets its reachability-distance to UNDEFINED, and determines its core-distance. In case $p$ is not a core distance; the algorithm leaves $p$ and moves to the next unprocessed point in the data set. However, if $p$ is a core point at a distance $\leq \varepsilon$ then the algorithm retrieves all the directly density-reachable points from $p$ and puts them in an ordered list depending on their reachability-distances from the closest core point from which they are directly density-reachable. OPTICS proceeds by choosing from this ordered list the point with the smallest distance, retrieving its $\varepsilon$-neighborhood, determining its core-distance, and finally checking if it is a core point in order to collect its directly density-reachable points and inserting them into the list.

After applying OPTICS, using the produced cluster-ordering with respect to *MinPts* and the generating-distance $\varepsilon$; we can extract any density-based clustering with respect to this *MinPts* and to any chosen distance $\varepsilon' \leq \varepsilon$. By following a simple procedure, the density-based clusters can be extracted through assigning cluster memberships to the points depending on their generated reachability-distance and core-distance [16].

The runtime of OPTICS, similarly to DBSCAN, is heavily dependent on the runtime of retrieving the $\varepsilon$-neighborhood which is needed for each point in the data set. More specifically, having $n$ points in the data set; the complexity of both algorithms OPTICS and DBSCAN is $O(n *$runtime of retrieving $\varepsilon$-neighborhood). Using tree-based spatial index; the runtime of OPTICS is $O(n * logn)$ for $n$ data points. Moreover, if the points are organized in a grid then a direct access could be performed and the runtime of OPTICS would be reduced to $O(n)$. An extensive performance test showed that the runtime of OPTICS is approximately 1.6 times the runtime of DBSCAN [16]. On the other hand, one of the advantages of OPTICS over DBSCAN is that OPTICS needs only one input parameter which is the *MinPts* whereas DBSCAN needs two input parameters *MinPts* and the clustering distance $\varepsilon$. Moreover, a major strength point of OPTICS is that the generated cluster-ordering contains information equivalent to applying DBSCAN with multiple parameter settings, i.e., OPTICS is not limited to one global parameter setting.

## 2.3 Space Partitioning

The main idea of space partitioning approaches is to divide a given space into multiple smaller regions [37]. The partitioning of a space was mainly introduced for enhancing the search tasks applied on this space, and make them faster [38]. Moreover, the idea of partitioning the space helps in reducing the complexity of the processing applied on the overall space. The latter is done through dividing this processing to local processing on the resulted smaller regions. Space partitioning approaches are applied either: (1) using algorithmic approaches such as KD Trees [38] and R-Trees [39], which are index structures for spatial databases allowing for a faster processing on these databases, or (2) using visual approaches such as Voronoi diagrams and Delaunay tessellations [7].

A Voronoi diagram is a canonical method for partitioning a space into smaller regions. Voronoi diagrams and Delaunay tessellations partition the space visually. More clearly, through applying these visual methods; the space will be partitioned into regions of specific shapes. Voronoi diagrams were selected as a superior alternative in an attempt to overcome the complexity of the data structures of geographic information systems [40]. Voronoi diagrams have many other benefits that make them a good choice for applying the required partitioning of the space [40]. The most important advantage of the Voronoi diagrams is that it can be computed, using some specific algorithms such as Fortune's algorithm [30], in a reasonable time. Therefore, Voronoi diagrams were chosen by the new approach (PROD) in order to perform the partitioning of the space.

**Voronoi Diagram and Delaunay Tessellation**

The origin of Voronoi diagrams dates back to the 17th century [7]. The mathematicians *Dirichlet* and *Voronoi* had introduced the concept of these diagrams which had been used in variant fields. The underlying idea is having a space *M* and a set *S* of points called *sites p* in *M*; the space is partitioned into regions such that each region *r* contains one of the sites. The region of $p$ consists of all points *x* for which $p$ is the closest site to *x* among all other sites $s \in S$, which is determined by measuring the Euclidean distance. The resulting structure from this partitioning called *Voronoi diagram* or *Dirichlet tessellation*.

The dual structure of Voronoi diagram was defined as the diagram where any two sites, which their regions have a common boundary, are connected. This dual structure was later introduced by Delaunay and it has been denoted *Delaunay tessellation* or *Delaunay triangulation* after him. Delaunay tessellation is obtained by defining that two sites are connected if and only if they lie on a circle that does not contain any other site in its interior.

The resulting Voronoi diagram of 11 sites is illustrated in Figure 2. Given the set $S$ of $n$ sites and $p \in S$; the Voronoi regions are disjoint and each region $VR(p,S)$ that contains the site $p$ is defined as the intersection of $n-1$ open half-planes. More formally, Voronoi region is defined as follow $VR(p,S) = \bigcap_{q\in S, q\neq p} D(p,q)$ such that $D(p,q) = \{x \mid d(p,x) < d(q,x)\}$ is the half-plane containing $p$. Consequently, the Voronoi diagram $V(S)$ is defined as follow:

$$V(S) = \bigcup_{p,q\in S, p\neq q} \overline{VR(p,S)} \bigcap \overline{VR(q,S)}$$

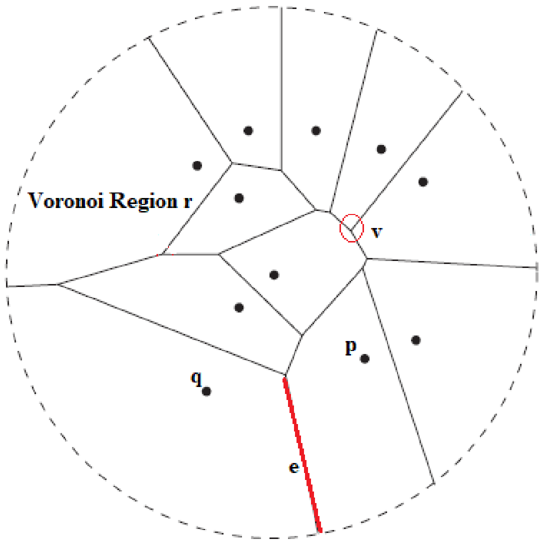


*Figure 2. Voronoi Diagram of 11 Points in the Euclidean Plane [7].*

The Voronoi diagram contains Voronoi edges and vertices. A Voronoi edge is a common boundary of two Voronoi regions. When a Voronoi edge $e$, illustrated as a red line in Figure 2, borders the regions of two sites $p$ and $q$; then this edge $e$ belongs to the bisector of $p$ and $q$, i.e., $e \in B(p, q)$. The endpoints of each Voronoi edge are called the Voronoi vertices. Each vertex $v$, shown as an empty red circle in Figure 2, belongs to the common boundary of at least three Voronoi regions, in other words, each Voronoi vertex is of degree at least three. There are some cases where sites are cocircular or collinear. In the case of cocircular points; the Voronoi vertices would be of a degree higher than three. One special case for the Voronoi diagram is having all the sites to be collinear, this case results in a disconnected Voronoi diagram as it will consist of parallel lines.

Given a Voronoi diagram and an arbitrary point $x$ in the plane; there are two ways in order to know the region to which $x$ belongs. The first method is by drawing a circle $C$ centered at $x$ with radius equal to zero, and then growing this radius gradually until $C$ hits one or more of the sites. If $C$ hits exactly one site $p$; then $x$ belongs to the Voronoi region of $p$. In the case that $C$ hits exactly two sites, $p$ and $q$, then $x$ lies on the Voronoi edge that borders the Voronoi regions of $p$ and $q$. If $C$ hits three or more sites, then $x$ is considered to be a Voronoi vertex that belongs to the common boundary of regions of sites which have been hit. The second possible method is starting by drawing circle around each site with radius equal to zero, and then expanding those circles at the same speed. Depending on sites whose circles will hit $x$ first; the belonging of $x$ is determined.

On the other hand, having $S$ as the set of sites; the Delaunay tessellation $DT(S)$, which is the dual structure of Voronoi diagram, is constructed by connecting via a line segment any two sites $p, q \in S$ which satisfy that there exists a circle $C$ on which $p$ and $q$ lie, and this circle does not contain any other site neither in its boundary nor in its interior. The edges of Delaunay tessellation are called Delaunay edges. One important connection between Voronoi diagram and Delaunay tessellation is that for two sites $p$ and $q$ to be joined by a Delaunay edge in Delaunay tessellation; their regions in Voronoi diagram should be edge-adjacent. If all sites in $S$ are in general positions, i.e., there is no four sites which are cocircular, then the dual of the Voronoi diagram called Delaunay triangulation. More clearly, in the latter case; the dual of the

Voronoi diagram will be a set of triangles where each of the three vertices of each triangle is a site in the Voronoi diagram. Figure 3 shows an example of Voronoi diagram V(S) drawn in solid line and Delaunay tessellation DT(S) drawn in dashed line, such that *S* is the set of sites.

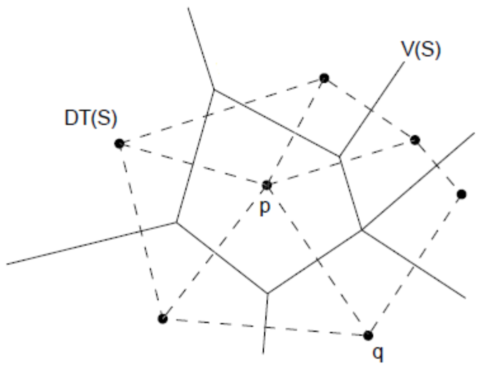


*Figure 3. Voronoi Diagram and Delaunay Tessellation [7].*

## 2.4 Sampling Algorithms

Sampling is a technique of selecting units from a large population called sample [17]. This sample is a subset of the population and it is chosen in order to reduce the complexity of studying the population through studying only this subset. Sampling algorithms are mainly divided into two main classes: probability sampling and non-probability sampling. Using the probability sampling to construct the sample; all units in the population have equal chance of being chosen into the sample. On the other hand, using the non-probability sampling; different probability is attached to each unit of the population and these units do not have equal chance of being part of the sample. Moreover, any kind of sampling can be done either with replacement or without replacement. Sampling with replacement means each unit can be chosen more than once, i.e., the selected unit from the population will not be removed from this population and it can be chosen again in the sample. On the other hand, sampling without replacement means each unit can be chosen only one time, i.e., after selecting a unit from the population it will be removed from this population and cannot be chosen into the sample again.

### 2.4.1 Random Sampling Method Without Replacement

The random sampling method is the easiest and the least complex sampling method [17]. All data points in the data set have equal probability to be selected into the sample. The main idea of this method is that the selection of the sample points is done randomly using either a random number generator or a random number table. More precisely, given *n* data points in the data set and *s* as the sample size; these *n* points are inserted into a list and each point is given a unique number between 1 and *n*. Using for instance the random number generator; *s* numbers will be selected randomly from this list to be included in the sample.

Many formulas can be used in order to calculate the sample size *s* [18]. One of the simplest formulas was provided by *Yamane* [18]. This formula calculates the size of the sample *s* relatively to the number of data points *n* and to an allowable sampling error *e* as depicted in Equation 1. Furthermore, another way for choosing the sample size relatively to the number of data points in the data set is to choose a relative portion of the total number of points. For instance, having *n*=100 data points; it is possible to choose *s*=10% of *n*, i.e., the sample size in this case is equal to 10 data points.

$$s = \frac{n}{1+n(e)^2} \quad (1)$$

### 2.4.2 Systematic Sampling Method Without Replacement

The systematic sampling method is an equal sampling method [17]. More clearly, all data points have equal probabilities to be chosen in the sample. The idea of the systematic sampling is to construct a sample of size *s* from an ordered set of $n$ data points through choosing always the $r^{th}$ point from this ordered set, where *r* is the sampling interval and is calculated using $n$ and *s* as in Equation 2. More precisely, having *n* data points in a two-dimensional plane; the following steps are followed: (1) the $n$ data points are sorted according to their position, (2) the sample size *s* is calculated, (3) the sampling interval *r* is computed, and (4) each $r^{th}$ data point is chosen to be added to the sample until a sample of size *s* is reached. The sample size *s* can be either set to a chosen portion of the data set, e.g. 5 percent of the data points, or it can be computed in a more elaborated way [18].

$$r = \frac{n}{s} \quad (2)$$

Sorting the $n$ data points in the two-dimensional plane is done according to the positions of these data points. Two steps are followed in order to apply the sorting depending on the positioning of the data points in the plane. First, the Euclidean distances between each data point and the point $p_0$ that has the coordinates *x*=0 and *y*=0 are computed. Second, the sorting of these data points is done according to the computed distances. Having the sample size *s* and the *n* data points sorted according to their position; each $r^{th}$ data point is chosen to be included in the sample. Sorting the data points according to their position and always choosing the $r^{th}$ points help in reducing the possibility of choosing two points, which are placed close to each other in the plane, to be included in the sample.

The complexity of computing the Euclidean distance for each of the $n$ data point is $O(n)$. Moreover, using the merge sort algorithm in order to sort the data points according to the computed distances has a complexity of $O(n * logn)$. Finally, in order to choose the $r^{th}$ data point from this sorted list of points; a sequential access over all of the *n* points is required, i.e., the complexity is $O(n)$. Consequently, the complexity of the systematic sampling method without replacement is $O(n * logn)$.

# 3. Related Work

Many previous prediction approaches were introduced in order to help the analysts in the analytical process through providing predictions for input parameters which are required by mining algorithms. However, some of these approaches have some issues. The most common issue is the need for applying the original mining algorithm multiple times in order to produce the prediction. Another issue is the complexity of some of these approaches, which make them not feasible for big data.

In Subsection 3.1, two previous methods are introduced. One concerns the prediction of the two input parameters of the DBSCAN algorithm, and the other is for predicting the input parameter of the OPTICS algorithm. Furthermore, Subsection 3.2 describes some of previous approaches used for predicting the input parameter of the K-Means algorithm.

## 3.1 Previous Prediction Methods for Density-Based Clustering Algorithms

In order to execute DBSCAN, it requires two parameters: *MinPts* which is the minimum number of data points that should exist in the neighborhood of the core point in a cluster, and $\varepsilon$ which determines the neighborhood. Moreover, regarding OPTICS, it should be provided with $\varepsilon$ as an input parameter in order to be executed. Knowing these two parameters for each data set in advance is not trivial. There exist heuristics and best practices which are used to determine *MinPts* and $\varepsilon$ [15, 16]. In this section, some of the previously proposed approaches for predicting the input parameters for DBSCAN and OPTICS are introduced.

### 3.1.1 Determining DBSCAN Parameters

Having the knowledge of $\varepsilon$ and *MinPts* in advance is not easy; therefore, an effective heuristic is used in order to determine general parameters which are used for all clusters [15]. The $\varepsilon$ and *MinPts* of the "thinnest" cluster are determined and chosen as the general parameters. More clearly, the thinnest cluster is the cluster with the lowest possible density, i.e., the density which is not considered noise yet.

The used heuristic for determining the general parameters is based on the following observation. Given a point $p$ from the data set and the distance $d$ from $p$ to its $k^{th}$ nearest neighbor; the $d$-neighborhood of $p$ contains exactly $k + 1$ points assuming that it is unlikely to have multiple points with equal distances. For a given *k*; a function *k-dist* is defined which assigns each data point to the distance from its $k^{th}$ nearest neighbor. After computing the function *k-dist* for each data point; all points are sorted in descending order according to this value and the resulted graph is called the *sorted k-dist graph* which gives a hint about the density distribution in the data set. For instance, by choosing an arbitrary point from this graph, and setting $\varepsilon$ equal to its *k-dist* and *MinPts* equal to the *k*; each other point that has equal or smaller *k-dist* is considered as a core point.

Finding the parameters of the thinnest cluster means finding the threshold point which is the point that has the highest *k-dist* value and still not considered as a noise. Choosing this threshold point from the sorted graph is done by finding the first "valley" in this graph. All points having *k-dist* higher than the *k-dist* of the threshold point are considered as noise, whereas, all other points having lower *k-dist* are assigned to clusters. Figure 4 shows an example of the sorted *k*-

*dist* graph where *k*=4. The first valley in this graph can be easily detected in this graphical representation. All points on the left of this valley are noise, and all points on the right side of the valley are assigned to clusters.

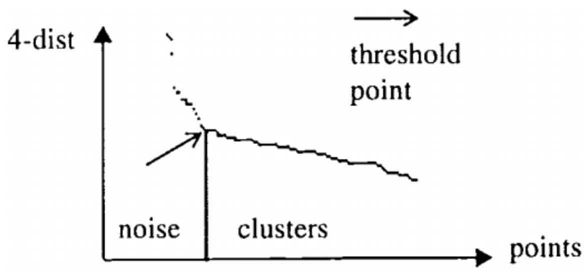


*Figure 4. Sorted 4-dist Graph [15].*

Automatic detection of the first valley in the graph is difficult, however, it is relatively easy to be done by the user which yields to proposing an interactive approach in order to detect this valley. Furthermore, many experiments done by *Ester*, *Kriegel*, *Sander*, and *Xu* show that for two-dimensional data; the *k-dist* graphs of *k*>4 do not differ significantly from the *k-dist* graph of *k*=4 [15]. As a result, the *MinPts* parameter is set to 4 for all two-dimensional data, i.e., *k* is set to 4. In order to determine the needed *k-dist*, i.e., the *ε* parameter; the following approach is followed. First, the system constructs the *4-dist* sorted graph and present it to the user. Then, in case the user has some knowledge about the noise percentage, he provides the system with this knowledge so that the system is able to automatically determine the first valley, otherwise, the system presents the *4-dist* sorted graph to the user in order to detect and choose the threshold point. Finally, the *ε* parameter of DBCSAN is set to the *4-dist* value of the detected threshold point.

### 3.1.2 Determining OPTICS Parameters

Visualizing the cluster-ordering can be done using the reachability-plot [16]. The reachability-plot is insensitive to both parameters *MinPts* and *ε*, which is an important advantage in comparison to other density-based clustering. However, the number of clustering levels which can be seen in the reachability-plot is influenced by the generating distance *ε*, for example choosing *ε* with smaller value prevent showing clusters with lower density. Many experiments conducted by *Ankerst*, *M. Breunig*, *Kriegel*, and *Sander* show that OPTICS generate good results when choosing the value of *MinPts* between 10 and 20 [16].

Regarding the generating-distance *ε*, simple heuristics can be used in order to choose a good value. For instance, assuming that the points are randomly distributed, using the *k-nearest-neighbor distance* the value of *ε* can be analytically computed such that *k* here is equal to *MinPts* [16]. More formally, having *n* points in data space DS, the value of *ε* is equal to the radius *r* of a *d*-dimensional hypersphere *S* containing exactly *k* points, i.e., *MinPts*. The radius of *S* can be computed through Equation 3. The function Γ in Equation 3 denotes the Gamma function, which is an extension of the factorial function with its arguments shifted down by 1.

$$r = \sqrt[d]{\frac{Volume_{DS} * k * \Gamma(\frac{d}{2}+1)}{n * \sqrt{\pi^d}}} \quad (3)$$

## 3.2 Previous Prediction Methods for K-Means Algorithm

K-Means algorithm requires as an input the parameter $k$, which is the number of clusters, in order to be executed. Knowing the right number of clusters for each data set in advance is not trivial. Many heuristics and best practices exist in order to determine number of clusters [19, 20, 23, 24, 25, 26, 27]. Some of these heuristics have different issues. The main problem with some of these already existing approaches is that they require applying the original K-Means algorithm multiple times, which is very time-consuming especially when having huge amount of data. This issue makes these approaches not feasible for big data. In this section, some of the previously proposed approaches for predicting the number of clusters are introduced and the main issues of each of them are mentioned.

### 3.2.1 Enhanced K-Means Clustering Algorithm Using a Heuristic Approach

The proposed algorithm, using a heuristic approach, predicts number of clusters $k$ as well as clusters' centers initialization without performing the actual K-Means algorithm [19]. The main assumption on which this algorithm works is that the clusters are well separated. Given a set P of $n$ points; the algorithm finds $k$ points from $k$ different clusters.

The algorithm starts by choosing an arbitrary point $newPoint \ \in P$ and considers it as a point in a new separated cluster $newCluster$. Depending on the assumption of having well separated clusters; the distance between $newPoint$ and all points belonging to the same cluster $newCluster$ will be less than the distance between $newPoint$ and all points which do not belong to this cluster. Taking this assumption into consideration, the algorithm computes the distances between $newPoint$ and all the other points $P$ and chooses the point which has the maximum distance to be in another new separated cluster. The algorithm repeats those steps to find all of the $k$ points assuring that each point will be from a different cluster.

In a certain iteration, the algorithm will reach a state where $k + 1$ clusters are chosen. In order to identify this state and to reject the last cluster; the algorithm uses a heuristic based on Euclidean distance to declare a stopping condition for the algorithm. The heuristic says that if two objects belong to the same cluster then they both would have similar distances from each point outside this cluster. Consequently, in each iteration the algorithm takes the $newPoint$ and finds the nearest point $nearestPoint$ to it between the previously identified clusters. Then, it computes the distances between $newPoint$ and all other points in $P$ as well as the distances between $nearestPoint$ and all other points in $P$. As a stopping condition for this algorithm, if the computed distances are similar; then $newPoint$ and $nearestPoint$ belongs to the same cluster so the algorithm stops and rejects the last identified point and returns the found $k$ points. The main disadvantage of this algorithm arises from the fact that the stopping condition is based on a heuristic, and therefore, it cannot be proven to work for all cases [19]. Another problem with this approach is that it is based on the assumption of the well separation of clusters, which is not the case in real world.

### 3.2.2 Improved K-Means Algorithm Based on the Clustering Reliability Analysis

The main idea of the Improved K-Means algorithm is to actually perform the original K-Means algorithm for different values of $k$ and then to evaluate, for each $k$, the effectiveness of the resulting clusters [20]. Having $n$ data points to cluster; the different values of $k$ for which the original K-Means algorithm is performed are limited to a range $[k_{min}, k_{max}]$ such that many researchers [21, 22] showed that $k_{max} \leq \sqrt{n}$, and $k_{min}$ is set by the algorithm to be equal to 2, i.e., the range in which $k_{OPT}$ falls is $[2, \sqrt{n}]$. The evaluation of the validity of the resulting clusters is done using specific measures which are based on two concepts: (1) maximizing the inter-clusters distances $B$, i.e., the distances between the clusters, and (2) minimizing the within-cluster distances $W$, i.e., the distances within one cluster. In other words, these measures are computed for each point in the resulting clusters.

More precisely, having a cluster *j* and a point *i* inside *j*; the goal is to (1) maximize $B(j, i)$ which is calculated depending on distances between this point *i* and each other point *p* not belonging to the same cluster *j*, and (2) minimize $W(j, i)$ which is calculated depending on distances between this point *i* and each other point *q* belonging to the same cluster *j*. These two measures are merged into one formula called the Between-Within Proportion (*BWP*), see Equation 4. *BWP* is used as an indicator of the clustering efficiency such that the bigger the value is, the more efficient the resulting clusters are. After computing this indicator for each data point in the resulting clusters; the average *BWP* of the resulting clusters from applying K-Means for *k* is computed using the formula depicted in Equation 5, such that $n_i$ is number of data points in the corresponding cluster. Having the average *BWP* for each of the resulting clusters from applying K-Means for $k \in [2, \sqrt{n}]$; the $k_{OPT}$ is determined through Equation 6. Clearly, the need to perform the original K-Means algorithm multiple times is one major disadvantage which could not be solved by this algorithm.

$$BWP = \frac{B(j,i) - W(j,i)}{B(j,i) + W(j,i)} \quad (4)$$

$$avg_{BWP}(k) = \frac{1}{n} \sum_{j=1}^{k} \sum_{i=1}^{n_i} BWP(j,i) \quad (5)$$

$$k_{opt} = arg_{2 \leq k \leq n} \max\{avg_{BWP}(k)\} \quad (6)$$

### 3.2.3 The Global K-Means Clustering Algorithm

The main goal of the global K-Means algorithm is to improve the process of finding the centroids of the clusters [23]. This algorithm clusters all data points in *k* clusters incrementally. In other words, in order to solve the problem of clustering *n* data points into *k* clusters; all intermediate problems of clustering *n* data points into $1, \dots, k-1$ clusters are solved in sequence. The algorithm starts with *k = 1* cluster for which the optimal solution is known, then for $k = 2$: the cluster centroid obtained from *k = 1* remains at its position and the centroid of the second cluster will be placed at several positions within the data space such that the original K-Means algorithm will be applied multiple times in order to find the optimal position of this centroid. More clearly, in order to solve the problem of clustering *n* data points into *k* clusters; at any stage *M* all previously found $M - 1$ centroids, which had been found by this algorithm, will be remained at their positions, and the remaining new $m^{th}$ centroid will be initially randomly placed and then its optimal solution will be founded by performing the original K-Means algorithm.

In addition to finding the centroids of the clusters, the global K-Means algorithm can be used to find the correct number of clusters. Through applying this algorithm, all intermediate values of *k* will be directly computed. The latter gives a huge advantage in reducing the computational effort needed to find the suitable number of clusters. On the other hand, one disadvantage of this algorithm is the computational complexity which is caused because of applying the original K-Means algorithm several times for each intermediate value of *k*.

### 3.2.4 The Elbow Method

The main idea of this method is to apply the original K-Means algorithm multiple times starting with *k*=2 and then increasing the value until the sum of the squared errors of the resulted clusters approximately stabilizes [24, 25, 26]. More clearly, starting with *k*=2; in each iteration K-Means is applied and the sum of squared errors $SSE$ is calculated, such that smaller value of $SSE$ indicates a good fit of the clustering. Increasing the number of clusters yields to dramatically decreasing the value of $SSE$, i.e., yields to having clusters with better homogeneity. This dramatical decreasing of $SSE$ continues until we reach a stage where adding a new cluster leads to partitioning points that should be in the same cluster, i.e., very small decreasing in the value of $SSE$. This value of $k$, where the $SSE$ starts to decrease slowly, is the desired number of clusters. Identifying this value is done through plotting a function of the value of SSE against the number of clusters such that the desired value of *k* will give an angle in the graph called the elbow; and the best number of clusters is chosen at this point. Figure 5 shows an example of the Elbow method where the red circle is the elbow which represents the best value of *k*. The main problem with this method is that this elbow cannot always be clearly identified either because it may not be visible in the plot or because there might be multiple elbows in that plot.

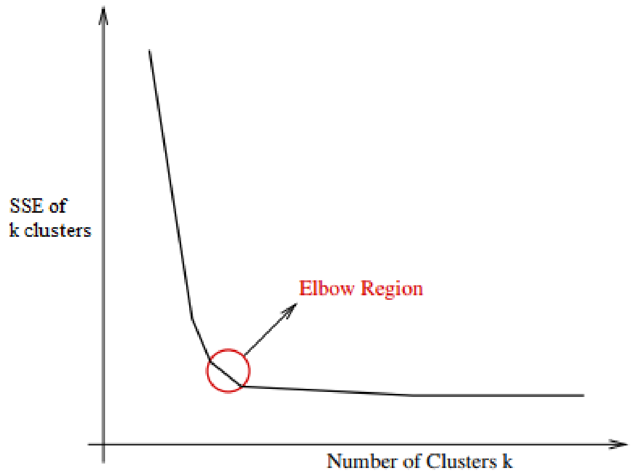


*Figure 5. Elbow Method and Identification of Elbow Point [26].*

### 3.2.5 The Information-Theoretic Approach

The information-theoretic approach is mainly based on determining number of clusters which maximizes the efficiency and minimizes error depending on information-theoretic standards [24]. More specifically, having *n* data points; the original K-Means algorithm is applied for all values of $k$ in the domain $]1, n[$, then the distortion of the resulting clustering is computed, which is a measure of the within-cluster dispersion. The sum of the squared error formula *SSE* is utilized and extended according to the Gaussian distribution model [29] in order to calculate the distance between a point *p* and the centroid of the cluster in which *p* belongs.

More specifically, having *k* clusters, a point *p* belongs to the cluster *C*, and $o_c$ as the centroid of the cluster *C*; the distance between $p$ and $o_c$ is calculated using Equation 7. Here, $\Gamma_c$ is the within covariance matrix, and $y_p$ is the vector representing the point *p* in the dimensional space. Finally, the resulting values will contain *jumps* which indicate the good values for the $k$ such that the largest jump is considered as the best value of *k*, i.e., the best number of clusters to be chosen. The jumps are defined as in Equation 8, where *W* is the extended *SSE* and *M* is the dimensionality of the data. The main disadvantage of this approach is the need of applying the original K-Means algorithm for all values of *k* between $]1, n[$.

$$distance = (y_p - o_c)^T \Gamma_c^{-1} (y_p - o_c) \quad (7)$$

$$JS(K) = W_K^{-\frac{M}{2}} - W_{K-1}^{-\frac{M}{2}} \quad (8)$$

### 3.2.6 The Silhouette Method

The main idea of the Silhouette method is to measure the closeness of data points within the cluster and the separation between the clusters [24]. This measurement is done through calculating the silhouette width value for each data point. The silhouette width value lies in the domain [-1, 1] and it indicates whether this point is in the correct cluster or not. More clearly, a point that has a silhouette width value close to 1 means that this point is contained in the correct cluster, however, a point having a silhouette close to –1 indicates that this point should be contained in the next closest cluster. For each data point *p*; given the average distance *avg* between *p* and all points in its cluster, and the average distances *other-avgs* between *p* and all points in each other cluster; the value of the silhouette width $s(p)$ is calculated using Equation 9, such that $minAvg(p)$ is the minimum average distance among all *other-avgs*.

$$s(p) = \frac{minAvg(p) - avg(p)}{\max(avg(p),\ minAvg(p))} \quad (9)$$

After calculating the silhouette width for each data point in the resulting clusters; the average silhouette width for the whole clusters is computed. The main disadvantage of this method is the necessity of applying the original K-Means algorithm multiple times with different values of *k* and performing large amount of computations in order to predict the best value of *k*. More specifically, predicting the best *k* is done through computing the average silhouette width for each resulted clusters, which requires computing the silhouette for each data point *p*. Performing all of these computations requires a lot of distance calculations. Having the different resulting clusters of applying K-Means with different values of *k*, in addition to the average silhouette width for each result; the largest average silhouette width indicates the best value of *k*, i.e., the best number of clusters.

# 4. Position-Based Prediction Using Voronoi Diagrams

The main goal is to find new prediction approach that is able to solve the problems of the previously existing approaches. More specifically, this new approach, which is able to predict the required input parameters, has three important properties. First, the main property in this new approach is that the prediction is performed without the need of applying the original clustering algorithm multiple times. Second, the new approach is feasible for big data sets, i.e., it can be applied for huge amount of data and perform the prediction within a reasonable time. Third, the last property of this new prediction approach is that it makes the exploration of the solution space much easier for the analysts.

This new approach is called PROD (**P**osition-Based P**r**ediction Using V**o**ronoi **D**iagrams). The basic idea of PROD is to predict the input parameters of the previously described clustering algorithms through drawing Voronoi diagrams for the data set before actually applying the clustering algorithms. An important assumption on which this approach is based, is that density areas are almost separated in order to be distinguished. Moreover, PROD focuses on two-dimensional data, however, all concepts and observations which are made can be transferred to higher dimensions as well. In this section, an overview of PROD is described, followed by a detailed explanation of each step in the workflow of this approach.

## 4.1 Overview

Having a data set consisting of *n* data points; PROD performs the following main steps: (1) *Sampling* of the original data set into a smaller one, (2) *Space Partitioning* which is applied to the space that contains the sampled data set, (3) *Fine Tuning* for the partitioned space in order to better address the characteristics of the mining algorithms, and finally (4) *Predicting* the input parameters which are required by these mining algorithms. Figure 6 illustrates the pipeline which contains these four main steps which are performed by PROD.

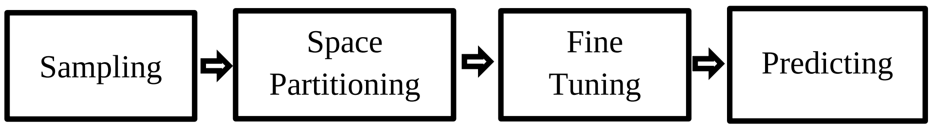


*Figure 6. The Pipeline of PROD.*

More precisely, having a data set consisting of *n* data points in the space as shown in Figure 7a; PROD starts by sampling these *n* data points into smaller set containing *m* points. For instance, Figure 7b illustrates sampling the original data set shown in Figure 7a into smaller set that consists of only 11 data points. PROD continues by considering each of these *m* points as a site and drawing a Voronoi diagram for these *m* sites, which yields to a space partitioned into *m* Voronoi regions. Figure 7c shows the Voronoi diagram drawn for the chosen 11 sites.

After having the Voronoi diagram drawn for the *m* chosen sites; the original *n* data points will be scattered over the *m* Voronoi regions as shown in Figure 7d. The essential idea of PROD lies in performing some fine tuning for the Voronoi diagram where all of the *n* data points are scattered over the *m* regions. The fine tuning consists mainly of two steps. First, inspecting the Voronoi regions in order to decide whether the regions should be merged. Second, applying the decided merging for these regions. For instance, Figure 7e illustrates inspecting the Voronoi regions and deciding that $region_1$ and $region_2$ should be merged into one region. Consequently, in the next step the merging is applied and a final diagram, which can be used to correctly predict the clustering parameters, will be reached. For instance, Figure 7f illustrates the final Voronoi diagram resulted from applying the merging in the previous step.

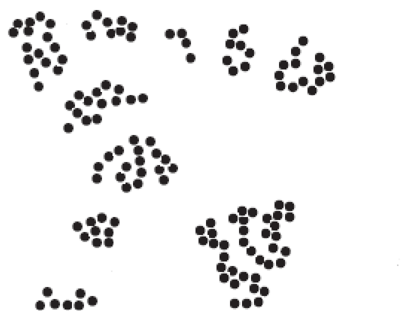


*Figure 7a. n Data Points in the Space*

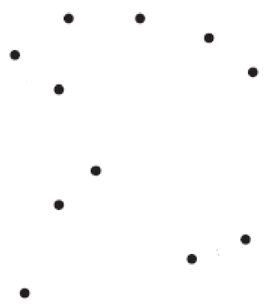


*Figure 7b. m=11 Chosen Data Points into the Sample*

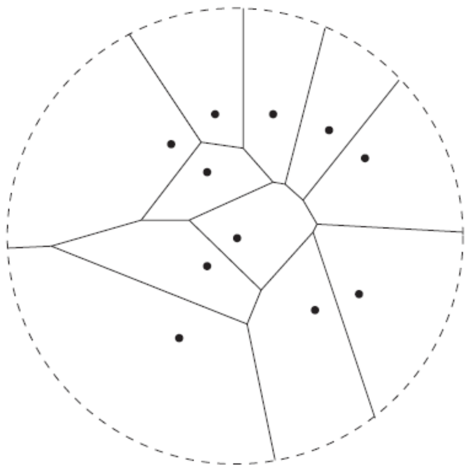


*Figure 7c. Voronoi Diagram for the m = 11 Sites*

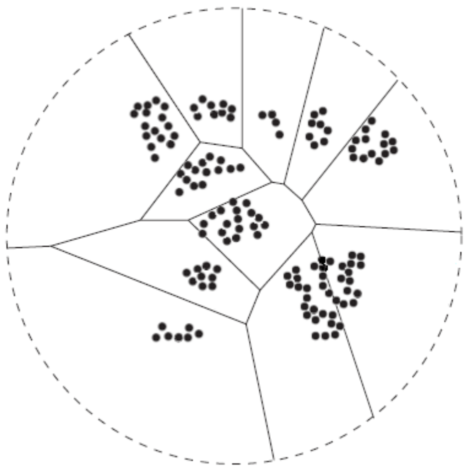


*Figure 7d.Scattring n Data Points Over m Voronoi Regions*

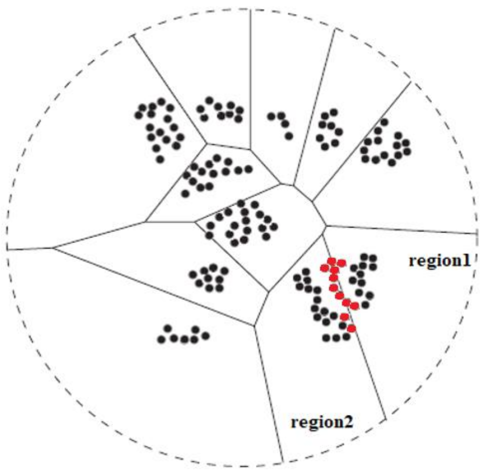


*Figure 7e.Inspecting Voronoi Regions*

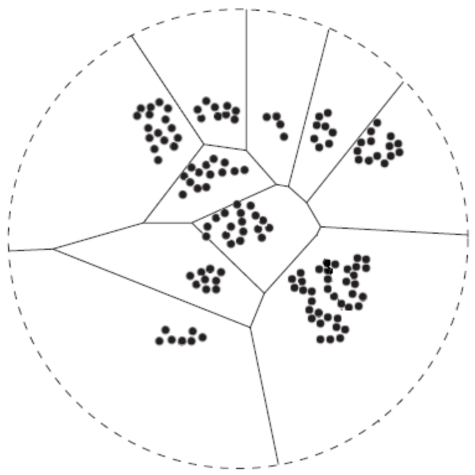


*Figure 7f. Final Voronoi Diagram with Merged Regions*

*Figure 7. Predicting Clustering Parameters Using Voronoi Diagram.*

Using this final diagram, PROD is able to predict input parameters for multiple clustering algorithms. For example, in order to predict the required input parameter for the K-Means algorithm, which is the number of clusters, PROD counts the number of Voronoi regions in this final diagram, and predicts that the number of clusters for this data set is equal to this number of Voronoi regions.

In the following subsections, each of the four briefly described steps of PROD will be described and explained in more details. More specifically, Subsection 4.2 describes how the sampling of the data set is applied and how the resulted sampled data set should be constructed. Subsection 4.3 describes how space partitioning is performed and which approach is used. Subsection 4.4 provides an explanation of the steps followed for performing the fine tuning for the partitioned space. Finally, Subsection 4.5 describes how the prediction of the parameters is done. Followed by an explanation in Subsection 4.6 on how the *outlier* data points can be identified. Finally, the overall complexity of PROD is shown in Subsection 4.7.

## 4.2 Sampling

Many approaches for sampling data sets exist, however, the result of each of these approaches differs. Hence, according to the sample needed; the applied sampling approach should be chosen. In PROD, choosing for instance random sampling to sample the given data sets may lead to sample data set that is not efficient for the next steps of PROD. For instance, taking the data set in Figure 8a, and applying random sampling for this data set, the resulted sampled data set might be as shown in Figure 8b. This sample data set is not efficient for the following steps of PROD as it will lead to a diagram where much more complex fine tuning needed in order to achieve a solid and adequate prediction.

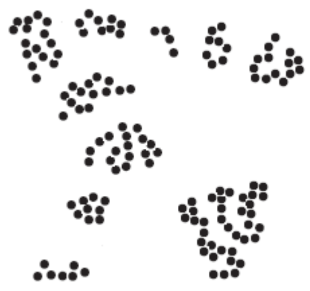


*Figure 8a. Example Data Set*

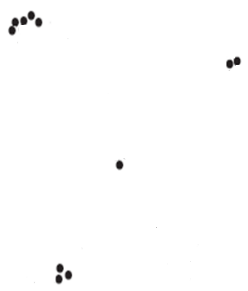


*Figure 8b. Sample Data Set*

*Figure 8. Random Sampling.*

Therefore, it is of huge benefit for the following processing, that the data set is sampled through taking the positioning of the data points in the space into consideration. More specifically, the positioning of data points should be analyzed in order to identify areas where there exist higher gathering of points, i.e., areas with high density of data points. Then, representative points from each identified density area are chosen to be included into the sample data set. Consequently, the sample will be representing the density areas existed in the original data set as it is constructed from the representative points of each area. The identification of these density areas and choosing number of representative points from each area is done using a new sampling approach called *position-based sampling*.

### 4.2.1 Position-Based Sampling Approach

Since the goal is to identify density areas and to choose representative points from each area, it is of importance to know the overall positioning of the points and which points are placed close to each other in order to identify them as a density area and to choose representative points of this area. The main idea of this approach is to construct a sample depending on the positioning of the points in the plane. Given *n* data points, in order to construct a sample of $m < n$ data points; this approach consists of the following steps: (1) sorting all *n* data points according to their closeness to each other, (2) identifying density areas, and (3) choosing representative points from each area.

In order to sort the *n* data points depending on their closeness to each other, the approach uses the *Nearest-Neighbor Search (NNS)*. Applying NNS for a data point *p* finds the nearest point *q* to this *p*. Having *n* data points in the data set, in order to construct a sorted list containing the sorted points: for each point *p* the approach (1) adds *p* to the sorted list, (2) applies NNS to identify the nearest point *q* to *p*, (3) adds *q* to the list, (5) applies NNS for this *q* in order to identify its nearest point from the remaining set of points. These steps are repeated for all data points, i.e., the nearest point for each of the *n* data points will be identified.

Next, after finding the nearest point for each data point in the data set and sorting them accordingly; density areas can be identified. The essential step in order to identify different areas, is to gradually remove data points from the original data set. In order to illustrate the importance of this step, an example of the normal case, which is without removing data points from the data set, is described. For instance, having the data set shown in Figure 9; and staring with the point P0, we search for the nearest point for this P0. By applying nearest-neighbor search, we find that P1 is the nearest point for P0. We continue by searching for the nearest neighbor for P1, and the search returns P2 as the nearest point for this P1. Following the same steps, we search now for the nearest point for P2, and here we find that again P1 is its nearest point. Consequently, we end up looping only between the points P0, P1, and P2 without the ability to move to another set of points.

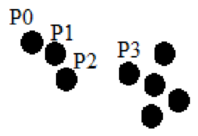


*Figure 9. Sampling Example without Removing Points from Data Set.*

On the other hand, through applying the essential step of gradually removing data points from the data set after identifying them as the nearest points for other points; we do not face the problem of looping between one set of points. To illustrate the result of the enhanced approach that removes points from the data set; let us take the same example data set in Figure 9. We start with P0, we remove P0 from the data set as shown in Figure 10a, and then we apply nearest-neighbor search for this point. The search returns P1 as the nearest point for P0, hence, we remove P1 from the data set as in Figure 10b, and then we search for its nearest neighbor. We find that P2 is the nearest point for P1, therefore, we remove P2 from the data set and next we apply the nearest-neighbor search for P2 on the remaining points in Figure 10c. Now, by applying nearest-neighbor search for P2; the search will return P3 as its nearest point in the remaining data set, i.e., we moved to another area and we did not stick in the same set of points as before.

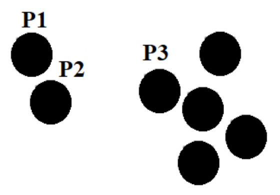


*Figure 10a. Removing P0 from the Data Set*

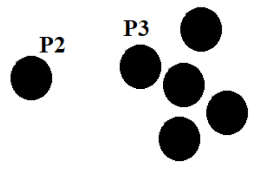


*Figure 10b. Removing P1 from the Data Set*

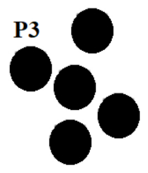


*Figure 10c. Removing P2 from the Data Set*

*Figure 10. Sampling Example with Removing Points from Data Set.*

Recapitulating how the position-based sampling approach works; given $n$ data points and starting from the point P0; the following steps are followed: (1) the point P0 is added to the beginning of an ordered list with a distance equal to 0 as it is the first point, (2) P0 is removed from the data set (3) NNS is applied on the other *n-1* data points in order to find the nearest point P1 to this P0, (4) the Euclidean distance between P0 and its nearest point P1 is computed, (5) P1 is added to the ordered list with its computed distance, (6) P1 is removed from the data set, (7) NNS is applied on the other *n-2* data points to find the nearest point P2 to P1, etc. The position-based sampling repeats all of these steps until the data set is emptied, i.e., until the nearest point for each data point in the data set is found.

Having a sorted list of data points according to their positioning and having the computed distances between each data point $p_i$ and its next point $p_{i+1}$ stored with each point; the position-based sampling approach continues by checking these computed distances sequentially in order to identify density areas. More precisely, the first point $p_0$ in the ordered list is considered as the starting of the first density area, and in case the distance between a point $p_i$ and another point $p_{i+1}$ exceeds a threshold; $p_{i+1}$ considered as the starting of a new separate density area. This threshold is chosen relatively to the overall computed distances, more specifically, the threshold is equal to the average distance. For instance, having the point $p_{i+1}$ identified as the first point with a distance > average-distance; $p_i$ will be considered as the end of the first density area and $p_{i+1}$ will be considered as the start of the second density area. At this stage, where the start and the end of a density area are identified, i.e., $p_0$ and $p_i$ respectively, the representative points will be chosen.

Identified density areas will have varying densities, i.e., varying number of points inside them. Hence, when choosing number of representative points from each of these density areas; it is necessary to take number of points contained in this area into consideration in order to construct a sample with density distribution similar to the density distribution in the original data set but with fewer number of points. In order to achieve the latter and to preserve the density distribution; the approach chooses from each density area number of representative points equal to the logarithm base 10 of the number of all points contained in this area, i.e., $number\ of\ chosen\ points\ from\ the\ area = \log_{10}(number\ of\ points\ in\ this\ area)$. Choosing number of representative points relatively to the number of points contained in each area lead to a better sampling result.

### 4.2.2 Complexity of Position-Based Sampling

Spatial access methods like R* trees are used in order to enhance the runtime of the nearest-neighbor search. Given $n$ data points, the complexity of applying NNS using R* trees for one point is $O(logn)$ and for $n$ points is $O(n * logn)$. Since position-based sampling gradually remove data points from the data set; this number of points *n* on which NNS is applied will be less by 1 in each iteration. More clearly, applying NNS for the first point has a complexity of $O(logn)$, whereas applying NNS for the second point has a complexity of $O(log(n-1))$, etc.

Moreover, in order to identify density areas; the approach goes through the list resulted from applying NNS for all points, and checks the calculated distances of these points. This checking has a complexity of $O(n)$; since the approach goes through all data points once and checks the $n$ computed distances. Consequently, the position-based sampling algorithm has in total a complexity of $O(n * logn)$ in the worst case.

## 4.3 Space Partitioning

Having the sample data set constructed from the $m$ chosen representative points; PROD continues by partitioning the space that contains these *m* data points through computing Voronoi diagram. More clearly, in order to partition the space; PROD considers each of these chosen *m* data points as a site, and then it draws Voronoi diagram for these *m* sites. Through drawing Voronoi diagram; the space will be partitioned into *m* Voronoi regions such that each of these regions contains one of the $m$ sites. Figure 7b on page 20 shows an example of a Voronoi diagram drawn for 11 chosen sites.

Thereafter, by having the Voronoi diagram with the *m* regions; and having each of the *m* chosen points placed into one of these regions as shown in Figure 7c; PROD continues by scattering all of the other *n-m* data points, which are contained in the original data set, over this partitioned space. Having all of the *n* data points scattered over the partitioned space as shown in Figure 7d; each of the *m* data points (the previously chosen sites) is placed into its own region, whereas each of the *n-m* data points has three possibilities in which it is placed: either the data point is contained in exactly one Voronoi region, or it is placed on a Voronoi vertex, or the point belongs to a Voronoi edge.

There exist many algorithms that can be used to draw the Voronoi diagram for a set of *m* data points. For instance, one of the most common algorithms used is the Fortune's algorithm [30]. This algorithm is a Sweep-Line algorithm that generates the Voronoi diagram from a set of *m* data points in the plane. The complexity of the Fortune's algorithm is $O(m * logm)$.

## 4.4 Fine Tuning

The essential idea of PROD lies in (1) *scattering* all of the $n$ data points over the partitioned space, i.e., over $m$ regions, (2) *inspecting* the Voronoi regions and examining the scattering of data points in order to decide which regions should be merged together, and finally (3) *merging* regions when necessary in order to end up with a diagram which can be used to correctly predict the clustering parameters.

Subsection 4.4.1 describes how the scattering of all data points on the partitioned space is done. Followed by Subsection 4.4.2 which introduces the inspection approach applied to the Voronoi regions. Next, Subsection 4.4.3 shows how the necessary merging is performed. Finally, the complexity of the fine tuning step is described in Subsection 4.4.4.

### 4.4.1 Scattering the Data Points

PROD starts by scattering all $n$-$m$ data points on the partitioned space, i.e., on the previously computed Voronoi diagram. While scattering, each data point should be assigned to one of the Voronoi regions. Since each Voronoi region contains data points which are closest to the site of this region among all other sites, therefore, PROD applies, for each of the $n$-$m$ data points, the nearest-neighbor search with respect to the $m$ sites. More clearly, in order to assign each of the $n$-$m$ data points, PROD finds the nearest site to this data point among all of the $m$ sites, and then assigns this data point to the region of this site.

### 4.4.2 Inspection of Voronoi Regions

In general, the inspection approach performs the following main steps: (1) computes the necessary distances, (2) checks these distances in order to decide whether there exist data points in different regions which are close to each other, and (3) decides whether to merge or not accordingly.

The inspection of any two regions is mainly based on checking the closeness of data points contained in one Voronoi region to the other data points contained in the other region. This closeness is measured depending on the distances between these data points and the bordering Voronoi edge of the two corresponding inspected regions. More specifically, these distances are compared to a specific *threshold*, and if they satisfy a defined condition with respect to this threshold, then the two regions will be merged. This *threshold* is set here to be equal to the same threshold used in the position-based sampling step.

For each Voronoi edge, the approach inspects the two neighboring Voronoi regions of this edge in order to decide whether these two regions should be merged. The decision of merging two regions is taken in case of having in both regions data points which are placed close to the edge bordering these two regions. The closeness of these data points should be checked depending on a specific threshold which is chosen relatively to the overall positioning of data points in the space.

More clearly, in order to decide whether two regions should be merged; the approach proceeds in the following steps: (1) sets the threshold distance to be equal to the threshold used in the sampling step, (2) searches for data points in $region_1$ with $distance \leq threshold$ to the bordering edge of these two regions, (3) similarly to $region_1$, searches for data points in $region_2$ with $distance \leq threshold$ to the bordering edge. If and only if the approach finds points in $region_1$ and in $region_2$ satisfying the condition; the decision of merging these two regions will be taken.

Moreover, one case that could occur while inspecting two Voronoi regions is having data points satisfying the condition in only one of these two regions. More clearly, if $region_1$ and $region_2$ were the two inspected regions, and in case the approach found in $region_1$ data points with $distance \leq threshold$ to the bordering edge, but it did not find any data points in $region_2$

with $distance \leq threshold$, hence, no merging will be applied for these two regions since $region_2$ did not satisfy the condition.

Figure 7e on page 20 shows an example where the red data points from both regions: $region_1$and $region_2$ are positioned close to the bordering edge of these two regions. More precisely, each of these red data points has distance to the bordering edge of these two regions $\leq threshold$. Consequently, $region_1$ and $region_2$ will be merged. In case the condition is satisfied only for one of these two regions; then the regions will not be merged.

### 4.4.3 Merging Voronoi Regions

The merging approach is a direct and simple approach, i.e., the Voronoi regions, for which the merging decision was taken, will be merged together directly. More precisely, the merging is simply done through constructing a new Voronoi region that contains all the data points which were contained in each of these merged regions. Finally, the new surrounding edges of this new constructed region will be computed.

Figure 7e on page 20 shows an example of two regions: $region_1$ and $region_2$ for which the merging decision from the inspection step was taken. Therefore, while conducting the merging step; these two regions will be merged into one region. Figure 7f illustrates the final result after applying the merging.

### 4.4.4 Complexity of the Fine Tuning

Scattering $n$ data points and assigning each data point to one Voronoi region is done through applying nearest-neighbor search for this data point. The complexity of applying NNS for a data point is $O(logn)$, and since this should be done for each of the $n$ data points, the complexity of the scattering step is $O(n * logn)$.

Inspecting Voronoi regions and deciding whether to merge regions or not requires following these two main steps: (1) get the threshold used in the sampling step and set the threshold of merging equals to that threshold, and (2) for each edge; inspect the two neighboring regions of this edge. Regarding the second step, for each of the inspected regions; the approach searches for points with distance to the bordering edge less or equal the defined threshold. In order to compute the distance between a point and the bordering edge, the equation of the line representing this edge is required. Therefore, in order to find points which are positioned closely to the bordering edge, the approach proceeds in the following steps: Firstly, it finds the line equation of this bordering edge, Secondly, it computes the distance between points in each region and this bordering edge. More clearly, for each bordering edge and its two neighboring regions $region_1$ and $region_2$; the complexity for inspecting these two regions is $O(m) + O(j)$ such that $m$ is number of points in $region_1$ and $j$ is number of points in $region_2$. Since the approach performs these steps for each bordering edge; and having, for example, $m$ as the largest number of points in a region; the total complexity of the inspection process is $O(e * m)$ where $e$ is the number of all bordering edges.

Finally, Merging two regions is done through constructing a new region and adding into this region all data points from $region_1$ as well as all data points from $region_2$. For each merged regions, the complexity is only $O(1)$. Therefore, having $r$ merged regions; the complexity of the merging step is $O(r)$. Consequently, the complexity of the overall fine tuning step is $O(e * m)$.

## 4.5 Predicting Clustering Parameters

Having the final Voronoi diagram in which some regions are merged; PROD uses this diagram in order to predict the input parameters for the three clustering algorithms: K-Means, DBSCAN, and OPTICS. PROD considers each of the regions in the final Voronoi diagram as a cluster, and hence, it performs the prediction of the clustering parameters accordingly.

### 4.5.1 Predicting Input Parameter for K-Means Algorithm

For the K-Means algorithm, the required parameter is *k* which is the number of clusters. This *k* is chosen as the number of the regions in the final Voronoi diagram. Moreover, providing K-Means algorithm with a good initialization for the clusters' centroids will improve the performance of the algorithm by decreasing the overall runtime. Since PROD considers each of the regions in the final Voronoi diagram as a cluster; the centroids of these regions will be computed and will be provided by PROD as prediction for the initial centroids for the K-Means algorithm. However, before calculating the number of regions and computing the centroids of each region; PROD starts by identifying the outlier data points which should not be taken into consideration when performing the prediction of number of clusters and the centroids of these clusters.

### 4.5.2 Predicting Input Parameters for Density-Based Clustering Algorithms

PROD continues by predicting parameters for the two density-based clustering algorithms: OPTICS and DBSCAN. More specifically, OPTICS algorithm needs *MinPts* as a parameter. PROD finds the region that contains the minimum number of points and sets *MintPts* equal to that number. Furthermore, DBSCAN algorithm requires, in addition to *MinPts*, the parameter *ε*. Using the final Voronoi diagram, PROD finds the largest distance between a point and its site and sets *ε* equal to this distance. However, it is also necessary here for PROD to identify the outlier data points in order not to take them into consideration when performing the prediction of the *MinPts* and *ε*.

### 4.5.3 Complexity of the Prediction

Predicting the number of clusters is done directly since this information is provided easily, i.e., the complexity is $O(1)$. Moreover, in order to provide the initial centroids; the mean point of each region should be computed. Having *r* regions in the final Voronoi diagram; the complexity of computing the mean point for all regions is $O(r)$.

On the other hand, in order to predict the *MinPts* parameter for DBSCAN and OPTICS; the approach needs to find the minimum number of points in a region. This information is directly provided for each region; and since the approach needs to check this information for all regions in order to find the smallest number; the complexity is $O(r)$.

Furthermore, in order to predict the remaining parameter which is *ε*; the approach needs to find the point with the largest distance to its site. The distance between each point and its site is computed previously and stored with each point; hence, the approach needs only to go through all points and checks their computed distances in order to find the largest one. Having *n* data points in the data set; the complexity of finding the largest distance is $O(n)$. In total, the prediction process has linear complexity.

## 4.6 Identifying Outliers

Having the final Voronoi diagram; PROD can identify outliers from this final diagram. Two different approaches for this identification of the outliers were evaluated. The first approach inspects the final Voronoi diagram, and whenever it finds a region with only one point lying inside; it considers this point as an outlier. However, we observe the pitfall that regions with just a few points may also be considered as outliers. Hence, it is of importance to regard the regions in relation to each other. In order to illustrate the importance of taking this information into consideration; let us take an example of the final Voronoi diagram for a data set as shown in Figure 11. In this example we have the two red regions; $region_1$ with approximately 200 points inside it, and $region_2$ with only 2 points lying inside. Following the described approach for identifying outliers; no outliers detected in these two red regions, i.e., each of these two red regions will be considered as a cluster, which is obviously wrong consideration to be made.

On the other hand, the second evaluated approach for identifying outliers takes into consideration the number of points lying in each region in the final Voronoi diagram. More specifically, PROD searches for the region with maximum number of points inside. After having the maximum number of points inside a region; PROD says that if a region has number of points less than ten percent of this maximum number of points; then points inside this region are considered as outliers.

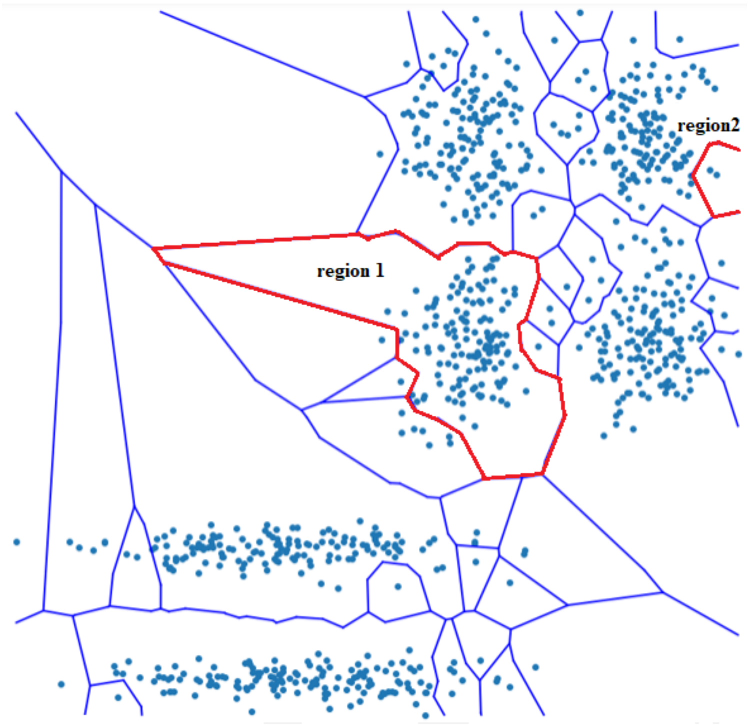


*Figure 11. Example on Identifying Outliers in the Final Voronoi Diagram*

For instance, taking the same example in Figure 11 and assuming that $region_1$ has the maximum number of points among all the other regions; i.e., maximum-number = 200, then any region that has number of points less than 10% of this maximum-number, i.e., less than 20 points, will be considered as outliers. Consequently, by following this approach; points inside $region_2$ will be identified as outliers.

## 4.7 The Overall Complexity of PROD

Knowing the complexity of PROD requires studying the complexity of each of the main steps followed by this approach. Given $n$ data points in the data set, $e$ bordering edges in the first original Voronoi diagram, *r* Voronoi regions in the final resulted Voronoi diagram, and $m$ as the maximum number of points in a region; the complexity is computed as follow. First, the complexity of the position-based sampling, as described in Subsection 4.2, is $O(n * logn)$. Second, drawing the Voronoi diagram using some algorithms, like the Sweepline algorithm, can be achieved in a complexity of $O(n * logn)$. Third, the inspection and merging procedure, as described in Subsection 4.4, has a complexity of $O(e * m)$. Furthermore, performing the prediction of the clustering parameters $k$, *MinPts*, $\varepsilon$ has a linear complexity as described in Subsection 4.5. Finally, identifying the outliers requires checking number of points in all regions, i.e., it has a complexity of $O(r)$. Hence, the overall complexity of PROD is $O(e * m)$.

# 5. Implementation

In order to do experiments and to evaluate the new described PROD approach; it was implemented in Java version '9.0.1'. The result of each step in PROD was drawn and visualized using Python version '3.6.4'. This visualization was implemented using iPython Jupyter Notebook. All experiments have been run on a notebook with Intel® Core™ i7-4500U CPU @ 1.80GHz 2.40 GHz, 8 GB RAM, and runs on Windows 10 Education.

In Subsection 5.1, the class diagram is introduced and the benefits of this class structure are indicated. Moreover, Subsection 5.2 describes the used libraries, which provide the implementation of some steps of PROD. Followed by Subsection 5.3 which briefly describes the visualization that is implemented using Python. Finally, Subsection 5.4 provides a brief description of the used data sets in all experiments.

## 5.1 Class Diagram

The implementation of PROD is divided into six main packages. Each step of PROD is implemented in a separate package. More specifically, the Nearest-Neighbor Search is implemented in a package called *kdtnearest*. Furthermore, each of the: sampling, space partitioning, fine tuning, and the final prediction is implemented in the separate packages: *sampling*, *spacePartitioning*, *fineTuning*, and *prediction* respectively. The final sixth package, which is called *com.mavenproject.predictingparams*, is the main package which imports all the other five packages in order to control the workflow of PROD. Figure 12 illustrates the package diagram as described above.

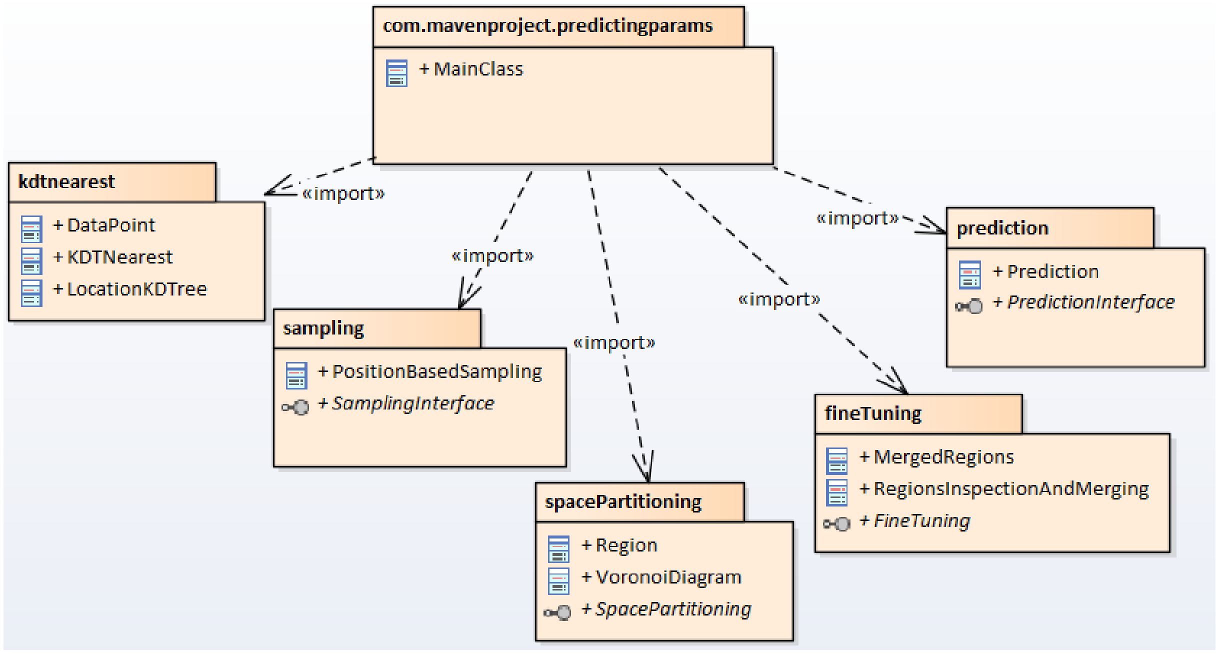


*Figure 12. Package Diagram*

Each package contains the corresponding Java interfaces and classes which provide the functionalities of the respective package. The subsequent subsections introduce the class diagram of each of these six packages. The main package *com.mavenproject.predictingparams* is described in the final Subsection 5.1.6, and the benefits of using this package design with the corresponding classes designs are indicated.

### 5.1.1 The *kdtnearest* Package

The package *kdtnearest*, shown in Figure 13, contains the implementation of finding the nearest-neighbor for each data point in the data set. More precisely, three classes are contained in this package: the class *KDTNearest*, the class *LocationKDTree*, and the class *DataPoint*. The class *DataPoint* represent each data point in the data set. Each data point is identified by its *x* coordinate and *y* coordinate. Furthermore, each data point has a specific distance, called *dist*, which is used here to represent the distance from its nearest neighbor that is calculated by the function *euclideanDistance()*. In later steps, using the same function *euclideanDistance()*, the *dist* will be representing the distance between this data point and the site of the Voronoi region to which this data point belongs.

The class *LocationKDTree* is the class provided by an open-source implementation called Code of Thrones [45]. This class provides the functionality of finding the nearest neighbor for a given data point through applying the Nearest-Neighbor Search on the set of all data points. This functionality is provided through calling the function *findNearest()* that takes the x-coordinate and y-coordinate of the data point. Furthermore, the function *kdTreeNearestNeighbor()*, contained in the class *KDTNearest*, calls the *findNearest()* function for each data point contained in the data set. Since data points are removed gradually from the data set as described in Subsection 4.2.1, hence, the function *removePointFromList()* removes a data point from the list of all data points.

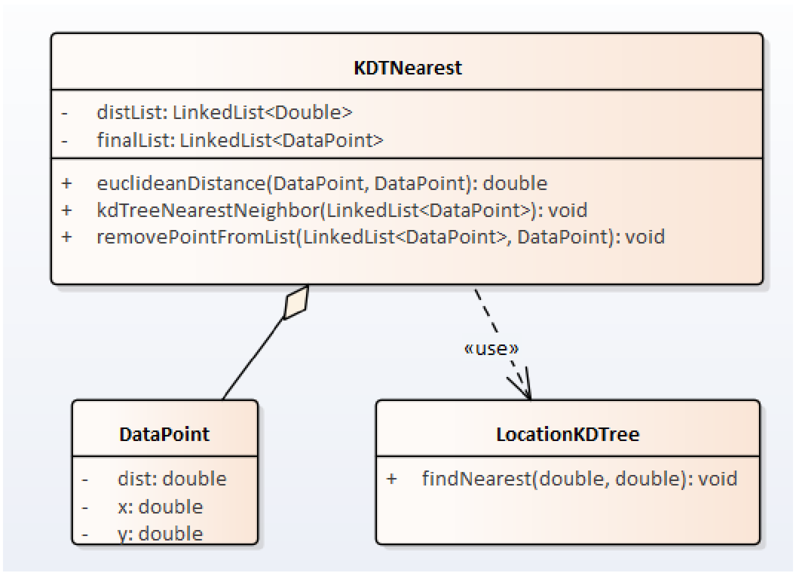

KDTNearest
- distList: LinkedList<Double>
- finalList: LinkedList<DataPoint>
+ euclideanDistance(DataPoint, DataPoint): double
+ kdTreeNearestNeighbor(LinkedList<DataPoint>): void
+ removePointFromList(LinkedList<DataPoint>, DataPoint): void
«use»
DataPoint
- dist: double
- x: double
- y: double
LocationKDTree
+ findNearest(double, double): void

*Figure 13. Class Diagram of the kdtnearest Package*

### 5.1.2 The *sampling* Package

The *sampling* package is responsible for sampling the data set. More precisely, this package contains one interface called *SamplingInterface* which provides the main function needed for performing the sampling. This function is called *sampling()* and it provides the actual sampling functionality.

Furthermore, the *sampling* package contains the class *PositionBasedSampling* that implements the above described interface with respect to the position-based sampling used by PROD. More precisely, this class contains the function *thresholdCalc()* which is for calculating the threshold used for identifying density areas. More specifically, this function computes the average distance of all calculated distances of data points and then sets the threshold, i.e., *samplingThreshold*, to be equal to this average distance. Moreover, the implemented function *sampling()* performs the position-based sampling as described in Subsection 4.2.1 through identifying density areas and storing them in the list *densityAreas*, then choosing specific number of data points from each identified density area, and finally adding the chosen points to the list *chosenPoints*. Figure 14 illustrates the class diagram of the *sampling* package.

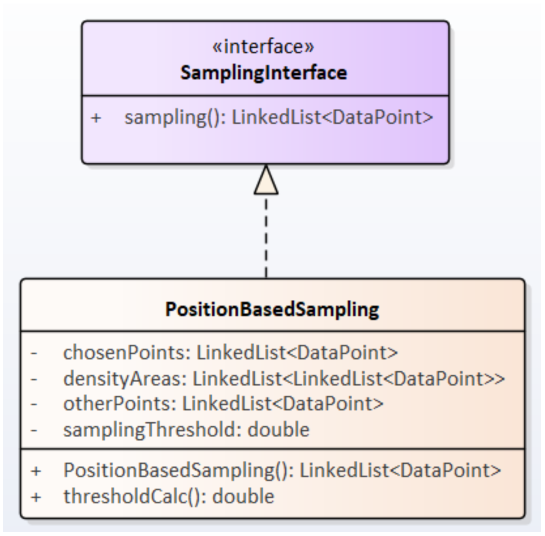

«interface»
SamplingInterface
+ sampling(): LinkedList<DataPoint>
PositionBasedSampling
- chosenPoints: LinkedList<DataPoint>
- densityAreas: LinkedList<LinkedList<DataPoint>>
- otherPoints: LinkedList<DataPoint>
- samplingThreshold: double
+ PositionBasedSampling(): LinkedList<DataPoint>
+ thresholdCalc(): double

*Figure 14. Class Diagram of the sampling Package*

### 5.1.3 The *spacePartitioning* Package

The *spacePartitioning* package, as illustrated in Figure 15, contains one interface called *SpacePartitioning* which provides the main function needed for partitioning the space. This function called *spacePartitioning()* and it provides the actual functionality.

Furthermore, this package contains two classes: the class *VoronoiDiagram* and the class *Region*. The class *Region* defines the attributes of each Voronoi region. More specifically, each Voronoi region contains: (1) a list of data points, called *points*, contained in this region, (2) a list of Voronoi edges, called *polyEdges*, defining this region, (3) the *site* of this region, and (4) an index identifying the region, called *siteIdx*. The class *Region* contains a function called *removeDuplicates()* which is used for removing any duplicated data points or any duplicated edges, through keeping only one instance of the duplicated objects.

Furthermore, the class *VoronoiDiagram* implements the interface *SpacePartitioning*, such that it provides the functionality of partitioning the space by computing Voronoi diagrams. The computation of Voronoi diagram is done through calling the corresponding functionality from a library called Simple Voronoi library [35]. This functionality is provided by the function *computeVoronoi()*, which takes as input: an array of all x-coordinates of all data points, an array of all y-coordinates of all data points, the minimum x-coordinate of a data point among all data points, the maximum x-coordinate of a data point among all data points, and similarly, the minimum y-coordinate and the maximum y-coordinate. Using this input, the library computes the Voronoi diagram and returns a list of all computed Voronoi edges. These computed Voronoi edges are stored in the list *allEdges*.

The next step performed by *VoronoiDiagram* class is constructing the Voronoi regions from the computed Voronoi edges. This is done by the function *constructRegion()* that uses the list *allEdges*, which contains the Voronoi diagram, and constructs a list of all Voronoi regions contained in the Voronoi diagram. The constructed Voronoi regions are stored in a list called *regions*. Moreover, the function *writeEdgesToCSV()* is responsible for writing the computed Voronoi edges *allEdges* into a CSV file in order to visualize them later using Python.

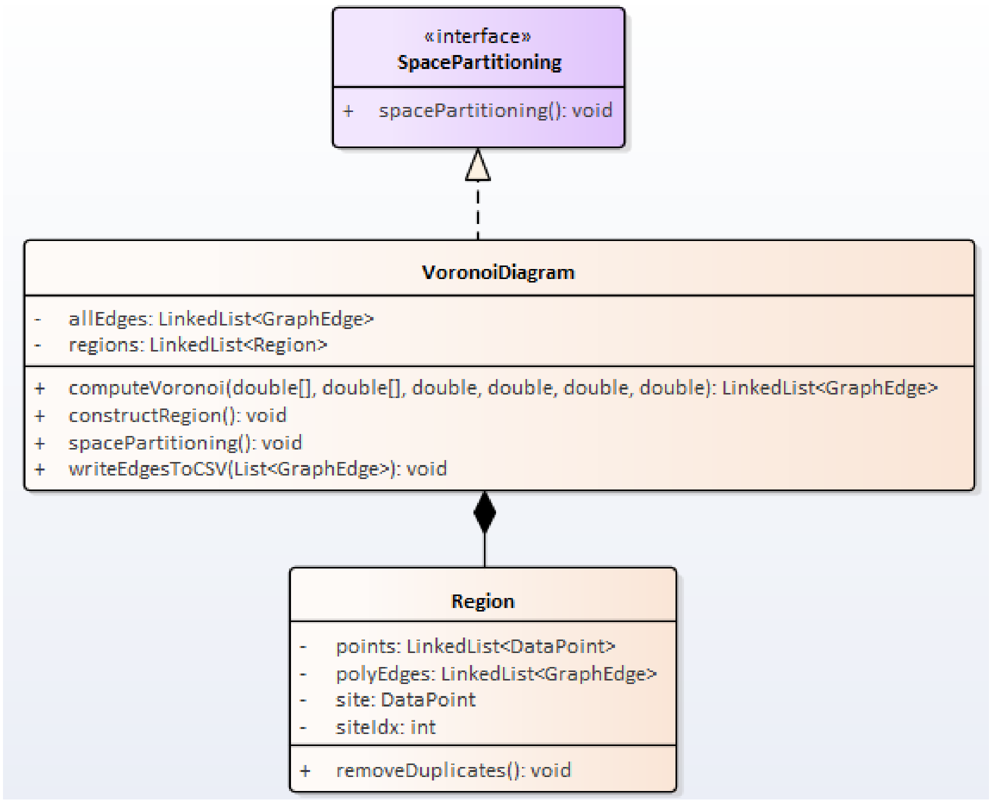

«interface»
SpacePartitioning
+ spacePartitioning(): void
VoronoiDiagram
- allEdges: LinkedList<GraphEdge>
- regions: LinkedList<Region>
+ computeVoronoi(double[], double[], double, double, double, double): LinkedList<GraphEdge>
+ constructRegion(): void
+ spacePartitioning(): void
+ writeEdgesToCSV(List<GraphEdge>): void
Region
- points: LinkedList<DataPoint>
- polyEdges: LinkedList<GraphEdge>
- site: DataPoint
- siteIdx: int
+ removeDuplicates(): void

*Figure 15. Class Diagram of the voronoi Package*

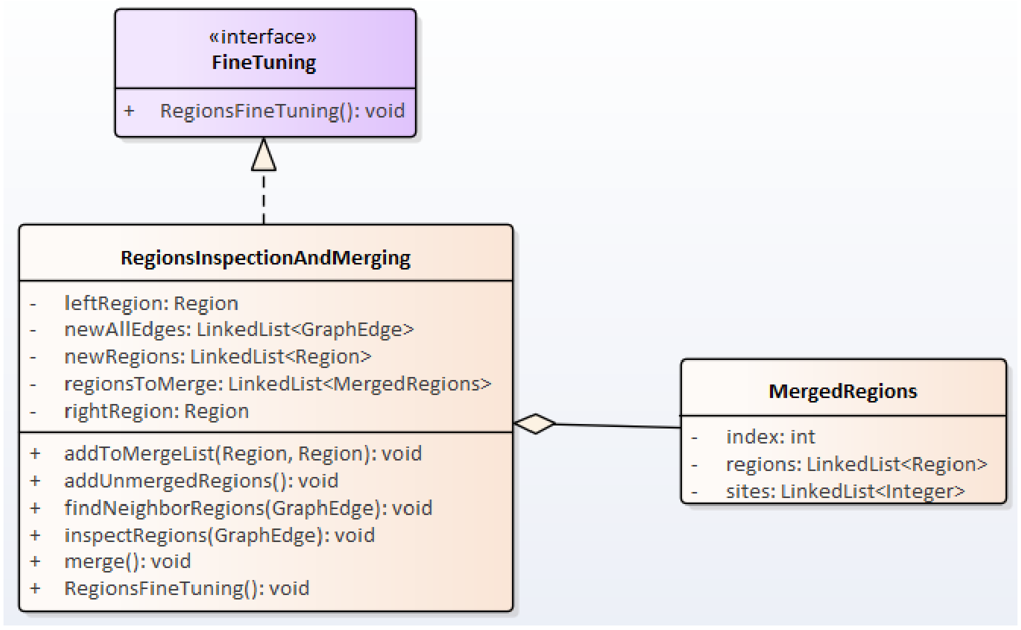
### 5.1.4 The *fineTuning* Package

The *fineTuning* package, as shown in Figure 16, contains one interface called *FineTuning* which provides the main function needed for fine tuning the partitioned space. This function called *RegionsFineTuning()* and it provides the actual functionality.

Furthermore, this package contains two classes: the class *RegionsInspectionAndMerging* that implements the above described interface with respect to the inspection and merging method used by PROD, and the class *MergedRegions*. The latter class, i.e., *MergedRegions* class, represents the identified Voronoi regions which should be merged into one region. Each identified set of regions, that should be merged, are stored together in a list of *regions*. The sites' indexes of those merged regions are also stored in a list of Integers called *sites*. Each list of regions, which should be merged together, is identified using the *index* attribute.

The class *RegionsInspectionAndMerging* provides the functionalities needed to perform the inspection and the merging of Voronoi regions. More specifically, for each Voronoi edge, the function *RegionsFineTuning()* performs the following: First, it finds the neighboring Voronoi regions, i.e., *leftRegion* and *rightRegion*, through calling the function *findNeighboringRegions()*. Second, it inspects these two neighboring regions in order to decide whether they should be merged or not via calling the function *inspectRegions()*. Third, in case the decision of merging these two regions is taken, it calls the function *addToMergeList()* which adds these regions to an element of the list *regionsToMerge*. Fourth, after finishing the inspection of all neighboring regions for each Voronoi edge, the function *RegionsFineTuning()* continues by calling the function *merge()* which performs the actual merging for each element of the list *regionsToMerge*, such that each element of this list is a *MergedRegions* which contains the regions that should be merged together.

After merging each *MergedRegions*; the new resulted region will be added to the list of all new regions, which is called *newRegions*, with a new assigned index. Finally, some of the Voronoi regions will not be merged with any other regions, therefore, these unmerged regions should also be added to the new list of regions. The addition of these unmerged Voronoi regions is done through calling the function *addUnmergedRegions()*.


«interface»
FineTuning
+ RegionsFineTuning(): void
RegionsInspectionAndMerging
- leftRegion: Region
- newAllEdges: LinkedList<GraphEdge>
- newRegions: LinkedList<Region>
- regionsToMerge: LinkedList<MergedRegions>
- rightRegion: Region
+ addToMergeList(Region, Region): void
+ addUnmergedRegions(): void
+ findNeighborRegions(GraphEdge): void
+ inspectRegions(GraphEdge): void
+ merge(): void
+ RegionsFineTuning(): void
MergedRegions
- index: int
- regions: LinkedList<Region>
- sites: LinkedList<Integer>
*Figure 16. Class Diagram of the fineTuning Package*

### 5.1.5 The *prediction* Package

The *prediction* package, as illustrated in Figure 17, contains one interface called *PredictionInterface* which provides the main function needed for predicting the parameters. This function called *predictParams()* and it provides the actual functionality.

Furthermore, this package contains only one class called *Prediction*. This class implements the above described interface and is responsible for performing the prediction of the parameters using the resulted Voronoi diagram. The function *predictParams()* controls the flow of the prediction through performing the following steps. First, it calls the function *findOutlierRegion()* which identifies the regions that contains only outlier data points. This function performs this identification through finding a threshold called *removedNumOfPoints* which represents the number of data points contained in a region and considered as outliers. Second, the function *predictParams()* continues by choosing all other actual regions, i.e., all regions that contain number of data points above the threshold *removedNumOfPoints*. Third, it performs the prediction of the *k* according to the chosen actual regions, which are considered as clusters. More clearly, depending on these chosen actual regions, it calls the function *calcClustersCentroids()* which calculates the centroid of each actual region, i.e., each cluster, and then it predicts the input parameters *k*. Finally, the function *predictParams()* calls the other function *predictDensityBasedParams()* which is dedicated to perform the prediction of *MinPts*, and *epsilon*.

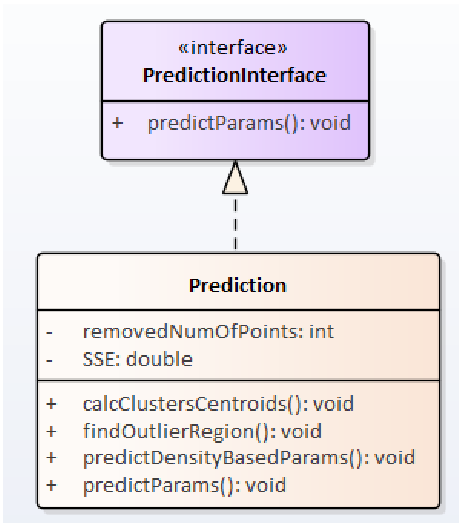

«interface»
PredictionInterface
+ predictParams(): void
Prediction
- removedNumOfPoints: int
- SSE: double
+ calcClustersCentroids(): void
+ findOutlierRegion(): void
+ predictDensityBasedParams(): void
+ predictParams(): void

*Figure 17. Class Diagram of the prediction Package*

### 5.1.6 The *com.mavenproject.predictingparams* Package

The *com.mavenproject.predictingparams* package, as shown in Figure 18, consists of only one runnable class called *MainClass*. This class controls the whole workflow of all steps of PROD and it has the *main()* function. In order to control the workflow of PROD correctly, the main function performs the following steps: First, it enables the user to enter the name of the file containing the data set for which PROD will be applied. Second, it calls the function *readDataPoints()* that reads the data set contained in the specified file, and adds all these data points to a list of *DataPoint*. Third, it continues by calling the functions corresponding to applying each step of PROD.

More specifically, the main function continues by calling the following functions: (1) the function *kdTreeNearestNeighbor()* contained in the package *kdtnearest*, which performs the first step in PROD, (2) the function *sampling()* contained in the package *sampling*, in order to sample the data set, (3) the function *spacePartitioning()* contained in the package *spacePartitioning*, which computes the Voronoi diagram for the sampled data set, (4) the function *RegionsFineTuning()* contained in the *fineTuning* package, which is responsible for inspecting and merging the Voronoi regions, and finally (5) the function *predicitParams()*, contained in the package *prediction*, in order to produce the prediction of the parameters.

Furthermore, the function *writeToCSV()* writes the result of each step performed by PROD in a separate CSV file in order to be visualized later. For instance, after performing the sampling; this function writes the *chosenPoints* into a CSV file in order to visualize the result of the position-based sampling. Finally, as a last step, the main function calls the *writePredictions()* function. This function writes the produced prediction of the input parameter *k*, *MinPts*, and *epsilon* (ε) to a CSV file.

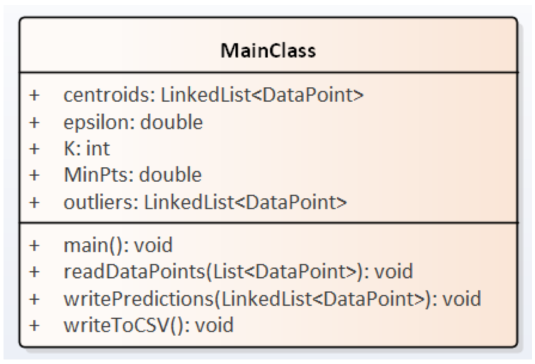

MainClass
+ centroids: LinkedList<DataPoint>
+ epsilon: double
+ K: int
+ MinPts: double
+ outliers: LinkedList<DataPoint>
+ main(): void
+ readDataPoints(List<DataPoint>): void
+ writePredictions(LinkedList<DataPoint>): void
+ writeToCSV(): void

*Figure 18. Class Diagram of the com.mavenproject.predictingparams Package*

Obviously, the design that is shown in the package diagram in Figure 12 on page 30 and described in detail in the previous subsections, has many benefits and advantages. One of the main advantages of this design is its maintainability. More clearly, in case one of the steps performed by PROD has to be changed or modified, this lead to only local changes in the corresponding class contained in the respective package without affecting any other package. Furthermore, the simplicity of the design is another important advantage. This simplicity comes from the fact that the implementation of each step of PROD is encapsulated in an interface, i.e., each interface is responsible for performing only one step in PROD.

Another important advantage is the scalability and extensibility of the design. More precisely, using other space partitioning approaches, e.g., Delaunay tessellation, only requires adding new implementation for the *spacePartitioning* interface, i.e., adding only new class in the package *spacePartitioning* such that this class implements the respective interface with respect to Delaunay tessellation. Similarly, in case of the need for using another sampling approach or another fine tuning approach, require adding new implementation for the corresponding interface in the respective package. Additionally, these modifications require changing only the class initiation in the *MainClass*, i.e., require changing only one line of code. For instance, adding new class providing new implementation for the *SamplingInterface* requires initiating in *MainClass* an object from this new added class instead of the object from the *PositionBasedSampling* class. The addition of a new class is done easily due to the simplicity of the design.

## 5.2 Libraries

PROD uses some existed libraries as a basis for building the following processing steps. More specifically, in order to perform the sampling step, it is necessary to implement or to use an open source implementation for the Nearest-Neighbor Search. Moreover, having the sampled data set; PROD continues by drawing Voronoi diagram for the sampled data set. Therefore, a library for computing the Voronoi diagram for a set of data points will be used.

Furthermore, after PROD is finished and having, for each data set, the prediction of the input parameters; then each of the K-Means, DBSCAN, and OPTICS algorithms should be applied with the respective predicted input parameters. Therefore, a library providing an implementation for each of these three clustering algorithms is used. Finally, three of the six previous prediction approaches, which are described in Subsection 3.2, were implemented in order to compare PROD to each of them. These three chosen approaches are: Improved K-Means, Silhouette Method, and Elbow Method.

Subsection 5.2.1 briefly describes the open source implementation used for the Nearest-Neighbor Search. Furthermore, in Subsection 5.2.2, the library used for computing the Voronoi diagram is introduced. Followed by describing the library which provides the mining algorithms in Subsection 5.2.3. Finally, the implementation of the three previous prediction approaches is described in Subsection 5.2.4.

### 5.2.1 Open Source Implementation of Nearest-Neighbor Search

Many open source implementations for NNS existed. Some of these implementations used a special data structure for enhancing the performance of finding the nearest neighbor for a data point. One of the open source implementations, which is used in the sampling step in PROD, called *Code of Thrones* [45]. This open source implementation uses KD-Tree as a data structure for storing the data points. Using this data structure enhances the performance of the nearest neighbor search, hence, this makes this open source implementation a good candidate to be used in PROD.

### 5.2.2 Voronoi Diagram Library

There are many algorithms for computing the Voronoi diagram for a set of data points. The most common used algorithm is the Fortune's Voronoi algorithm [30]. Therefore, this algorithm is chosen for computing the Voronoi diagram for the sampled data set in PROD. Many Java libraries for computing the Voronoi diagram were found and each one of them was tested in PROD [51, 52, 53, 54, 55]. However, these libraries produce unsatisfying results. More clearly, the subsequent steps performed by PROD require having defined Voronoi regions as a result of the space partitioning steps, however, none of these libraries produced this result as expected by PROD. Furthermore, the source codes of these libraries could not be modified accordingly because they lack the necessary documentations to understand the code and change it. On the other hand, a project called *Simple Voronoi* [34] provides Java code for computing the Voronoi diagram using the Fortune's algorithm. This project also did not produce the expected result, i.e., did not produce the defined Voronoi regions, whereas, it only computes the Voronoi edges. However, constructing the Voronoi regions from the computed Voronoi edges can be performed directly. Therefore, this project was chosen to be used by PROD in order to perform the space partitioning step.

The Simple Voronoi project is written in Java and is hosted on Sourceforge [35]. This library provides a function called *computeVoronoi* which takes the list of data points as an input, computes the Voronoi diagram for these data points, and finally provides all Voronoi edges of the computed diagram. The main issue is that in further steps performed by PROD; it is necessary to have the Voronoi regions defined in order to be able to decide which data points contained in which regions. Therefore, as the result of the library was only the Voronoi edges, a construction for the Voronoi regions was implemented by PROD in order to enhance the result of the Simple Voronoi library. The code provided by the library was not modified, whereas, the construction of the regions was implemented inside PROD.

This fact caused somehow the execution runtime of PROD to be increased. This added complexity is not related to PROD, whereas, it is caused by the incomplete result provided by the Simple Voronoi library. However, more elaborated libraries may overcome this pitfall.

### 5.2.3 Mining Algorithms Library

Each of the three clustering algorithms K-Means, DBSCAN, and OPTICS should be applied with the respective predicted input parameters. Therefore, a library providing an implementation for each of these three algorithms is used. Many machine learning libraries exist, such that each of these libraries provide specific number of mining algorithms [56, 57, 58]. One important library in the machine learning area, which is specifically specialized in pattern mining, called *SPMF* [36]. SPMF is an open source data mining library written in Java code and provides implementations of 150 different mining algorithms of various mining techniques. This library is one of the simplest libraries used in the machine learning area, and it has a satisfying documentation and large number of test cases for testing and applying each of the mining algorithms. More clearly, the library provides number of data sets for each mining algorithm in order to test and apply this algorithm for the respective data sets. Moreover, the library also provides the ability to modify the source code in order to apply these mining algorithms for other data sets. All of these make this library a good candidate to be used by PROD. SPMF version v2.30c is the current used version of this library.

The implementation of each of the K-Means, DBSCAN, and OPTICS algorithms are also provided by SPMF. In order to apply each of these three algorithms, some changes were made to the code of each of the three implementations. Subsection 5.2.3.1 describes the modified code for both DBSCAN and OPTICS. Similarly, in Subsection 5.2.3.2, the modified code for the K-Means algorithm is described. Furthermore, some additional code that encapsulates these implementations was necessary to be added to the SPMF library. Subsection 5.2.3.3 describes this additional code.

#### *5.2.3.1 Modified Code in DBSCAN and OPTICS Algorithms*

Each mining algorithm was provided in a separate Java file containing a separate runnable main class. The first step of the modification was to change the runnable main class to a normal class containing a function that can be called from outside this Java file. Furthermore, SPMF library provides example data sets which are used as an input for each of the mining algorithms. More clearly, the input of each mining algorithm was fixed to one of these example data sets. Since we have chosen the data sets for which the mining algorithms should be applied, therefore, the second modification made to the code was to enable entering the name of the input data set by the user.

Finally, each mining algorithm performs the clustering and writes the resulted clusters into a separate file. This file has a specific format chosen by the algorithm. Since all results are visualized using Python, the resulted clusters should be written into a separate file using a specific format as it is needed by the visualization code in Python.

#### *5.2.3.2 Modified Code in K-Means Algorithm*

Similar to DBSCAN and OPTICS, K-Means was provided in a separate Java file containing a runnable main class. Therefore, the runnable class was changed into normal class containing a function that can be called from outside this file. Moreover, the algorithm was changed in order to enable entering the input data set by the user. Likewise, writing the resulted clusters in the required format is also part of changes made to the K-Means algorithm.

Moreover, K-Means algorithm randomly generates initial centroids in each run. However, PROD provides a prediction of the initial centroids for each data set. It is necessary, for the evaluation, to be able to apply the K-Means algorithm either with these predicted centroids or with randomly initiated centroids. Therefore, the algorithm was modified in order to enable the user to choose either applying K-Means with randomly generated centroids, or applying K-Means with the predicted centroids produced by PROD. In case of choosing the latter option; the CSV file produced by PROD, which contains the predicted centroids, will be read.

#### *5.2.3.3 Additional Code Encapsulating Implementations of Algorithms*

Normally, SPMF library provides different mining algorithms, each provided in a separate runnable Java file. Therefore, when the user wants to apply specific mining algorithm, the respective Java file will be executed. However, since the code of the K-Means, DBSCAN, and OPTICS algorithms was modified into functions that can be called, therefore, a new Java class that is able to call each of the three functions is necessary.

More clearly, a new package called *applyMiningAlgorithms* was added to the SPMF library. This package contains one runnable main class called *ApplyAlgos*. This class provides to the user three different main choices: (1) apply K-Means algorithm, (2) apply DBSCAN algorithm, or (3) apply OPTICS algorithm. For each choice, the class continues by enabling the user to enter the respective predicted input parameters. More specifically, if the user chooses applying K-Means algorithm, hence, the code enables the user to enter the number of clusters, as well as, providing two options regarding the initial centroids. Moreover, in case the user chooses applying DBSCAN, then the code enables the user to enter the two parameters *MinPts* and $\varepsilon$. Similarly, when choosing OPTICS, the user will be able to specify the input parameter $\varepsilon$.

Moreover, these three clustering algorithms require an input data set which is provided in a separate file that has a specific format. This specific format is necessary in order for the file to be readable by the algorithms. The required format is nothing more than just a specific syntax for writing the input. For example, indicating a name for each instance in the file through putting the keyword @NAME before each instance and then writing the wanted name for this instance. Since each of the twenty-eight files containing the chosen data sets has different format than the one required by the algorithms, therefore, a transformation of the format of each of these files is needed. This format transformation is provided by a new added class called *PrepareTextFilesMiningAlgos*. This class is also included in the package *applyMiningAlgorithms* and it takes the original file containing the data set as an input and produces a new file with the required format.

### 5.2.4 Implementation of Previous Prediction Approaches

One part of the evaluation of PROD, as described later in Section 6, depends on comparing this new approach, with respect to some specific aspects, to the other previous prediction approaches. Three out of six previous prediction approaches: Improved K-Means, Silhouette Method, and The Elbow Method were chosen in order to perform this evaluation. The other three approaches: Enhanced K-Means, The Global K-Means, and The Information-Theoretic Approach were not included in the comparison for various reasons. More specifically, The Enhanced K-Means clustering algorithm is mainly based on a heuristic and cannot be proven to work in all cases [19]. Therefore, it was excluded from subsequent comparisons. Furthermore, the main focus of the Global K-Means is on improving the process of finding the centroids of the clusters and while focusing on this aspect it tries to find a good amount of clusters [23]. The latter means that the main goal of this approach is not the same as the main goal of PROD. This fact is sufficient in order to also exclude this approach from later comparisons. Finally, the Information-Theoretic approach is based on information theoretic standards in order to find the best number of clusters, such that these standards are relatively highly complex [24]. Therefore, this approach was also excluded from subsequent evaluations due to its high complexity. On the other hand, the main goal of the other remaining three approaches: Improved K-Means, Silhouette Method, and Elbow Method is finding the best amount of clusters which is the same goal of PROD. Moreover, they are all mainly based on performing some proven measurements and applying K-Means multiple times in order to achieve the best value of *k* [20, 24, 25, 26]. Consequently, these three approaches were chosen to be implemented and used in all subsequent comparisons.

Each of these three approaches was implemented in Java code. Since each of these previous approaches mainly depends on applying the original K-Means algorithm, hence, the implementation of each of them is added to the SPMF library as a separate package. More specifically, a new package called *improvedkmeans* is added to the SPMF library. This package contains a class called *ImprovedKMeans*. This class is responsible for (1) reading the name of the input data set for which the Improved K-Means algorithm is applied, and (2) applying the Improved K-Means algorithm. Furthermore, implementing the Silhouette method is done through adding a new package called *silhouettemethod*. This package contains the class *SilhouetteMethod* which has the required functions for reading the input data set as well as applying the Silhouette method for the specified data set. Similarly, the new package *elbowmethod* is added to SPMF, with the class *ElbowMethod* inside. This class enables the user to specify the input data set and has the function required for applying the Elbow method. This function computes the SSE of the resulted clusters for each value of *k* which is later visualized.

## 5.3 Python Visualization

Python is a general-purpose language with an easy-to-understand syntax, and it is amongst the most popular languages used for data analysis. Python can read multiple file types, and CSV is one type that can be read by this language. Python can be used to easily read and plot data stored inside these files.

Since PROD stores the result of each step in a CSV file, the visualization of these results can be implemented using Python. Those CSV files are stored in the project folder and the path in which they are stored will be specified in the Python code. More specifically, the result of the first step performed by PROD, which is the sampling, is a list of the chosen data points in the sample, i.e., the CSV file resulted from this step will contain these data points such that in each line the *x*-coordinate and the *y*-coordinate of the data point will be stored in first column and second column respectively. Moreover, the result of computing the Voronoi diagram is a list of all edges of this diagram. Therefore, the resulted CSV file will contain four columns such that for each Voronoi edge the starting point will be stored in the first two columns, and similarly, the ending point will be stored in the second two columns. The next step performed by PROD is the fine tuning in which the result is reduced number of Voronoi edges, i.e., Voronoi edges of the new merged regions. Hence, the resulted CSV file will also contain four columns in which each row represents one Voronoi edge.

Python provides a library called *matplotlib* which can be used for plotting points and lines. More specifically, this library provides a function called *scatter()* which is used for plotting the data points. Moreover, regarding the visualization of lines, *matplotlib* provides two functions: the function *add_line()* which is used, as a first step, to add the lines to the subplot on which these lines will be visualized, and the function *plot()* which provides the actual visualizing for these lines.

Having these three functions provided by the *matplotlib* library, as well as the three CSV files containing the intermediate results from the sampling, space partitioning, and fine tuning steps; these results can be visualized through reading these CSV files using Python. More precisely, visualizing the sampled data set is done through reading the CSV file containing the chosen data points and then calling the function *scatter()*. Moreover, in order to draw the computed Voronoi diagram, the CSV file that contains the Voronoi edges will be read and the lines representing the edges will be added to the subplots using *add_line()* function and later visualized using the function *plot()*. Similarly, visualizing the result of the fine tuning step is done through reading the file that contains the new Voronoi edges and then calling the two functions *add_line()* and *plot()*.

## 5.4 Data Sets

Running all experiments require using various data sets on which PROD is applied and evaluated. More specifically, twenty-eight data sets were used in total. All of these used twenty-eight data sets are two-dimensional as PROD works on a two-dimensional space. Most of these data sets were chosen from the GitHub website [31], while others are provided by some Universities [32, 33].

Table 1 provides the most relevant information about each data set. The provided information are: the alias name which is given to this data set during the experiments, i.e., the used name, the original name of the data set as it is provided by its source, the number of instances in this data set, and the source from which the data set was chosen.

The used twenty-eight data sets are variant. More clearly, each data set has a specific shape and a specific positioning of data points in the space which is different from the positioning of data points contained in another data set. Furthermore, these data sets have varying number of data points inside, i.e., they are not of the same size whereas they have varying sizes. These two facts are important in order to justify that PROD works in a more general context, i.e., has a higher probability to give correct predictions as well for previously unseen data sets. In order to visually understand how these twenty-eight data sets are variant, a screenshot for each of them is provided in the appendix at the end of the thesis. Each of these data sets was provided by a specific source except Diamond2 data set. This data set was not provided by any source, whereas, it is a fraction of the Diamond9 data set. More clearly, Diamond2 data set was constructed through taking a small part from the bigger data set which is the Diamond9 data set.

| Used Name | Original Name | Number of Instances | Source |
|---|---|---|---|
| **4C** | 2d-4c-no9 | 876 | [31] |
| **20C** | 2d-20c-no0 | 1517 | [31] |
| **Aggregation** | Aggregation | 788 | [32] |
| **Compound** | Compound | 399 | [32] |
| **Curet0** | cure-t0-2000n-2D | 2000 | [31] |
| **Curet1** | cure-t1-2000n-2D | 2000 | [31] |
| **D31** | D31 | 3100 | [32] |
| **Dartboard** | dartboard1 | 1000 | [31] |
| **Diamond2** | Diamond2DataSet | 667 | Fraction of diamond9 |
| **Diamond9** | diamond9 | 3000 | [31] |
| **DS** | DS850 | 850 | [31] |
| **Elliptical** | elliptical_10_2 | 500 | [31] |
| **Flame** | Flame | 240 | [32] |
| **Fourty** | fourty | 1000 | [31] |
| **Jain** | Jain | 373 | [32] |
| **Long1** | long1 | 1000 | [31] |
| **Longsquare** | longsquare | 900 | [31] |
| **Lsun** | Lsun | 400 | [33] |
| **R15** | R15 | 600 | [32] |
| **Shapes4** | 2d-4c | 1261 | [31] |
| **Shapes9** | 2d-10c | 2990 | [31] |
| **Smile1** | smile1 | 1000 | [31] |
| **Spherical4** | spherical_4_3 | 400 | [31] |
| **Spherical6** | spherical_6_2 | 300 | [31] |
| **Triangle** | triangle1 | 1000 | [31] |
| **Xclara** | xclara | 3000 | [31] |
| **Zelnik4** | zelnik4 | 622 | [31] |
| **Zelnik5** | zelnik5 | 512 | [31] |

*Table 1. The twenty-eight used Data Sets*

# 6. Evaluation

Evaluating PROD requires evaluating each step performed by this approach. More clearly, each of the four main steps: sampling, Voronoi diagram, fine tuning, and prediction; should be evaluated. All previously mentioned twenty-eight data sets were used in order to perform the evaluation of each of these four main steps.

Subsection 6.1 starts by evaluating the first step in PROD, which is the position-based sampling. Next, an evaluation of the computation of Voronoi diagram for the sampled data set is performed in Subsection 6.2. Furthermore, in Subsection 6.3, the evaluation of the third step in PROD, which is the fine tuning of the Voronoi diagram, is performed. Finally, Subsection 6.4 evaluates the final prediction performed by PROD in terms of quality and runtime behavior.

## 6.1 Sampling Evaluation

The evaluation of the position-based sampling performed by PROD is done through evaluating the time required in order to perform this sampling for each data set. More specifically, in order to perform the position-based sampling; the NNS should be applied for each data point in the data set, then, the actual sampling is applied.

Therefore, computing the required time by the position-based sampling is done through measuring the time for applying the NNS for all data points in addition to the time for applying the actual sampling. The measured time is evaluated and compared, for each data set, to the time required by a random sampling.

The sample size for each data set is determined by the position-based sampling approach itself, such that after identifying density areas, the approach chooses from each area $log_{10}(number\ of\ points\ containted\ in\ this\ area)$ data points to be included into the sample. In order to perform a comparison between the two sampling approaches with respect to the runtime, all other aspects should be equal. More clearly, the sample size resulted from both sampling approaches should be the same. Since the sample size is determined by the position-based sampling approach itself, therefore, for each data set, this approach was applied first, then the size of the resulted sample was given as an input for the random sampling in order to construct a sample with this size. Table 2 shows for each data set the size of the sample which was constructed by both sampling approaches.

Through inspecting Table 2, it is noticeable that the constructed samples sizes are relatively small. More clearly, for twenty-seven out of twenty-eight data sets, the size of the sample is less than the half size of the data set, i.e., the sample contains number of data points less than half of the number of data points contained in the original data set. Having the size of the data sets reduced by more than half is of huge benefit for reducing the complexity of the subsequent step. More precisely, computing the Voronoi diagram for less than the half of the data set is less complex than computing this diagram for the whole data set.

| Data Set | Relative Sample Size | Sample Size in Percentage |
|---|---|---|
| **4C** | 225/876 | 25.7% |
| **20C** | 407/1517 | 26.8% |
| **Aggregation** | 167/788 | 21.2% |
| **Compound** | 121/399 | 30.3% |
| **Curet0** | 672/2000 | 33.6% |
| **Curet1** | 650/2000 | 32.5% |
| **D31** | 840/3100 | 27.1% |
| **Dartboard** | 504/1000 | 50.4% |
| **Diamond2** | 213/667 | 31.9% |
| **Diamond9** | 916/3000 | 30.5% |
| **DS** | 253/850 | 29.8% |
| **Elliptical** | 141/500 | 28.2% |
| **Flame** | 51/240 | 21.3% |
| **Fourty** | 243/1000 | 24.3% |
| **Jain** | 94/373 | 25.2% |
| **Long1** | 239/1000 | 23.9% |
| **Longsquare** | 228/900 | 25.3% |
| **Lsun** | 40/400 | 10% |
| **R15** | 132/600 | 22% |
| **Shapes4** | 326/1261 | 25.9% |
| **Shapes9** | 742/2990 | 24.8% |
| **Smile1** | 420/1000 | 42% |
| **Spherical4** | 131/400 | 32.8% |
| **Spherical6** | 67/300 | 22.3% |
| **Triangle** | 260/1000 | 26% |
| **Xclara** | 757/3000 | 25.2% |
| **Zelnik4** | 150/622 | 24.1% |
| **Zelnik5** | 108/512 | 21.1% |

*Table 2. Samples Sizes Constructed by Position-Based Sampling and Random Sampling.*

Regarding runtime measurement, and since these measurements are prone to errors due to resource allocation, the position-based sampling as well as random sampling were applied five times for each data set, then the median value is computed for this data set. Table 3 illustrates, for each data set, the median runtime of performing the position-based sampling and the median runtime of performing the random sampling.

Obviously, it is noticeable from Table 3 that the runtime of performing a random sampling, for each of the twenty-eight data sets, is always less than the runtime of performing the position-based sampling. This is due to the processing that should be done by the position-based sampling, i.e., NNS, setting the threshold, and checking distances of all data points. On the other hand, in the random sampling, no processing is required, whereas, just a random selection for a specific number of data points is performed.

| **Data Set** | **Required Time in Seconds** | |
|---|---|---|
| | **Position-Based Sampling** **(median of 5 runs)** | **Random Sampling** **(median of 5 runs)** |
| **4C** | 0.81 | 0.02 |
| **20C** | 1.84 | 0.02 |
| **Aggregation** | 0.74 | 0.02 |
| **Compound** | 0.28 | 0.02 |
| **Curet0** | 2.86 | 0.02 |
| **Curet1** | 2.88 | 0.02 |
| **D31** | 6.42 | 0.05 |
| **Dartboard** | 0.97 | 0.02 |
| **Diamond2** | 0.61 | 0.02 |
| **Diamond9** | 5.94 | 0.05 |
| **DS** | 0.75 | 0.02 |
| **Elliptical** | 0.38 | 0.02 |
| **Flame** | 0.14 | 0.02 |
| **Fourty** | 0.99 | 0.02 |
| **Jain** | 0.24 | 0.02 |
| **Long1** | 1.0 | 0.02 |
| **Longsquare** | 0.88 | 0.02 |
| **Lsun** | 0.3 | 0.02 |
| **R15** | 0.5 | 0.02 |
| **Shapes4** | 1.58 | 0.02 |
| **Shapes9** | 6.61 | 0.03 |
| **Smile1** | 0.94 | 0.02 |
| **Spherical4** | 0.30 | 0.02 |
| **Spherical6** | 0.19 | 0.02 |
| **Triangle** | 1.03 | 0.02 |
| **Xclara** | 7.08 | 0.03 |
| **Zelnik4** | 0.55 | 0.02 |
| **Zelnik5** | 0.41 | 0.02 |

*Table 3. Runtime of Position-Based Sampling and Random Sampling*

However, in the beginning of Subsection 4.2, the benefit of position-based sampling over random sampling with respect to the resulted sampled data set is described. This benefit indicates that the characteristics of the data set are better reflected in the sample constructed by position-based sampling than the sample constructed by random sampling.

Additionally, the position-based sampling has another important advantage over the random sampling. This advantage is that the position-based sampling is a deterministic sampling. More precisely, applying this sampling on the same data set many times; will produce, in each time, the same resulted sampled data set.

On the other hand, the random sampling is not a deterministic sampling. More clearly, applying the random sampling many times, on the same data set, will produce a different nondeterministic result each time. Having a deterministic sampling is of huge benefit for PROD as the subsequent steps mainly depend on the sampled data set. Therefore, having some knowledge about how the result of the sampling will be is very crucial for PROD.

## 6.2 Space Partitioning Evaluation

Evaluating the space partitioning approach applied for the sampled data set is done through evaluating the time needed to compute the Voronoi diagram for this sampled data set. Since the used Simple Voronoi library produces only Voronoi edges, hence, the construction of the Voronoi regions should also be applied. Therefore, the time required for the space partitioning step is divided into: time required for producing the Voronoi edges by the library, in addition to the time required for constructing the Voronoi regions by PROD.

More specifically, two runtime measurements for computing the Voronoi diagram, for each of the twenty-eight data sets, will be performed as follows: Firstly, the runtime for only computing the Voronoi edges, and Secondly, the runtime for computing these edges in addition to constructing the Voronoi regions. Similar to measuring the runtime of the sampling step, the computation of the Voronoi diagram will be applied five times for each data set, then the median value is computed for this data set. Table 4 shows these two median values.

| **Data Set** | **Required Time in Seconds** | |
|---|---|---|
| | **Producing Voronoi Edges (median of 5 runs)** | **Producing Voronoi Edges and Constructing Voronoi Regions (median of 5 runs)** |
| **4C** | 0.05 | 0.14 |
| **20C** | 0.06 | 0.67 |
| **Aggregation** | 0.05 | 0.09 |
| **Compound** | 0.06 | 0.09 |
| **Curet0** | 0.08 | 3.55 |
| **Curet1** | 0.06 | 3.08 |
| **D31** | 0.08 | 8.09 |
| **Dartboard** | 0.06 | 0.92 |
| **Diamond2** | 0.06 | 0.16 |
| **Diamond9** | 0.09 | 10.48 |
| **DS** | 0.05 | 0.20 |
| **Elliptical** | 0.06 | 0.09 |
| **Flame** | 0.05 | 0.06 |
| **Fourty** | 0.05 | 0.17 |
| **Jain** | 0.06 | 0.08 |
| **Long1** | 0.05 | 0.17 |
| **Longsquare** | 0.05 | 0.16 |
| **Lsun** | 0.06 | 0.08 |
| **R15** | 0.06 | 0.09 |
| **Shapes4** | 0.05 | 0.30 |
| **Shapes9** | 0.08 | 5.42 |
| **Smile1** | 0.05 | 0.44 |
| **Spherical4** | 0.06 | 0.09 |
| **Spherical6** | 0.05 | 0.06 |
| **Triangle** | 0.05 | 0.20 |
| **Xclara** | 0.08 | 4.56 |
| **Zelnik4** | 0.05 | 0.09 |
| **Zelnik5** | 0.06 | 0.08 |
| **AVERAGE Runtime** | **0.06** | **1.41** |

*Table 4. Runtime of Computing Voronoi Diagram*

If the result provided by the Simple Voronoi library, which is only the Voronoi edges, is sufficient, then the runtime of computing Voronoi diagram will be as shown in the left column in Table 4. However, having only the edges is not sufficient for the subsequent steps of PROD, whereas, the Voronoi regions are needed to be constructed. Therefore, the actual runtime for computing the Voronoi edges by the Simple Voronoi library in addition to constructing the Voronoi regions by PROD will be as shown in the right column in Table 4.

The last row in Table 4 shows the average runtime for each of the two cases. Clearly, the average runtime required for only computing Voronoi edges is much less than the average runtime for computing the edges and constructing the regions. However, using more elaborated libraries which may provide an implementation for building Voronoi regions directly, helps in saving this runtime.

## 6.3 Fine Tuning Evaluation

The process of the fine tuning step is mainly divided into two parts: (1) inspecting the Voronoi regions and deciding whether to merge or not, and (2) applying the merging. Therefore, the evaluation of this step consists of two parts. First, in order to decide whether the merging should be applied or not; a threshold, on which all the decisions are based, should be set. This threshold should be chosen carefully since the subsequent steps depend on the resulted Voronoi diagram that contains the merged regions.

Therefore, an evaluation for different settings of this threshold is performed in Subsection 6.3.1, and the best setting is chosen. The second part of the evaluation concerns evaluating the overall time needed in order to perform the inspection of all the regions and to apply the necessary merging decisions. The evaluation of the required time is performed in Subsection 6.3.2.

### 6.3.1 Evaluation of Different Settings of the Threshold Distance

While inspecting Voronoi regions; the decision whether some regions should be merged or not depends on finding points with distances that exceed the threshold distance. Hence, choosing this threshold distance correctly is very important as it can affect the merging decisions and therefore affects the final Voronoi diagram from which the prediction is made.

When inspecting the Voronoi regions; the merging of some regions should only be applied if each of these regions have points placed *close* to other points in the other regions. Therefore, the threshold for deciding whether these points are indeed placed *close* to other points should be chosen relatively to the *closeness* of all points to each other. In order to find the best threshold distance for applying the merging; five different threshold possibilities were evaluated.

#### *6.3.1.1 Region-Specific Thresholds*

The first evaluated possibility is to set a different threshold for each inspected region. This region-specific threshold is set according to the distances between data points contained in this region and the site of this region. More clearly, when inspecting $region_1$ and $region_2$; the approach sets two different thresholds as follows: (1) $threshold_1$ as the average distance between $points_1$in $region_1$ and their site, and (2) $threshold_2$ as the average distance between $points_2$ in $region_2$ and their site.

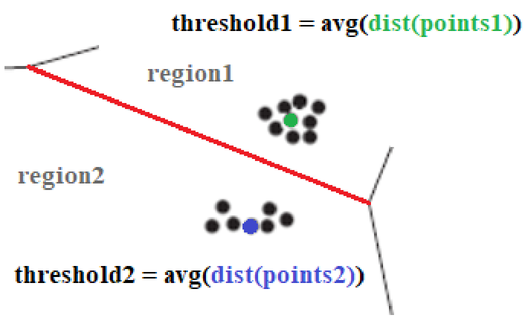

threshold1 = avg(dist(points1))
region1
region2
threshold2 = avg(dist(points2))

*Figure 19. Setting the Region-Specific Thresholds*

Figure 19 shows an example of two Voronoi regions, where the green data point is the site of $region_1$, the red data point is the site of $region_2$, and the red line is the bordering edge of these two regions. The approach sets $threshold_1$ for $region_1$ as the average distance between $points_1$ and the green site, i.e., $threshold_1 = avg(dist(points_1))$. Similarly, the approach sets $threshold_2$ for $region_2$ as the average distance between $points_2$ and the red site, i.e., $threshold_2 = avg(dist(points_2))$.

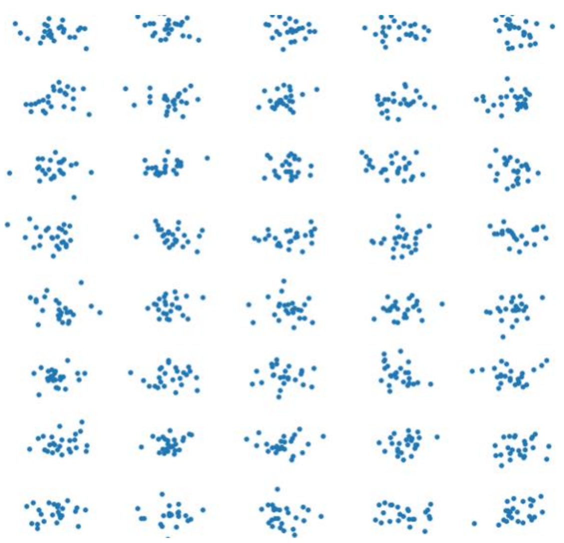


*Figure 20a. Example Data Set*

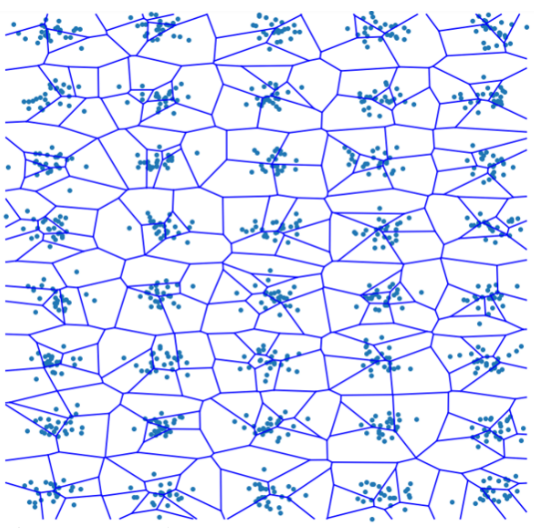


*Figure 20b. Voronoi Diagram for m Points with all n Points Scattered*

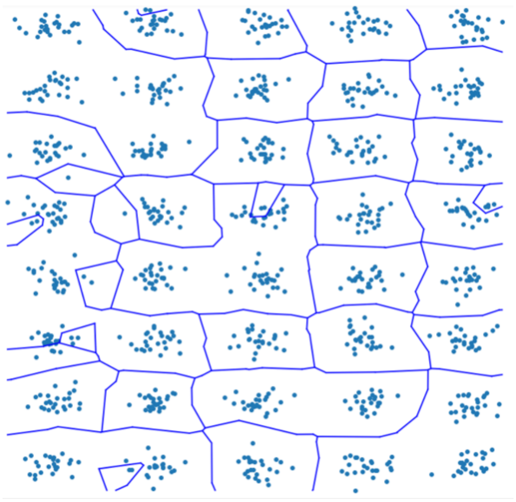


*Figure 20c. Final Voronoi Diagram*

*Figure 20. Example on Fine Tuning with Region-Specific Thresholds*

The approach continues by searching for points in $region_1$ with distances to the bordering edge $\leq avg(dist(points_1))$ and points in $region_2$ with distances to the bordering edge $\leq avg(dist(points_2))$. If and only if the approach finds points in $region_1$ satisfying their own condition and points in $region_2$ also satisfying their own condition; the merging for these two regions will be applied. This setting of the threshold led to merging decisions for regions that should not be merged.

To illustrate the problem more clearly; let us take the example data set in Figure 20a, and the Voronoi diagram for the chosen *m* points with all of the *n* data points scattered over the diagram as shown in Figure 20b. Through setting the threshold of merging for each region as the average distance in that region; the final Voronoi diagram shown in Figure 20c is reached. This final diagram contains wrong merging, for example, all of the regions in the upper left corner of the diagram were merged together into one region, which is obviously not the correct merging to be applied.

### *6.3.1.2 Average-Of-Average Distance*

The second evaluated possibility is to set one common threshold for all regions in the Voronoi diagram. This threshold is set as the average of the average distances of all regions. More specifically, as a first step and before starting to inspect any regions; the approach calculates the average distance in each region. After computing all of the average distances; the approach continues by calculating the average of all of these computed averages, and sets the threshold of merging equals to this general average distance. Figure 21 shows an example for this setting of the threshold.

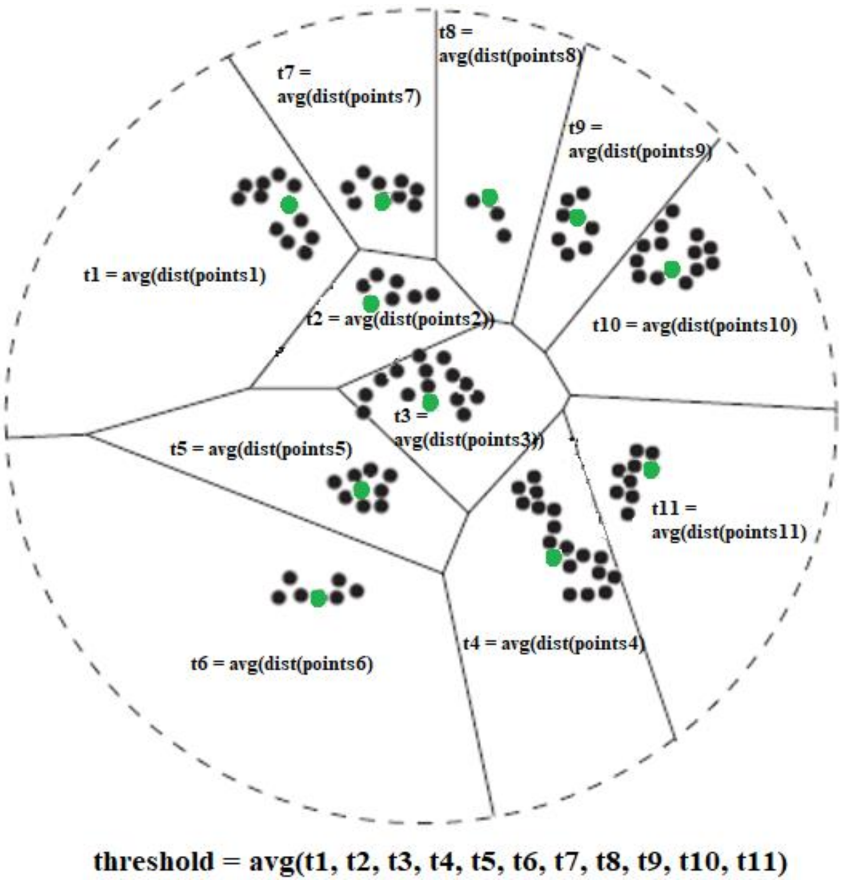


*Figure 21. Setting the Threshold as the Average-of-Average Distance of All Regions*

In Figure 21 an example of a Voronoi diagram with 11 regions, where each green data point is the site of the respective region. The approach starts by calculating the average of each of these 11 regions, i.e., $t_1 = avg(dist(points_1))$, $t_2 = avg(dist(points_2))$, …, $t_{11} = avg(dist(points_{11}))$. Next, the approach continues by computing the general average of $t_1, t_2, \dots, t_{11}$ and sets the $threshold$ to be equal to this computed general average, i.e., $threshold = avg(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11})$.

After computing the general threshold, the approach continues by inspecting each neighboring region. For instance, when inspecting $region_1$ and $region_2$; the approach starts searching for points in $region_1$ with distances to the bordering edge less or equal the general threshold, i.e., the general average distance, and points in $region_2$ with distances to the bordering edge less or equal the same threshold. If and only if the approach finds points in $region_1$ satisfying the condition and points in $region_2$ satisfying the same condition; the merging for these two regions will be applied.

Computing the threshold as described above led to miss some necessary merging decisions. More precisely, the approach rejected some merging decisions for regions that were supposed to be merged. To illustrate the problem more clearly; let us take the same example data set in Figure 20a, and the Voronoi diagram in Figure 20b. Through setting the general threshold of merging for all regions as the average of average distances of all regions; the final Voronoi diagram shown in Figure 22 will be reached.

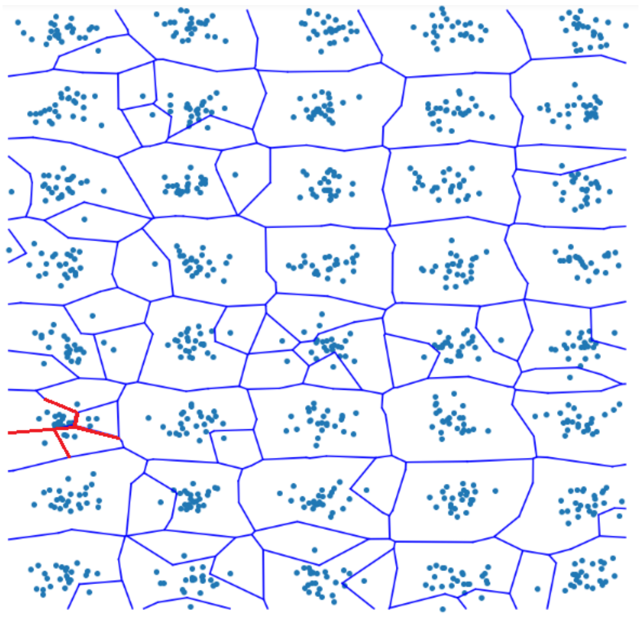


*Figure 22. Example on Fine Tuning with Threshold Equals to Average-of-Average Distance*

This final diagram, shown in Figure 22, contains missed merging, for example, the red regions in the diagram were supposed to be merged together into one region since they have points located very closely to the bordering edges. However, these merging decisions were rejected because some of these regions did not satisfy the condition of merging.

### *6.3.1.3 Average Distance of Two Neighboring Regions*

The third evaluated possibility is to set the threshold of merging as the average distance of the distances in the two inspected regions. More specifically, when inspecting $region_1$ and

$region_2$; and having the distances between points in $region_1$ and their site, and similarly, the distances between points in $region_2$ and their site; the approach calculates the average distance of these distances and sets the threshold equals to this average distance.

The main difference between this setting of the threshold and the second previously described possibility, is that the latter one sets a general threshold for all the regions in the Voronoi diagram which is the average of all average distances of each region. Whereas, in this third possibility, each two neighboring regions have their own threshold, which is the average of all distances in both regions.

To illustrate how the computation of the threshold is performed more clearly, Figure 23 shows an example of setting the threshold for the two neighboring regions: $region_1$ and $region_2$. The green data point is the site of $region_1$, the red data point is the site of $region_2$, and the red line is the bordering edge of these two regions. The approach sets the $threshold$ for these two regions as the average distance of the distances between $points_1$ and the green site and the distances between $points_2$ and the red site, i.e., $threshold = avg(dist(points_1), dist(points_2))$.

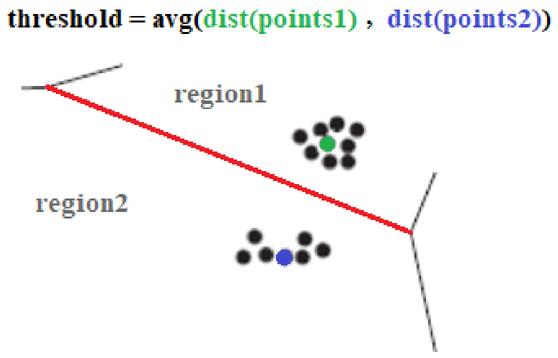


*Figure 23. Setting the Threshold as the Average-Distance of Two Neighboring Regions*

As a next step, the approach starts searching for points in $region_1$ with distances to the bordering edge less or equal the computed average distance of these two regions and points in $region_2$ with distances to the bordering edge less or equal the same computed average distance. If and only if the approach finds points in $region_1$ satisfying the condition and points in $region_2$ satisfying the same condition; the merging for these two regions will be applied.

This setting of the threshold is still leading to reject some merging decisions for regions that should be merged. To illustrate the problem more clearly; let us take the same example data set in Figure 20a, and the Voronoi diagram in Figure 20b. Through setting the threshold of merging for each two inspected regions as the average distance of these regions; the final Voronoi diagram shown in Figure 24 is reached.

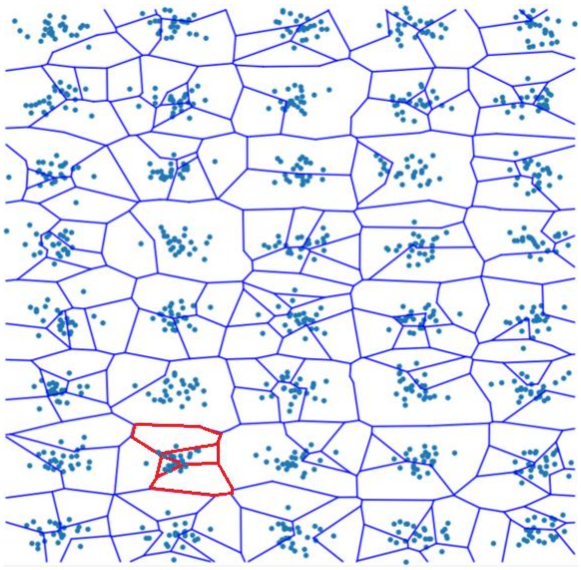


*Figure 24. Example on Fine Tuning with Threshold Equals to the Average Distance of Two Neighboring Regions*

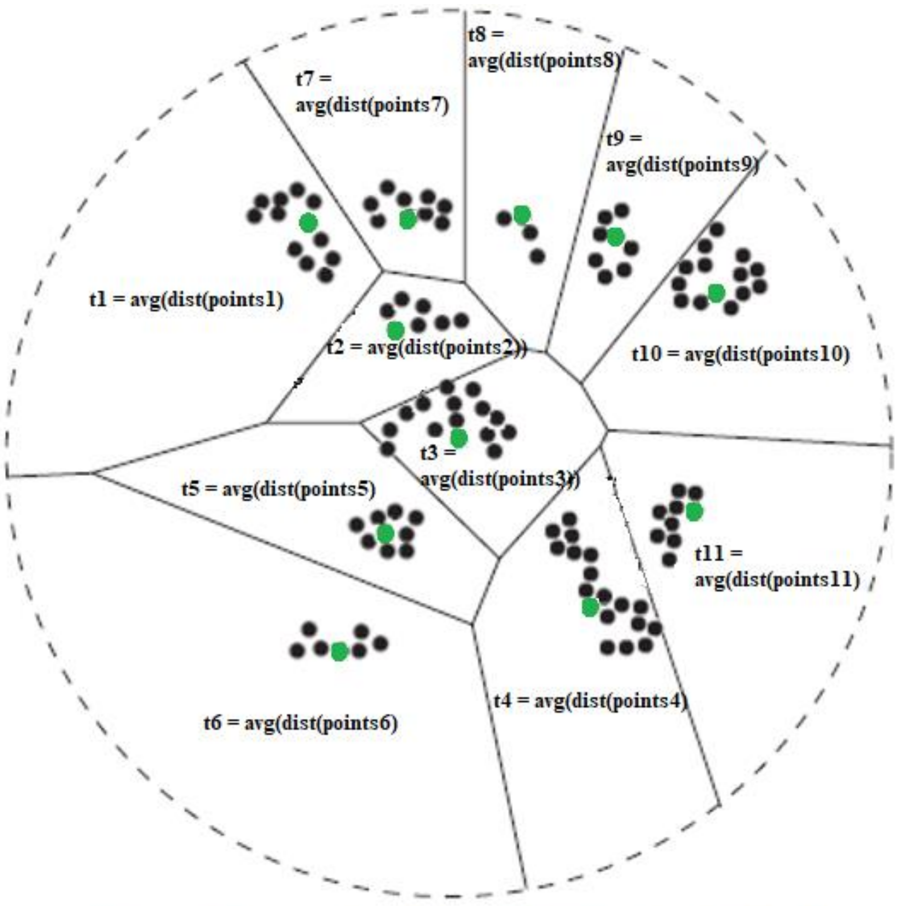
This final diagram, shown in Figure 24, contains missed merging, for example, all of the red regions in the diagram were supposed to be merged together into one region since they have points located very closely to the bordering edges. However, this merging decision was rejected because these regions did not stratify the condition.

#### 6.3.1.4 *Maximum Average Distance*

Similar to the second evaluated possibility; the fourth possibility is also to set one common threshold for all regions in the Voronoi diagram. However, the threshold here is set as the maximum average distance among all average distances of each region. More precisely, the approach starts by calculating the average distance of each region. Then, the approach continues by finding the maximum average distance among of all of these computed average distances, and sets the threshold of merging equals to this maximum average distance.

Figure 25 shows an example of a Voronoi diagram with 11 regions, where each green data point is the site of the respective region. The approach starts by calculating the average of each of these 11 regions, i.e., $t_1 = avg(dist(points_1))$, $t_2 = avg(dist(points_2))$, …, $t_{11} = avg(dist(points_{11}))$. Next, the approach continues by finding the maximum average distance among $t_1, t_2, \ldots, t_{11}$ and sets the $threshold$ to be equal to this maximum average, i.e., $threshold = max(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11})$.

Having the maximum average distance computed; the approach starts inspecting the Voronoi regions. For instance, when inspecting $region_1$ and $region_2$; the approach searches for points in $region_1$ with distances to the bordering edge less or equal the maximum average distance, and points in $region_2$ with distances to the bordering edge less or equal this maximum average distance. If and only if the approach finds points in $region_1$ satisfying the condition and points in $region_2$ satisfying the same condition; the merging for these two regions will be applied.



**threshold= max(t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11)**

*Figure 25. Setting the Threshold as the Maximum Average Distance*

This setting of the threshold led to apply many wrong merging decisions for regions that should not be merged. To illustrate the problem more clearly; let us take the same example data set in Figure 20a, and the Voronoi diagram in Figure 20b. Through setting the threshold of merging for all regions as the maximum average distance; the final Voronoi diagram shown in Figure 26 will be reached.

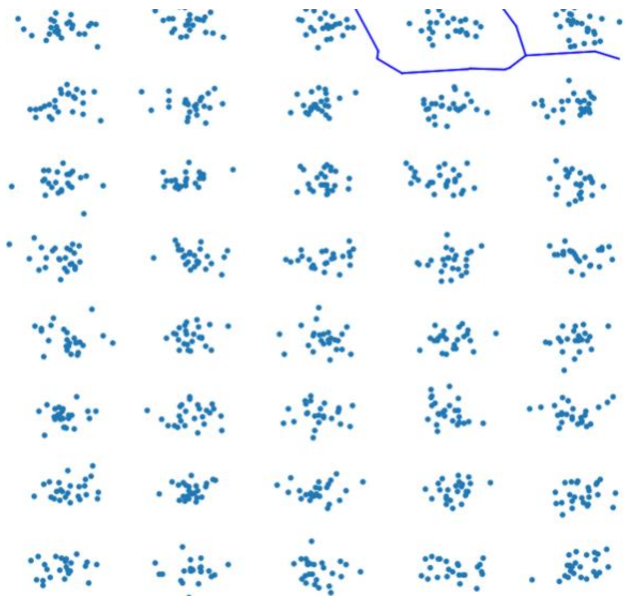


*Figure 26. Example on Fine Tuning with Threshold Equals to the Maximum Average Distance*

Obviously, this final diagram in Figure 26 contains wrong merging in which most of the regions were merged together since they have points with distance to the bordering edge $\leq$ $maximum\_average\_distance$.

### 6.3.1.5 Sampling Threshold

The final evaluated possibility is to set one common threshold for all regions which is here equal to the threshold used in the position-based sampling step. In other words, the threshold of merging is set to the computed average distance of the distances between each data point and its nearest point in the data set. More precisely, before inspecting any region; the approach gets the threshold used in the sampling step, sets the threshold of merging equals to this sampling threshold, and then starts inspecting the Voronoi regions.

As an example, when inspecting $region_1$ and $region_2$; the approach searches for points in $region_1$ with distances to the bordering edge less or equal this sampling threshold, and points in $region_2$ with distances to the bordering edge also less or equal the sampling threshold. If and only if the approach finds points in $region_1$ satisfying the condition and points in $region_2$ satisfying the same condition; the merging for these two regions will be applied.

This setting of the threshold led to take the best merging decisions where the merging was applied only to regions which were supposed to be merged. To illustrate the final result, taking the same example data set in Figure 20a, and the Voronoi diagram with all data points as in Figure 20b; and by applying the inspection of the regions with the $threshold = sampling\_threshold$; a final diagram with reasonable decisions of merging is reached.

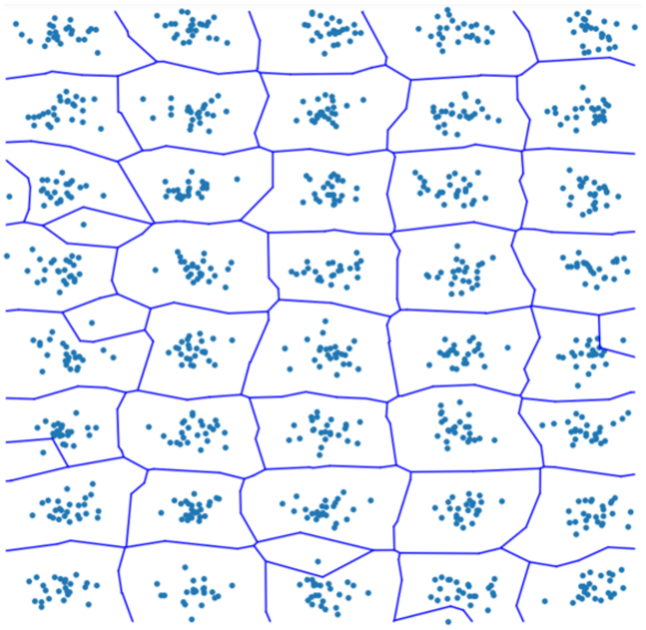


*Figure 27. Example on Fine Tuning with Threshold Equals to the Sampling-Threshold*

The final resulted diagram is shown in Figure 27. Different data sets were evaluated with this threshold. For each of these evaluated data sets; we always ended up with well-performing merging decisions of the Voronoi regions.

### 6.3.2 Evaluation of the Required Time

The second part of evaluating the fine tuning step concerns evaluating the time needed to perform the overall process of inspecting the Voronoi regions and applying the necessary merging. More specifically, the fine tuning of the Voronoi diagram, for each of the twenty-eight data sets, is applied and the time needed for this application is measured.

Similar to the previous runtime measurements of the previous steps, measuring the runtime of performing the fine tuning step is done through executing this step, for each of the twenty-eight data sets, 5 times. Then, the median value for each data set will be computed. Table 5 illustrates, for each data set, the median runtime for performing the overall fine tuning of the Voronoi diagram.

Through inspecting Table 5, it is noticeable that the runtime of performing the fine tuning for the Voronoi diagram, for all data sets, is relatively small. More specifically, the inspection and merging approach performed by PROD for the previously computed Voronoi diagram, has reasonable runtime.

| **Data Set** | **Time Required for Performing the Fine Tuning in Seconds (median of 5 runs)** |
|---|---|
| **4C** | 0.08 |
| **20C** | 0.16 |
| **Aggregation** | 0.05 |
| **Compound** | 0.05 |
| **Curet0** | 0.36 |
| **Curet1** | 0.28 |
| **D31** | 0.58 |
| **Dartboard** | 0.14 |
| **Diamond2** | 0.05 |
| **Diamond9** | **0.67** |
| **DS** | 0.08 |
| **Elliptical** | 0.03 |
| **Flame** | 0.03 |
| **Fourty** | 0.06 |
| **Jain** | 0.03 |
| **Long1** | 0.08 |
| **Longsquare** | 0.08 |
| **Lsun** | 0.05 |
| **R15** | 0.03 |
| **Shapes4** | 0.11 |
| **Shapes9** | 0.39 |
| **Smile1** | 0.13 |
| **Spherical4** | 0.03 |
| **Spherical6** | **0.02** |
| **Triangle** | 0.06 |
| **Xclara** | 0.5 |
| **Zelnik4** | 0.09 |
| **Zelnik5** | 0.03 |

*Table 5. Runtime of Performing the Fine Tuning*

## 6.4 Evaluation of the Overall PROD Approach

The evaluation process of the overall prediction made by PROD is divided into two main parts. The first part concerns evaluating the accuracy, quality, and runtime of predicting the parameter of the K-Means algorithm. The second part in the evaluation is about evaluating the process of predicting the DBSCAN and OPTICS parameters.

### 6.4.1 Evaluation of Predicting K-Means Parameter

The evaluation of the prediction made by PROD for predicting the K-Means parameter consists of four steps. The first step is to evaluate the accuracy of the predicted *k*, i.e., the predicted number of clusters, with respect to the optimal *k* of each data set. The second step of the evaluation is about evaluating the quality of PROD in terms of measuring the Sum of the Squared Errors of the resulted clusters. Furthermore, the third step concerns evaluating the time needed by PROD to perform the prediction. Finally, the last step of evaluating the prediction of K-Means parameter is to measure and evaluate number of iterations performed by the original K-Means algorithm, when it is provided with the predicted *k* as an input. In order to perform all of these evaluations, twenty-six out of twenty-eight data sets were used, such that the remaining two data sets: Dartboard and Smile1 clearly demand a density-based algorithm.

#### *6.4.1.1 Accuracy Evaluation*

The accuracy of the prediction is measured and evaluated with respect to the optimal *k* for each data set. More specifically, the accuracy of the prediction made by PROD is evaluated through comparing it, with respect to the optimal *k*, to the accuracy of the prediction made by the other three previous prediction methods: Improved K-Means, Silhouette Method, and Elbow Method. In order to do this comparison; these three prediction methods are implemented and applied for each of the twenty-six data sets. More clearly, for each data set, PROD as well as the other three approaches were applied.

Having the predicted *k* from each of the four approaches; a comparison between each of these four *k*s and the optimal *k* is performed and the difference between each predicted *k* and this optimal *k* is calculated. This difference shows how much the predicted *k* differs from the expected *k*. Furthermore, for each approach; the average difference of all predictions made by this approach is calculated. Finally, the average difference of PROD is compared to the average difference of each of the other three approaches.

Since this evaluation requires knowing the optimal *k* for each data set; detecting this optimal *k* was necessary. For some data sets, the source from which these data sets were chosen, provides the optimal *k* for each of these data sets. However, for other data sets, no optimal *k* was provided explicitly, whereas, for some of them it was indicated implicitly in the name of the data set, and for others no optimal *k* was provided even implicitly. While searching for the optimal *k* of the latter data sets; some issues emerged. As clustering is unsupervised learning, hence, the main issue is that knowing the '*optimal' k* in advance is very difficult. Moreover, even if the optimal *k* was indicated for a data set; it is still not guaranteed that it is indeed the optimal number of clusters for this data set. For instance, Figure 28a shows an example of a data set called *Aggregation Data Set*. The optimal *k* for this data set is given as 7 by the source from which this data source was chosen [32], and Figure 28b illustrates the clustering of this data set into seven clusters. However, it is not guaranteed that this optimal $k$ is indeed optimal in all cases. More clearly, one analyst may find that the optimal $k$ here is 5 instead of 7. Figure 28c illustrates clustering this data set into five clusters. Through looking to Figure 28b and Figure 28c, we can indeed find that the two clustering options are valid, and it depends on the analyst to decide which one is the optimal clustering for him, i.e., the optimality of $k$ depends on the individual perception of clustering.

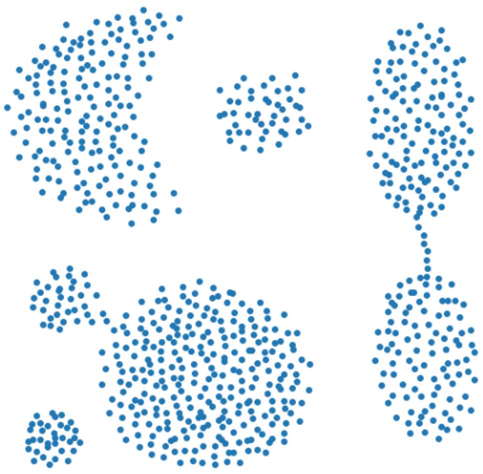


*Figure 28a. Aggregation Data Set*

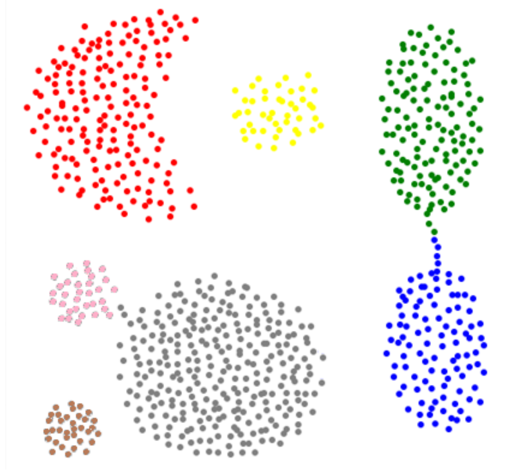


*Figure 28b. Clustering into 7 Clusters*

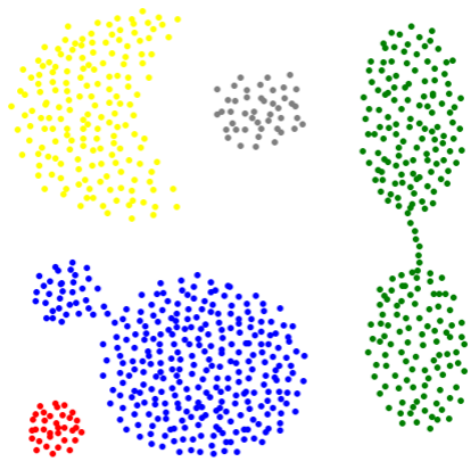


*Figure 28c. Clustering into 5 Clusters*

*Figure 28. Different Clustering for Aggregation Data Set*

Table 6 shows, for each of the twenty-six data sets, the optimal $k$ as well as the predicted $k$ from PROD and from the other three approaches. Each of the three approaches: Improved K-Means, Silhouette method, and Elbow method requires applying the original K-Means algorithm multiple times in order to perform the prediction. More precisely, for the Improved K-Means approach, K-Means was applied for each value of $k$ in the range $[2, \sqrt{n}]$, such that $n$ is the number of data points in the data set. Furthermore, regarding each of the Silhouette method and Elbow method, K-Means was applied for each value of $k$ in the range $[1, n]$. K-Means is not deterministic because it depends, on one hand, on the random initialization of the centroids, and on the other hand, on finding local minimum. This fact might lead to different results in each run of the algorithm. Therefore, for each data set, each of the other approaches was applied 5 times and the median value is chosen for this data set.

Since some of the approaches requires in some cases very long time to be executed; a threshold for the execution time is set to 30 minutes. More clearly, the sign '-' in the approach column in the table indicates that this approach, when applying it to the respective data set, took more than 30 minutes and did not finish, hence, the execution was stopped and the prediction could not be completed.

| **Data Set** | **Optimal K** | **Predicted K** | | | |
|---|---|---|---|---|---|
| | | **PROD** | **Improved K-Means (median of 5 runs)** | **Silhouette Method (median of 5 runs)** | **Elbow Method** |
| **4C** | 4 | 3 | 4 | 7 | Many elbows |
| **20C** | 20 | 20 | 29 | - | Many elbows |
| **Aggregation** | 7 | 5 | 4 | 4 | Many elbows |
| **Compound** | 6 | 3 | 2 | 2 | Many elbows |
| **Curet0** | 3 | 3 | 7 | - | Many elbows |
| **Curet1** | 4 | 5 | 8 | - | Many elbows |
| **D31** | 31 | 13 | 34 | - | - |
| **Diamond2** | 2 | 2 | 2 | 2 | Many elbows |
| **Diamond9** | 9 | 4 | 13 | - | Many elbows |
| **DS** | 5 | 5 | 7 | 2 | Many elbows |
| **Elliptical** | 7 | 7 | 12 | 11 | Many elbows |
| **Flame** | 2 | 1 | 2 | 2 | Many elbows |
| **Fourty** | 40 | 40 | 29 | 70 | Many elbows |
| **Jain** | 2 | 2 | 12 | 9 | Many elbows |
| **Long1** | 2 | 2 | 14 | 7 | Many elbows |
| **Longsquare** | 6 | 6 | 2 | 2 | Many elbows |
| **Lsun** | 3 | 4 | 2 | 2 | Many elbows |
| **R15** | 15 | 11 | 17 | 28 | Many elbows |
| **Shapes4** | 4 | 4 | 5 | 12 | Many elbows |
| **Shapes9** | 9 | 9 | 12 | 3 | Many elbows |
| **Spherical4** | 4 | 4 | 5 | 5 | Many elbows |
| **Spherical6** | 6 | 6 | 4 | 5 | Many elbows |
| **Triangle** | 4 | 4 | 5 | 4 | Many elbows |
| **Xclara** | 3 | 3 | 3 | - | - |
| **Zelnik4** | 4 | 4 | 12 | 11 | Many elbows |
| **Zelnik5** | 4 | 4 | 12 | 9 | Many elbows |

*Table 6. The Optimal $k$ and the Four Predicted $k$s for Each Data Set*

In the Elbow method, a function is computed and exploited to produce a prediction for the best *k*. This function is computed in terms of the SSE value of the resulted clusters of each value of *k*. Furthermore, this computed function is drawn for each data set, and then the "elbow" in the function should be identified in order to determine the best *k* for this data set. However, after drawing the function for each of the twenty-six data sets, and by inspecting each drawn function; many elbows were found in each of them. More clearly, none of the drawn functions contained only one clear identifiable elbow, whereas, they all contained many elbows, which is indicated by the phrase "Many elbows" in the table. Consequently, since it is not clear which elbow should be chosen, the number of clusters for all data sets could not be predicted and the median value could not be computed using the Elbow method. Hence, this method could not be taken into consideration in the subsequent evaluations.

Through applying the approaches to each of the twenty-six data sets, and by having the predicted *k* from PROD and the two median values of the predicted *k*s from Improved K-Means and Silhouette Method; the difference between each predicted *k* and the respective optimal *k* of the data set is calculated. Table 7 shows, for each data set, the calculated difference between each predicted *k* and the respective optimal *k*. The last row in the Table 7 shows the average difference for each approach among all data sets.

The sign '-' in Table 7 indicates that the prediction of this approach for the respective data set could not be completed, hence, the *k* is not available and this data set will not be taken into consideration when computing the average difference for this approach. More specifically, the prediction of the silhouette method could not be completed for six data sets, therefore, when calculating the average difference for this method, the total number of data sets is twenty instead of twenty-six.

| **Data Set** | **Difference between Predicted *k* and Optimal *k*** | | |
|---|---|---|---|
| | **PROD** | **Improved K-Means** | **Silhouette Method** |
| **4C** | 1 | 0 | 3 |
| **20C** | 0 | 9 | - |
| **Aggregation** | 2 | 3 | 3 |
| **Compound** | 3 | 4 | 4 |
| **Curet0** | 0 | 4 | - |
| **Curet1** | 1 | 4 | - |
| **D31** | 18 | 3 | - |
| **Diamond2** | 0 | 0 | 0 |
| **Diamond9** | 5 | 4 | - |
| **DS** | 0 | 2 | 3 |
| **Elliptical** | 0 | 5 | 4 |
| **Flame** | 1 | 0 | 0 |
| **Fourty** | 0 | 11 | 30 |
| **Jain** | 0 | 10 | 7 |
| **Long1** | 0 | 12 | 5 |
| **Longsquare** | 0 | 4 | 4 |
| **Lsun** | 1 | 1 | 1 |
| **R15** | 4 | 2 | 13 |
| **Shapes4** | 0 | 1 | 8 |
| **Shapes9** | 0 | 3 | 6 |
| **Spherical4** | 0 | 1 | 1 |
| **Spherical6** | 0 | 2 | 1 |
| **Triangle** | 0 | 1 | 0 |
| **Xclara** | 0 | 0 | - |
| **Zelnik4** | 0 | 8 | 7 |
| **Zelnik5** | 0 | 8 | 5 |
| **AVERAGE Difference** | 1,38 | 3,92 | 5,25 |

*Table 7. The Difference Between Each Predicted k and the Optimal k*

Having the computed difference equals to zero means that the predicted *k* is equal to the optimal *k* for the respective data set. More specifically, through applying PROD; the predicted *ks* for seventeen data sets were exactly the same as the indicated optimal *ks* for each of these data sets. More clearly, PROD predicted the optimal *k* for seventeen out of twenty-six data sets, i.e., for more than the half of data sets. On the other hand, by applying Improved K-Means, the optimal *k* was predicted for only four out of twenty-six data sets. Furthermore, Silhouette method was able to predict the optimal *k* for three out of twenty data sets only.

Through inspecting Table 7, it is noticeable that the predicted *k* for the data set D31 from PROD differs from the optimal *k* by 18, which is a huge difference compared to the rest of the computed differences for this approach. This significant difference is caused by the density of the D31 data set.

More clearly, as shown in Figure 29a, the D31 data set is a dense data set in which the data points inside are not well separated. More clearly, as PROD mainly depends, as a first step, on identifying separated density areas in the data set, hence, it was not able to identify the correct density areas as they are not clearly separated in the D31 data set which lead at the end to wrong prediction. However, even though the predicted number of clusters is not as expected, the resulted 13 clusters by PROD are logical. In order to illustrate this fact, Figure 19b shows the resulted clusters from PROD such that each cluster is represented in different color. On the other hand, the original K-Means algorithm was also applied for this data set, such that *k* was given the value 13, i.e., the same predicted value by PROD. Figure 19c shows the 13 clusters resulted from applying the original K-Means with random centroids. Through inspecting Figures 29b and 29c, it is noticeable that the 13 resulted clusters by PROD, as shown in Figure 29b, are more logical than the 13 clusters resulted from applying K-Means as in Figure 29c. One important benefit, that makes the clustering made by PROD more logical, is the fact that this approach is able to identify outlier data points, which are shown as black points in Figure 29b.

Moreover, through inspecting and comparing Figures 29b and 29c; it is noticeable that the clustering of PROD somehow corresponds to the density. For example, the pink cluster in Figure 29b contains close data points, i.e., not well separated data points, which correspond to a dense area. Whereas, the clusters produced by K-Means mainly depend on the positioning of the randomly initiated centroids. For instance, Figure 29c shows the light blue cluster which obviously contains data points that should be included into the light green cluster instead. This is due to the unsatisfying random initiation of centroids as well as the fact that density of data points is not taken into consideration by K-Means.

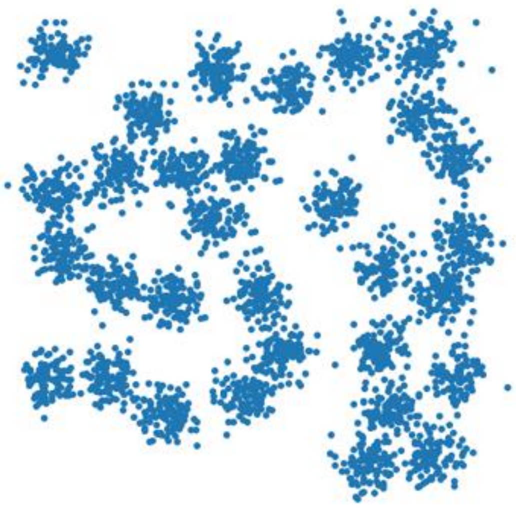


*Figure 29a. D31 Data Set*

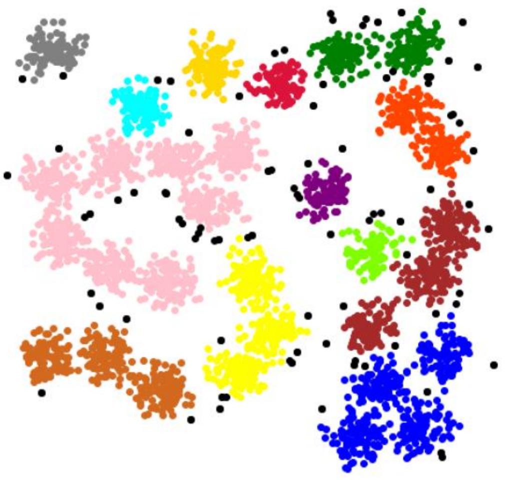


*Figure 29b. Result of Applying PROD*

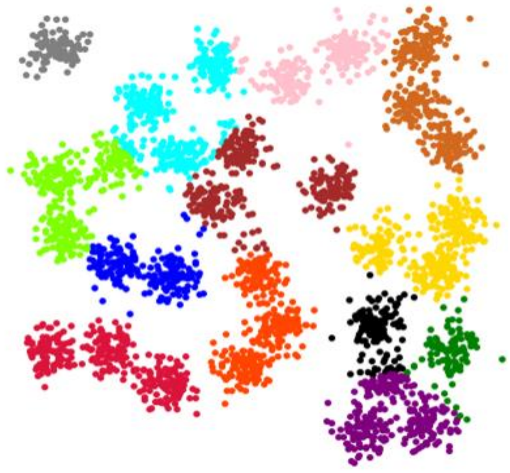


*Figure 29c. Result of Applying K-Means with Random Centroids*

*Figure 29. Clustering the D31 Data Set*

Having the average difference computed for each approach, as well as the predicted $k$ for each data set; the range in which the optimal $k$ of this data set lies in can be computed accordingly. More clearly, having the average difference as $avgDiff$ and the predicted number of clusters as $k_{pred}$; the optimal $k$ for the respective data set lies in the range $[k_{pred} - avgDiff\ , k_{pred} + avgDiff]$. Since number of clusters should be an integer number; the computed average difference will be rounded up to the nearest integer number.

More specifically, regarding PROD; the average difference is equal to 1,38 and after rounding up to the nearest integer, we end up with an average difference equals to 2. Therefore, the range in which the optimal $k$ lies in is $[k_{pred} - 2\ , k_{pred} + 2]$. Clearly, PROD dramatically reduces the solution space for predicting $k$, i.e., reduces the values of $k$ for which the analyst might try to apply the original K-Means algorithm. This means that after having the predicted $k$ from PROD; and if the analyst is not satisfied with this predicted $k$, there exist only four more values for which this analyst might apply the original K-Means algorithm, which is not cumbersome. In case the average difference is computed without the D31 data set, then we end up with a rounded average difference equals to 1. Therefore, the range in which optimal $k$ lies in will be

$[k_{pred} - 1\, , k_{pred} + 1]$, i.e., if the analyst is not satisfied with the predicted *k*; he has to try only two more values.

Moreover, through inspecting Table 7, it is noticeable that for nine data sets the predicted *k*s by PROD are not equal to the optimal *k*. Taking these nine predicted *k*s; it is noticeable that seven of them have values below the optimal *k*, i.e., for more than the half of these nine predicted *k*s, the value is less than the value of the optimal *k*. This fact means that when the analysts are not satisfied with the predicted $k$ and want to test another value for $k$ in the range $[k_{pred} - 2\, , k_{pred} + 2]$; they can start with the upper half of the range, i.e., with the two values of *k*: $k_{pred} + 1$ and $k_{pred} + 2$.

Regarding the predicted *k*s by the Improved K-Means; the average difference is equal to 3,92. Therefore, after rounding up this values, we end up with a range of values equals to $[k_{pred} - 4\, , k_{pred} + 4]$. Moreover, for the silhouette method, having 5,25 as the average difference and 6 as the rounded value, we end up with the following range of values $[k_{pred} - 6\, , k_{pred} + 6]$. Clearly, PROD caused the highest reduction in the solution space for predicting the $k$.

#### *6.4.1.2 Sum of Squared Errors Evaluation*

The quality of the prediction made by PROD is evaluated in terms of measuring the Sum of the Squared Errors (SSE) of the resulted clusters. The evaluation of the sum of the squared errors is measured for (1) the final result produced by PROD, (2) the clusters resulted from applying K-Means algorithm with randomly initiated centroids, and (3) the clusters resulted from applying K-Means algorithm with the predicted centroids by PROD.

The final result produced by PROD is the final Voronoi diagram after merging some regions. In other words, the merging performed in the Voronoi diagram depends on the closeness of the data points to each other, i.e., whenever a region contains points which are close to other points in another region; then these two regions will be merged. As the merging is applied depending on the closeness of data points, then each merged region corresponds to having one cluster containing all data points which are close to each other. Therefore, the final Voronoi diagram resulted from applying PROD for a data set is considered as a clustering for this data set, where each merged region in this diagram is considered as a cluster. Furthermore, as PROD is also able to identify outlier data points; it is therefore considered as a partition-based clustering algorithm that can handle outliers. To the best of our knowledge, there is currently no approach which can handle both things at the same time. Moreover, since PROD is considered as a standalone clustering algorithm, the sum of squared errors of these clusters can be computed and the performance of PROD can be evaluated accordingly.

Moreover, PROD produces a prediction for the centroids that can be used instead of the randomly initiated centroids by the K-Means algorithm. More clearly, as each region in the final Voronoi diagram is considered as a cluster; PROD computes the centroid of each region, and provides these computed centroids to be used as the initial centroids in the K-Means algorithm.

In order to perform this evaluation; the following is performed for each of the twenty-six data sets: (1) PROD is applied and the SSE of the resulted clusters is computed, (2) the original K-Means algorithm is applied with a randomly initiated centroids and the SSE is computed, and

(3) the original K-Means algorithm is applied with the predicted centroids as the initial centroids and the SSE is computed. As applying K-Means with randomly initiated centroids may lead to different clustering result each time; therefore, the second step is performed five times, then the median value of the five SSEs is taken.

Table 8 presents, for each data set, the value of the computed SSE for each of the three described steps. In order to do the comparison between PROD, the K-Means with random centroids, and K-Means with the predicted centroids, the median SSE is computed for each of them. The last row in Table 8 shows the computed median SSE for each of the three compared choices.

| Data Set | Sum of Squared Errors (SSE) | | |
|---|---|---|---|
| | PROD | K-Means with Random Centroids (median of 5 runs) | K-Means with Predicted Centroids |
| **4C** | 2.323,48 | 10.583,23 | 11.278,68 |
| **20C** | 1.098,35 | 2.912,83 | 1.895,35 |
| **Aggregation** | 3.999,76 | 19.690,13 | 16.598,23 |
| **Compound** | 1.338,46 | 7.732,68 | 7.732,68 |
| **Curet0** | 925,08 | 347,97 | 347,97 |
| **Curet1** | 542,50 | 211,92 | 266,64 |
| **D31** | 8.055,07 | 19.284,59 | 18.962,21 |
| **Diamond2** | 370,13 | 229,97 | 1.539,30 |
| **Diamond9** | 4.111,02 | 4.360,98 | 4.375,33 |
| **DS** | 433,38 | 642,83 | 410,57 |
| **Elliptical** | 1.335,04 | 4.739,05 | 3.857,83 |
| **Flame** | 1.038,24 | 5.187,11 | 5.187,11 |
| **Fourty** | 622,61 | 3.120,61 | 499,09 |
| **Jain** | 2.491,62 | 22.209,25 | 22.209,25 |
| **Long1** | 610,87 | 637,08 | 1.080,50 |
| **Longsquare** | 1.844,10 | 14.894,05 | 6.637,37 |
| **Lsun** | 310,64 | 593.049,60 | 593.049,60 |
| **R15** | 526,82 | 795,04 | 364,84 |
| **Shapes4** | 4.312,85 | 20.434,32 | 20.434,32 |
| **Shapes9** | 9.210,21 | 223.603,76 | 52.654,11 |
| **Spherical4** | 590,55 | 3.279,74 | 1.030,96 |
| **Spherical6** | 364,11 | 1.351,05 | 543,17 |
| **Triangle** | 1.444,54 | 53.903,15 | 7.035,54 |
| **Xclara** | 33.323,28 | 611.605,88 | 611.605,88 |
| **Zelnik4** | 17,16 | 34,49 | 17,07 |
| **Zelnik5** | 61,62 | 8,09 | 9,73 |
| **MEDIAN SSE** | 1.068,30 | 4.550,02 | 4.116,58 |

*Table 8. Sum of Squared Errors of the Resulted Clusters*

Through inspecting the last row in Table 8, it is noticeable that the median SSE of K-Means with predicted centroids is slightly lower than the median SSE of K-Means with random centroids. The latter means that the resulted clusters from applying K-Means with the predicted centroids as initial centroids is slightly better than the produced clusters by K-Means with randomly initiated centroids. Therefore, the predicted centroids slightly contribute in enhancing the performance of the original K-Means algorithm through reducing the SSE of the resulted clusters.

On the other hand, the overall median SSE of PROD is much lower than the median SSE of both K-Means with random centroids and K-Means with predicted centroids. More precisely, the produced clusters by PROD are better, with respect to the performance of the clustering, than the clusters produced by the original K-Means algorithm.

One main difference between the clustering made by PROD and the clustering made by the original K-Means algorithm is that K-Means is not able to identify outliers, i.e., all data points will be assigned to one of the clusters, even data points that should be considered as outliers. This fact leads to a higher SSE for some data sets as the distance between the outlier data point and the centroid of the cluster to which it is assigned will be relatively large.

| Data Set | Relative Amount of Outliers | Outliers Percentage |
|---|---|---|
| **4C** | 61/876 | 7% |
| **20C** | 100/1517 | 6.6% |
| **Aggregation** | 0/788 | 0% |
| **Compound** | 28/399 | 7% |
| **Curet0** | 22/2000 | 1% |
| **Curet1** | 182/2000 | 9% |
| **D31** | 88/3100 | 2.8% |
| **Dartboard** | 0/1000 | 0% |
| **Diamond2** | 0/667 | 0% |
| **Diamond9** | 22/3000 | 0.7% |
| **DS** | 64/850 | 7.5% |
| **Elliptical** | 0/500 | 0% |
| **Flame** | 2/240 | 0.8% |
| **Fourty** | 7/1000 | 0.7% |
| **Jain** | 63/373 | 17% |
| **Long1** | 120/1000 | 12% |
| **Longsquare** | 71/900 | 7.9% |
| **Lsun** | 10/400 | 2.5% |
| **R15** | 12/600 | 2% |
| **Shapes4** | 9/1261 | 0.7% |
| **Shapes9** | 74/2990 | 2.5% |
| **Smile1** | 118/1000 | 11.8 % |
| **Spherical4** | 1/400 | 0.3% |
| **Spherical6** | 6/300 | 2% |
| **Triangle** | 81/1000 | 8.1% |
| **Xclara** | 178/3000 | 5.9% |
| **Zelnik4** | 134/622 | 21.5% |
| **Zelnik5** | 3/512 | 0.6% |

*Table 9. Relative Amount and Percentage of Identified Outliers.*

On the other hand, PROD is able to identify outliers depending on the distance between data points. More clearly, the distance between an outlier data point and any other data point is relatively larger than the distance between other data points that are close to each other. Therefore, through applying all the steps of PROD; these outlier data points will remain *alone* in some regions in the final Voronoi diagram, i.e., we end up with some regions that contains relatively small number of points. Consequently, since the approach sets a threshold on the number of data points contained in a region in order to be considered as a cluster; these regions with the relatively small number of points will be considered as outliers. These outlier data points will not be taken into consideration when computing the SSE for the resulted clusters, which leads to a lower SSE for some data sets. Table 9 shows the relative amount of outliers identified by PROD as well as the percentage of these outliers.

Through inspecting Table 9, it is noticeable that the amount of outlier data points, for most data sets, is reasonable. However, PROD identified, for some data points, relatively large amount of outliers. For instance, regarding the Zelnik4 data set, PROD identified 134 outlier data points out of 622 points, which means that approximately 22% of the data set was identified as outliers.

However, the decision whether this amount of identified outlier data points is reasonable or not strongly depends on the data set itself. For instance, Figure 30a shows the Zelnik4 data set. Through looking to the figure of this data set, it obviously contains relatively large amount of outlier data points. Therefore, the fact that PROD identified 134 outlier data points out of 622 points is reasonable for this data set. Figure 30b illustrates the clusters and the identified outlier data points by PROD, such that these outlier data points are shown as black data points in this figure.

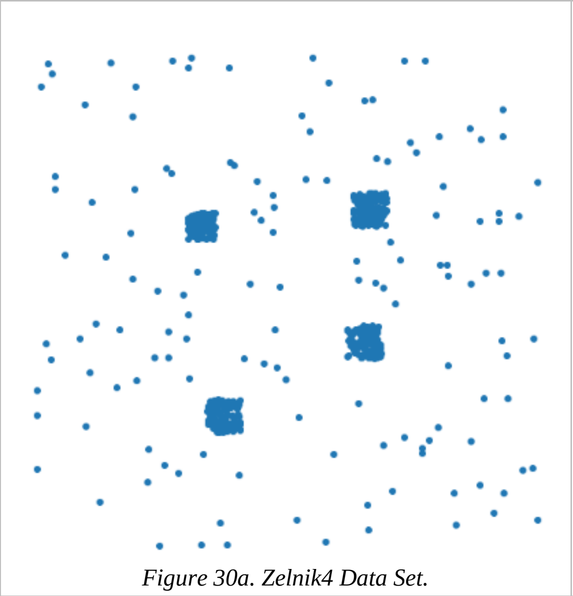


*Figure 30a. Zelnik4 Data Set.*

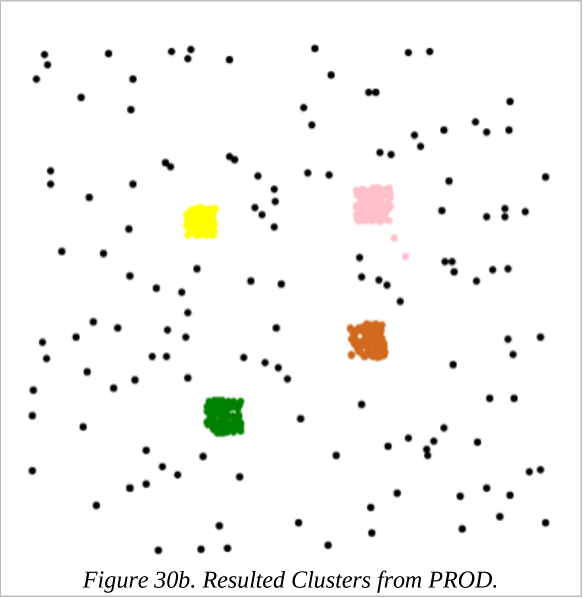


*Figure 30b. Resulted Clusters from PROD.*

*Figure 30. Applying PROD for Zelnik4 Data Set.*

#### *6.4.1.1 Time Evaluation*

An important step in evaluating the performance of an approach is to evaluate the overall time required to do the prediction using this approach. Therefore, this sub-part evaluates the time required by PROD to perform all steps and produce the final prediction, and compares it to the time required by each of the other three approaches: Improved K-Means, Silhouette Method, and Elbow Method. Some of these approaches, due to its high complexity, require huge amount of time to perform the prediction for some data sets. Hence, a threshold for the execution time is set to 30 minutes, i.e., whenever an approach runs for more than 30 minutes and did not finish; the execution will be stopped and the prediction of the *k* will not be completed.

Similar to measuring the runtime of any step in PROD, the whole prediction approach will be applied five times for each data set, then the median value is computed for this data set. Additionally, each of the other previous approaches will also be applied five times for each data set in order to compute the median value for this data set.

Table 10 shows, for each of the twenty-six data sets, the median time required by each of the four approaches in order to perform the overall prediction. More clearly, the measured time here is the runtime of the whole approach, i.e., of the whole pipeline. In case an approach, for a specific data set, took more than 30 minutes in all the 5 runs; then the time is specified as '>30 min'.

| **Data Set** | **Time** | | | |
|---|---|---|---|---|
| | **PROD** (median of 5 runs) | **Improved K-Means** (median of 5 runs) | **Silhouette Method** (median of 5 runs) | **Elbow Method** (median of 5 runs) |
| **4C** | 1.05 s | 23 s | 14 min 50 s | 52 s |
| **20C** | 2.67 s | 1 min 4 s | >30 min | 3 min 50 s |
| **Aggregation** | 0.88 s | 17 s | 5 min 15 s | 29 s |
| **Compound** | 0.42 s | 7 s | 45 s | 12 s |
| **Curet0** | 6.75 s | 3 min 2 s | >30 min | 10 min 5 s |
| **Curet1** | 6.23 s | 2 min 7 s | >30 min | 11 min 30 s |
| **D31** | 15.16 s | 6 min 30 s | >30 min | >30 min |
| **Diamond2** | 0.8 s | 14 s | 4 min 31 s | 35 s |
| **Diamond9** | 17.31 s | 6 min 34 s | >30 min | 30 min |
| **DS** | 1.06 s | 20 s | 8 min 45 s | 41 s |
| **Elliptical** | 0.52 s | 8 s | 1 min 31 s | 12 s |
| **Flame** | 0.23 s | 5 s | 14 s | 6 s |
| **Fourty** | 1.24 s | 28 s | 16 min 40 s | 5 s |
| **Jain** | 0.36 s | 6 s | 38 s | 7 s |
| **Long1** | 1.23 s | 30 s | 15 min 15 s | 1 min 15 s |
| **Longsquare** | 1.11 s | 1 min 5 s | 10 min 5 s | 50 s |
| **Lsun** | 0.42 s | 8 s | 51 s | 10 s |
| **R15** | 0.64 s | 12 s | 2 min 45 s | 13 s |
| **Shapes4** | 1.98 s | 49 s | 29 min 58 s | 3 min 32 s |
| **Shapes9** | 12.52 s | 5 min 45 s | 5 min 50 s | 25 s |
| **Spherical4** | 0.42 s | 20 s | 54 s | 2 min 43 s |
| **Spherical6** | 0.28 s | 8 s | 30 s | 7 s |
| **Triangle** | 1.3 s | 27 s | 14 min 56 s | 1 min 22 s |
| **Xclara** | 12.3 s | 7 min 30 s | >30 min | >30 min |
| **Zelnik4** | 0.74 s | 19 s | 6 min 54 s | 24 s |
| **Zelnik5** | 0.55 s | 17 s | 3 min 51 s | 27 s |
| **AVERAGE Runtime** | **3.39 s** | 89.81 s | **749.92 s** | 300.46 s |

*Table 10. Time Required by PROD and the Other Three Approaches*

In order to compare the four approaches with respect to the runtime; the average runtime for each of these approaches was computed as shown in the last row in Table 10. In case the execution time of an approach for a specific data set exceeds the threshold, i.e., exceeds 30 minutes, it will be included into the average computation but with a value equals to exactly 30 minutes. For instance, when applying the Silhouette method for the 20C data set; the execution time exceeded 30 minutes, therefore, when computing the average runtime for this approach, the execution time with regard to the 20C data set was considered as exactly 30 minutes.

Through inspecting the last row in Table 10, it is noticeable that PROD requires for all twenty-six data sets the least time, among all approaches, for performing the prediction of number of clusters for all the twenty-six data sets. More clearly, the number of clusters for each data set can be predicted within only few seconds using PROD which is much faster than the prediction made by the other three approaches.

Regarding Silhouette method and Elbow method, the average runtime of each of these two approaches is far away from the average runtime of PROD, even though while computing the average runtime, the best case for each of them was taken, which is considering the runtime that exceeds 30 minutes as exactly 30 minutes. Moreover, even the best existing approach which is Improved K-Means, has an average runtime equals to 89.81 seconds, whereas, the average runtime of PROD is only 3.39 seconds. More precisely, PROD is faster by a factor of 26.5 than the best existing one. This fact clearly show that PROD is suitable for big data.

##### *6.4.1.2 Number of Iterations Evaluation*

The last step in the evaluation, is to evaluate the number of iterations performed by the original K-Means algorithm when it is applied with *k* equals to the predicted *k* by PROD. More specifically, for evaluating the number of iterations; two steps are followed for each data set. First, the original K-Means algorithm is applied with randomly initiated centroids for five runs, i.e., the algorithm was executed 5 times and number of iterations for each run was calculated, then the median number of iterations is chosen. Second, the K-Means algorithm is applied with the predicted centroids which are produced by PROD. In all cases, the K-Means is applied, for each data set, with the predicted number of clusters by PROD.

Table 11 shows, for each of the twenty-six data sets, the median number of iterations from applying K-Means with random centroids for 5 runs, as well as the number of iterations from using K-Means with the predicted centroids. The last row in the table shows the computed median of all the values respectively. As number of iterations should be an integer, the median is rounded up to the nearest integer, if necessary. More specifically, as the median number of iterations for applying K-Means with the predicted centroids is 4.5, then this number will be rounded up to 5.

Clearly, through inspecting Table 11, PROD does not require performing multiple iterations. Moreover, the number of iterations required by K-Means with the predicted centroids is less than the number of iterations required by K-Means with randomly initiated centroids. More specifically, for twenty-three out of twenty-six data sets, the number of iterations required by K-Means was reduced when using the predicted centroids.

Regarding the three remaining data sets: Curet0, D31, and Flame data sets; the number of iterations either remained the same or was increased. More clearly, the predicted centroids for two of these three data sets, which are Curet0 and D31, lead to increase the number of iterations required by K-Means instead of reducing this number. Furthermore, for the last remaining data set, which is the Flame data set, the number of iterations remained the same.

| **Data Set** | **Number of Iterations** | | |
|---|---|---|---|
| | **PROD** | **K-Means with Random Centroids (median of 5 runs)** | **K-Means with Predicted Centroids** |
| **4C** | - | 6 | 3 |
| **20C** | - | 11 | 3 |
| **Aggregation** | - | 17 | 9 |
| **Compound** | - | 10 | 2 |
| **Curet0** | - | 12 | 39 |
| **Curet1** | - | 30 | 23 |
| **D31** | - | 21 | 48 |
| **Diamond2** | - | 3 | 2 |
| **Diamond9** | - | 26 | 14 |
| **DS** | - | 22 | 5 |
| **Elliptical** | - | 21 | 15 |
| **Flame** | - | 0 | 0 |
| **Fourty** | - | 15 | 2 |
| **Jain** | - | 10 | 7 |
| **Long1** | - | 10 | 2 |
| **Longsquare** | - | 14 | 10 |
| **Lsun** | - | 11 | 10 |
| **R15** | - | 14 | 10 |
| **Shapes4** | - | 8 | 3 |
| **Shapes9** | - | 10 | 2 |
| **Spherical4** | - | 12 | 2 |
| **Spherical6** | - | 8 | 2 |
| **Triangle** | - | 17 | 5 |
| **Xclara** | - | 12 | 2 |
| **Zelnik4** | - | 8 | 4 |
| **Zelnik5** | - | 18 | 7 |
| **MEDIAN Number of Iterations** | - | 12 | 4,5 |

*Table 11. Number of Iterations Performed.*

In order to clearly illustrate the reduction in number of iterations when using the predicted centroids, the difference between number of iterations required by K-Means with random centroids and K-Means with predicted centroids is calculated. Since Curet0 and D31 data sets are considered as outliers here, the difference in number of iterations will not be computed for them. Table 12 shows, for each of the remaining twenty-four data sets, this calculated difference.

The last row in Table 12 shows the average number of saved iterations over the twenty-four data sets which is equal to 7.04. More precisely, using the predicted centroids as initial centroids when applying K-Means lead to approximately saving 7 iterations in average. Consequently, the produced centroids by PROD enhance the performance of the original K-Means algorithm through reducing the required number of iterations. This fact is of importance especially when having huge amount of data, which means that PROD helps in enhancing the performance of K-Means algorithm for big data.

| Data Set | Number of Saved Iterations |
|---|---|
| **4C** | 3 |
| **20C** | 8 |
| **Aggregation** | 8 |
| **Compound** | 8 |
| **Curet1** | 7 |
| **Diamond2** | 1 |
| **Diamond9** | 12 |
| **DS** | 17 |
| **Elliptical** | 6 |
| **Flame** | 0 |
| **Fourty** | 13 |
| **Jain** | 3 |
| **Long1** | 8 |
| **Longsquare** | 4 |
| **Lsun** | 1 |
| **R15** | 4 |
| **Shapes4** | 5 |
| **Shapes9** | 8 |
| **Spherical4** | 10 |
| **Spherical6** | 6 |
| **Triangle** | 12 |
| **Xclara** | 10 |
| **Zelnik4** | 4 |
| **Zelnik5** | 11 |
| **AVERAGE Number of Saved Iterations** | 7.04 |

*Table 12. Number of Saved Iterations.*

### 6.4.2 Evaluation of Predicting DBSCAN and OPTICS Parameters

The second part of evaluating PROD is about evaluating the process of predicting the input parameters of the two density-based algorithms: DBSCAN and OPTICS. As the prediction of *MinPts* and *ε* mainly depends on the actual regions in the Voronoi diagram which are considered as clusters; hence, the outlier regions in the diagram should be identified and then the prediction can be performed. More clearly, the process of predicting *MinPts* and *ε* can be divided into two main steps: (1) identifying the outlier regions, and (2) performing the prediction taking into consideration all regions in the Voronoi diagram except the identified outlier regions. Twelve different approaches were tested in order to perform these two steps.

#### *6.4.2.1 Identify Outlier Regions*

In order to perform the first step, which is identifying the outlier region, two different approaches were implemented. The first approach identifies outlier regions through setting a threshold equals to a specific percent of the maximum number of points in a region. More clearly, the maximum number of data points in a region *MaxRegion* is identified, then each region that contains number of points less or equal a specific percent of this maximum number is considered as an outlier region.

The second approach for identifying outlier regions works similarly to the idea of identifying the *elbow* in a data set. More clearly, the elbow is identified depending on the number of data points in each region in the final Voronoi diagram. This elbow should represent the number of data points such that any region that has less or equal this number of data points; is considered as outlier. This approach follows these steps: (1) sort ascendingly the regions from the final Voronoi diagram depending on the number of data points inside each region, (2) calculate the difference between number of points in the current region and number of points in the following region, (3) find the maximum difference and consider it as the elbow, and (4) set the threshold as number of points in the region that caused this maximum difference. Finally, each region that contains number of points less or equal this threshold, is considered as an outlier region.

#### *6.4.2.2 Prediction of MinPts and ε*

Regarding the second step which is performing the prediction, the *MinPts* is always predicted as the minimum number of points in a region *MinRegion* in the Voronoi diagram. Whereas, for predicting the *ε*, three different approaches were implemented and tested.

The first approach predicts *ε* as the largest distance between a data point in a region and the site of this region in the Voronoi diagram. On the other hand, the second approach as well as the third approach perform the prediction of *ε* depending on the identified region that has the minimum number of data points *MinRegion*. More specifically, after identifying *MinRegion*, the second approach predicts *ε* as the largest distance between a point in this *MinRegion* and the site of this region. However, the third approach for predicting *ε*, computes the average distance of all distances between all data points in the *MinRegion* and the site of this region. Then, it sets *ε* to be equal to this average distance.

#### *6.4.2.3 Combinations of the Approaches*

Each of the two approaches for identifying outlier regions as well as the three approaches for predicting *ε*; was implemented twice. The first time, these approaches were performed using the final Voronoi diagram which is the result of last step in PROD. The second time, these approaches were performed using the original Voronoi diagram which is produced from the space partitioning step that draws the Voronoi diagram for the sampled data set. More clearly, the original diagram is the diagram of the sampled data set with all data points scattered over this diagram before performing the fine tuning, i.e., before applying any merging.

All the possible combinations of the two approaches of step one and the three approaches of step two are tested and evaluated, hence, there exists six possible combinations. Moreover, each of these six combinations is tested with regard to the final Voronoi diagram as well as to the original Voronoi diagram. Consequently, there exists twelve possible combinations which are implemented and evaluated. In order to do these evaluations, two data sets were used. Since this part of the evaluation concerns evaluating the prediction of the DBSCAN and OPTICS parameters, therefore, the used data sets should demand density-based algorithms. Figure 31a and Figure 31b shows the two used data sets: Dartboard Data Set and Smile1 Data Set respectively, such that these data sets clearly require density-based algorithms. each different color corresponds to a different cluster, i.e., Dartboard Data Set has four different clusters and Smile1 Data Set has also four different clusters.

For each of these two data sets, each of the twelve combinations is tested. More clearly, for each data set, *MinPts* and *ε* were predicted twelve times, such that each time the prediction is done using one different combination. Furthermore, for each prediction of *MinPts* and *ε* of a data set; the original DBSCAN algorithm is applied for this data set with these predicted parameters as input parameters. Moreover, the OPTICS algorithm is also applied for the respective data set with the predicted *MinPts* as an input parameter. Finally, the results of applying DBSCAN and OPTICS are used in order to evaluate the respective combination.

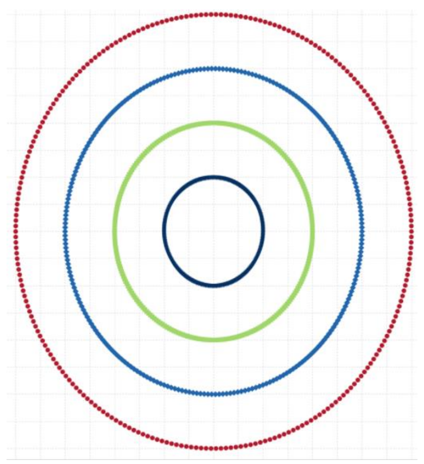


*Figure 31a. Dartboard Data Set*

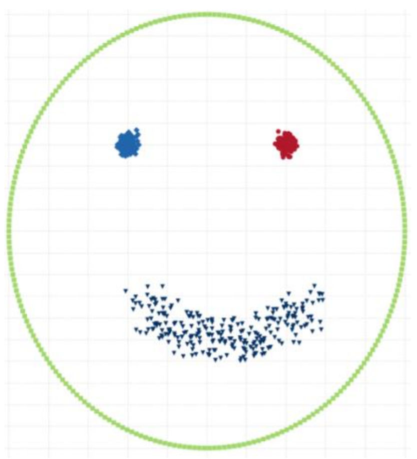


*Figure 31b. Smile1 Data Set*

*Figure 31. The Data Sets Used for The Evaluation of Predicting Density-Based Algorithms Parameters*

***Maximum Difference and Largest Distance***

This combination consists of identifying the outlier regions using the maximum difference approach, and predicting the *ε* as the largest distance in a region in the Voronoi diagram. Table 13 shows the number of resulted clusters from applying DBSCAN and OPTICS for both data sets using the two Voronoi diagrams.

| **Data Set** | **Using Final Voronoi Diagram** | | **Using Original Voronoi Diagram** | |
|---|---|---|---|---|
| | **Number of Clusters from DBSCAN** | **Number of Clusters from OPTICS** | **Number of Clusters from DBSCAN** | **Number of Clusters from OPTICS** |
| **Dartboard** | 1 | 1 | 1 | 1 |
| **Smile1** | 1 | 1 | 2 | 1 |

*Table 13. Number of Resulted Clusters for Both Data Sets Using the Two Diagrams*

First, we start by evaluating this combination against the final Voronoi diagram. Through applying DBSCAN as well as OPTICS with the predicted parameters for the Dartboard data set; the result of each of the two algorithms was only one cluster. More clearly, the predicted *MinPts* and *ε* caused DBSCAN and OPTICS to produce only one cluster that contains all the data points in the Dartboard data set. Similar to Dartboard data set, the predicted parameters for this data set caused DBSCAN and OPTICS to produce also only one cluster containing all data points in the Smile1 data set.

Second, we evaluate this combination against the original Voronoi diagram. Similar to the final Voronoi diagram, applying DBSCAN as well as OPTICS with the predicted parameters for the Dartboard data set resulted only one cluster. On the other hand, the predicted parameters for the Smile1 data set, using this combination on the original Voronoi diagram, caused DBSCAN to produce two different clusters as shown in Figure 32. Clearly, this resulted clustering is not valid as it caused most of the data points to be considered as outliers and therefore not included in the resulted clusters. However, applying OPTICS with the predicted parameters for the Smile1 data set resulted only one cluster for the whole data set.

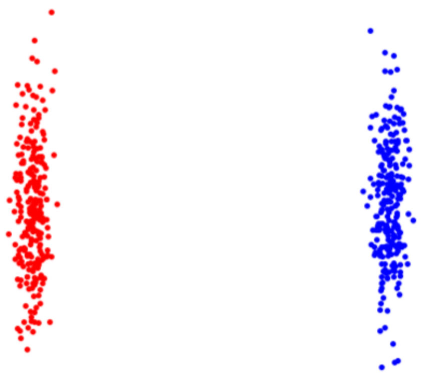


*Figure 32. The Resulted Clustering of applying DBSCAN for Smile1 Data Set Using Original Voronoi Diagram*

Consequently, using the combination of the maximum difference and the largest distance, from both final diagram as well as original diagram, is not valid and cannot be used to predict the parameters of DBSCAN and OPTICS correctly.

### *Maximum Difference and Largest Distance in MinRegion*

The second combination consists of identifying the outlier regions using the maximum difference approach, and predicting the $\varepsilon$ as the largest distance in the region that contains minimum number of points, i.e., in the MinRegion. Table 14 shows the number of resulted clusters from applying DBSCAN and OPTICS for both data sets using the two Voronoi diagrams.

| **Data Set** | **Using Final Voronoi Diagram** | | **Using Original Voronoi Diagram** | |
|---|---|---|---|---|
| | **Number of Clusters from DBSCAN** | **Number of Clusters from OPTICS** | **Number of Clusters from DBSCAN** | **Number of Clusters from OPTICS** |
| **Dartboard** | 1 | 1 | 1 | 1 |
| **Smile1** | 3 | 1 | 2 | 1 |

*Table 14. Number of Resulted Clusters for Both Data Sets Using the Two Diagrams*

Regarding the Dartboard data set, the predicted parameters resulted from this combination using both Voronoi diagrams, caused DBSCAN as well as OPTICS to produce only one cluster containing all the data points in the Dartboard data set.

On the other hand, using this combination to predict the parameters for the Smile1 data set from the final Voronoi diagram, and applying DBSCAN with these predicted parameters lead to produce three different clusters. Figure 33 illustrates the resulted three clusters. Obviously, this clustering is not valid as it has considered some of the data points as outliers which is not true for this data set. However, applying OPTICS with the predicted *MinPts* caused again to produce only one cluster containing all data points.

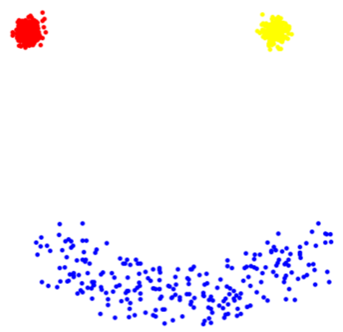


*Figure 33. The Resulted Clustering of applying DBSCAN for Smile1 Data Set Using Final Voronoi Diagram*

Furthermore, using the original Voronoi diagram to perform the prediction for Smile1 data set, caused DBSCAN to produce two different clusters same as shown in Figure 32. However, this prediction caused OPTICS to produce only one cluster for the whole data set.

Consequently, using the combination of the maximum difference and the largest distance in the MinRegion, from both final diagram as well as original diagram, is not valid and cannot be used to predict the parameters of DBSCAN and OPTICS correctly.

***Maximum Difference and Average Distance in MinRegion***

This combination consists of identifying the outlier regions using the maximum difference approach, and predicting $\varepsilon$ as the average distance in the region that contains minimum number of points, i.e., in the MinRegion. Table 15 shows the number of resulted clusters from applying DBSCAN and OPTICS for both data sets using the two Voronoi diagrams.

| **Data Set** | **Using Final Voronoi Diagram** | | **Using Original Voronoi Diagram** | |
|---|---|---|---|---|
| | **Number of Clusters from DBSCAN** | **Number of Clusters from OPTICS** | **Number of Clusters from DBSCAN** | **Number of Clusters from OPTICS** |
| **Dartboard** | 1 | 1 | 1 | 1 |
| **Smile1** | 2 | 1 | 2 | 1 |

*Table 15. Number of Resulted Clusters for Both Data Sets Using the Two Diagrams*

Regarding the Dartboard data set, the predicted parameters resulted from this combination using both Voronoi diagrams, caused DBSCAN as well as OPTICS to produce only one cluster containing all the data points in the Dartboard data set.

Regarding the Smile1 data set, the predicted parameters resulted from using the final Voronoi diagram as well as the predicted parameters resulted from using the original Voronoi diagram lead to the same result for both algorithms. More clearly, using the predicted parameters from the final Voronoi diagram caused DBSCAN to produce two different clusters same as shown in Figure 32. Furthermore, these parameters caused OPTICS to produce only one cluster. Similarly, the predicted parameters from the original Voronoi diagram also caused DBSCAN to produce two different clusters same as in Figure 32, and caused OPTICS to produce only one cluster for the whole data set.

Consequently, using the combination of the maximum difference and the average distance in the MinRegion, from both final diagram as well as original diagram, is not valid and cannot be used to predict the parameters of DBSCAN and OPTICS correctly.

### *MaxRegion Percent and Largest Distance*

The fourth combination consists of identifying the outlier regions using the approach of setting the threshold to be equal to a specific percent of the maximum number of points in a region MaxRegion, and predicting the $\varepsilon$ as the largest distance in a region in the Voronoi diagram. Many percentages were tested but no common percent, which fits for all data sets, could be found. More specifically, for some data sets, setting the threshold as 5% of the maximum number of points lead to correctly identify outlier regions. Whereas, for other data sets, this 5% lead to a wrong identification of the outlier regions, and another percent, e.g. 30%, lead to the correct outlier identification. In order to illustrate this problem, we choose to set the threshold to the 5% of the maximum number of points for both data sets.

Table 16 shows the number of resulted clusters from applying DBSCAN and OPTICS for both data sets using the two Voronoi diagrams.

| **Data Set** | **Using Final Voronoi Diagram** | | **Using Original Voronoi Diagram** | |
|---|---|---|---|---|
| | **Number of Clusters from DBSCAN** | **Number of Clusters from OPTICS** | **Number of Clusters from DBSCAN** | **Number of Clusters from OPTICS** |
| **Dartboard** | 1 | 1 | 1 | 1 |
| **Smile1** | 1 | 1 | 4 | 1 |

*Table 16. Number of Resulted Clusters for Both Data Sets Using the Two Diagrams*

Regarding the Dartboard data set, the predicted parameters resulted from this combination using both Voronoi diagrams, caused DBSCAN as well as OPTICS to produce only one cluster containing all the data points in the Dartboard data set.

Regarding the Smile1 data set, and using the final Voronoi diagram to perform the prediction, also lead DBSCAN and OPTICS to produce only one cluster for the whole data set. On the other hand, using the original Voronoi diagram to predict *MinPts* and $\varepsilon$, and using these two parameters as input for applying DBSCAN, lead to produce the correct four different clusters for this data set as shown in Figure 34. However, these predicted parameters caused OPTICS again to produce only one cluster.

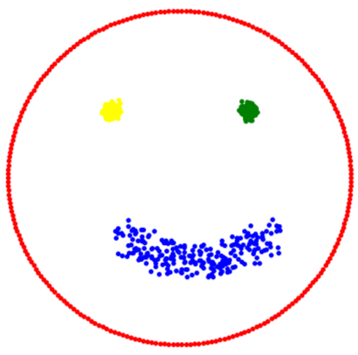


*Figure 34. The Resulted Clustering of applying DBSCAN for Smile1 Data Set Using Original Voronoi Diagram*

The predicted parameters from using the 5% of maximum number as the threshold and the largest distance in a region as the $\varepsilon$, lead to correct clustering only for some data sets. More clearly, for the two used data sets, this percent lead to correct clustering only for the Smile1 data set when using the original Voronoi diagram, whereas, it did not work correctly for the Dartboard data set. Hence, in order to use this combination and as each data set is different from any other data set; the percent should be adjusted for each data set separately, i.e., it cannot be a fully automated. Therefore, using the combination of the specific percent of the maximum number of points and the largest distance in a region, from both final diagram as well as original diagram, is not feasible for an automatic approach for a prediction.

### *MaxRegion Percent and Largest Distance in MinRegion*

This combination consists of identifying the outlier regions using the approach of setting the threshold to be equal to a specific percent of the maximum number of points in a region, and predicting $\varepsilon$ as the largest distance in the region that contains minimum number of points, i.e., in the MinRegion. Table 17 shows the number of resulted clusters from applying DBSCAN and OPTICS for both data sets using the two Voronoi diagrams.

| **Data Set** | **Using Final Voronoi Diagram** | | **Using Original Voronoi Diagram** | |
|---|---|---|---|---|
| | **Number of Clusters from DBSCAN** | **Number of Clusters from OPTICS** | **Number of Clusters from DBSCAN** | **Number of Clusters from OPTICS** |
| **Dartboard** | 1 | 1 | 2 | 1 |
| **Smile1** | 4 | 1 | 4 | 1 |

*Table 17. Number of Resulted Clusters for Both Data Sets Using the Two Diagrams*

Regarding the Dartboard data set, the predicted parameters resulted from this combination using the final Voronoi diagram caused DBSCAN and OPTICS to produce one cluster for the whole data set. However, using the original Voronoi diagram, caused DBSCAN to produce two different clusters as shown in Figure 35. Clearly, this resulted clustering is not valid as it caused non-outlier data points to be considered as outliers and therefore not included in the resulted clusters.

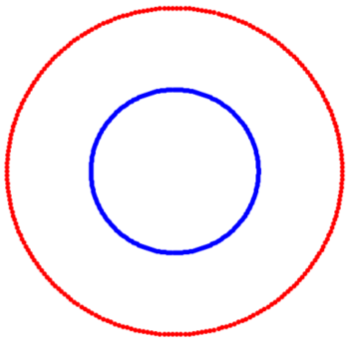


*Figure 35. The Resulted Clustering of applying DBSCAN for Dartboard Data Set Using Original Voronoi Diagram*

With regard to the Smile1 data set, using both the final Voronoi diagram as well as the original Voronoi diagram to predict *MinPts* and $\varepsilon$, and applying DBSCAN with these predicted parameters as input; lead to produce the correct four different clusters for this data set as shown in Figure 34 on page 74. However, these predicted parameters caused OPTICS again to produce only one cluster.

As for this combination there exist no one common percent for setting the threshold for all data sets, therefore, using the combination of the specific percent of the maximum number of points and the largest distance in the MinRegion, from both final diagram as well as original diagram, is not feasible for an automatic approach for a prediction.

### *MaxRegion Percent and Average Distance in MinRegion*

The final combination consists of identifying the outlier regions using the approach of setting the threshold to be equal to a specific percent of the maximum number of points in a region, and predicting the $\varepsilon$ as the average distance in the region that contains minimum number of points, i.e., in the MinRegion. Table 18 shows the number of resulted clusters from applying DBSCAN and OPTICS for both data sets using the two Voronoi diagrams.

| **Data Set** | **Using Final Voronoi Diagram** | | **Using Original Voronoi Diagram** | |
|---|---|---|---|---|
| | **Number of Clusters from DBSCAN** | **Number of Clusters from OPTICS** | **Number of Clusters from DBSCAN** | **Number of Clusters from OPTICS** |
| **Dartboard** | 1 | 1 | 1 | 1 |
| **Smile1** | 4 | 1 | 4 | 1 |

*Table 18. Number of Resulted Clusters for Both Data Sets Using the Two Diagrams*

Regarding the Dartboard data set, the predicted parameters resulted from this combination using both Voronoi diagrams, caused DBSCAN as well as OPTICS to produce only one cluster containing all the data points in the Dartboard data set.

With regard to the Smile1 data set, using both the final Voronoi diagram as well as the original Voronoi diagram to predict *MinPts* and $\varepsilon$, and applying DBSCAN with these predicted parameters as input; lead to produce the correct four different clusters for this data set as shown in Figure 34 on page 74. However, these predicted parameters caused OPTICS again to produce only one cluster.

As for this combination there exist no one common percent for setting the threshold for all data sets, therefore, using the combination of the specific percent of the maximum number of points and the largest distance in the MinRegion, from both final diagram as well as original diagram, is not feasible for an automatic approach for a prediction.

Obviously, none of the twelve different combinations for predicting *MinPts* and $\varepsilon$ worked well. Therefore, none of these twelve combinations can be used to perform the prediction of the parameters of the two density-based algorithms DBSCAN and OPTICS. However, PROD can be modified to achieve solid results for these two algorithms. One approach could be changing the prediction for the parameter *MinPts*. Moreover, another modification could be finding another threshold of identifying outlier data points, which can be used for a better result than the previously tested thresholds.

# 7. Conclusion

Being in the big data era is causing the world to be drown in data, and lacking the useful information as well as the required knowledge. Solving the problem of knowledge lacking requires an analysis of this huge amount of data in order to extract this knowledge. Many reference process models, such as the KDD process [2], exist in order to guide the analytical process which is performed by the user. The most essential step in such analytical processes is the Data Mining step since the patterns, which are interpreted in later steps to deliver the new necessary knowledge, are discovered in this step.

Data Mining makes use of set of mining algorithms which are applied on the data sets in order to generate models which deliver useful information from these data sets [4]. These mining algorithms require a set of input parameters. The decision of which mining algorithm should be applied and which setting of the respective parameters should be chosen is made by the analysts. In order to make the correct decision of the mining algorithm and the parameters setting, the analysts need to have prior technical and domain knowledge respectively. However, having such knowledge in advance is difficult. Some techniques are introduced in order to reduce the need of having the technical knowledge by the analysts [3]. However, overcoming the need of having a prior domain knowledge requires other approaches that enable the analysts to apply the mining algorithm without this prior domain knowledge. More specifically, these approaches provide the user with a good setting of parameters required by the mining algorithm, i.e., they provide predictions for these input parameters.

Clustering is one mining technique that groups the data into set of clusters [8]. K-Means [9], DBSCAN [13, 14, 15], and OPTICS [16] are the most commonly used clustering algorithms. Many previous prediction approaches exist, such that each approach provides prediction for the input parameters of one of these algorithms [15, 16, 19, 20, 23, 24, 25, 26, 27]. However, most of these approaches have one main issue which is the need for applying the original clustering algorithm multiple times in order to perform the prediction [20, 23, 24, 25, 26, 27]. Another issue is the high complexity of these approaches which make them not feasible for big data. Therefore, a new approach, which overcomes these issues and is able to provide a prediction for the input parameters, is required.

In this thesis, the Position-Based Prediction Using Voronoi Diagram (PROD) approach was introduced. This approach produces a prediction for the input parameter for one of the most commonly used clustering algorithm, which is K-Means algorithm. The prediction is made by PROD using Voronoi diagrams [7], which is a space partitioning approach. Moreover, this prediction is performed for two-dimensional data sets, however, it can be easily expanded to multiple dimensions.

In order to produce the prediction; PROD performs the following four main steps: (1) *Sampling* the data set using the new position-based sampling approach, (2) *Space Partitioning* for the space containing the sampled data set using Voronoi diagrams, (3) *Fine Tuning* for the partitioned space using an approach for inspecting and merging the Voronoi regions, and finally (4) *Predicting* the required input parameters using the final produce Voronoi diagram.

The new prediction approach PROD was implemented and evaluated. The evaluation of this approach is mainly divided into two parts: (1) evaluating the ability of PROD to produce a prediction for the input parameter *k* for the K-Means algorithm, and (2) evaluating the ability to predict the input parameters for DBSCAN and OPTICS algorithms as well. In order to perform these evaluations, twenty-eight different data sets were used in all the experiments.

First, evaluating the prediction made by PROD for the parameter *k* which is required by the K-Means algorithm, is again divided into four parts which concerns evaluating: the *accuracy* of the predicted $k$, the *Sum of Squared Errors* of the resulted clusters, the *time* needed to perform the prediction, and the *number of iterations* required by K-Means taking the prediction made by PROD as an input.

On one hand, the evaluation of the accuracy of the predicted *k* and the evaluation of the required time by PROD are both done through comparing PROD with the predictions made by other previous prediction approaches. Three previous prediction approaches, i.e., Improved K-Means, Silhouette Method, and the Elbow Method were chosen in order to perform these two evaluations. The result of these evaluations indicated that PROD is better, with respect to both time and accuracy, than the other three previous approaches. More precisely, PROD produces a prediction which reduces the solution space for searching for the optimal $k$, i.e., the analyst has to only check the range $[k_{pred} - 2 , k_{pred} + 2]$ in order to find the optimal $k$ instead of $[k_{pred} - 4 , k_{pred} + 4]$ for the best previously available solution. Hence, the overall solution space can be shrinked down to 4 possibilities to check instead of previously 8 possibilities.

On the other hand, the evaluation of the Sum of Squared Errors (SSE) is done through comparing the result of PROD to the result of applying K-Means with the predicted $k$ as an input two times: one time with randomly initiated centroids and another time with the predicted centroids as initial centroids. Since the fine tuning step produces a Voronoi diagram where each region is considered as a cluster, therefore, the result of PROD is considered as a clustering for the data set. More precisely, PROD is considered as a partition-based clustering algorithm that is able to handle outliers. Therefore, the SSE of these resulted clusters is computed and compared to the SSE of the resulted clusters from applying the K-Means algorithm. The result of this comparison showed that the resulted clusters by PROD are better with respect to the SSE. This is due to the fact that PROD, in contrast to K-Means, is able to identify outliers and not including them in any of the resulted clusters. More precisely, PROD is able to identify relatively reasonable amount of outlier data points for each data set. Moreover, the predicted centroids, when used as initial centroids by the K-Means algorithm, contributed in enhancing the resulted clusters, i.e., the SSE of the resulted clusters by applying K-Means with the predicted centroids is lower than the SSE of the resulted clusters by applying K-Means with randomly initiated centroids.

Finally, the evaluation of the number of iterations is done through performing a comparison between applying the K-Means with randomly initiated centroids and K-Means with the predicted centroids as initial centroids. This comparison revealed that the predicted centroids lead to a fewer number of iterations required by the K-Means algorithm. More precisely, using the predicted centroids as initial centroids when applying K-Means lead to saving 7 iterations in average, which means better performance of K-Means with respect to runtime.

Second, evaluating the ability of PROD to predict the input parameters for DBSCAN and OPTICS algorithms is done through evaluating twelve different approaches for producing a prediction for *MinPts* and $\varepsilon$. These twelve approaches were implemented and tested. Each produced prediction was tested through applying DBSCAN and OPTICS with the predicted parameters. Unfortunately, the experiments showed that none of these twelve approaches was able to correctly predict these two input parameters. Therefore, PROD cannot be used to predict input parameters of DBSCAN and OPTICS as expected in its current state. Further research has to unveil if any further improvements yield to better results for density-based algorithms.

Eventually, PROD is an approach that can be used either: (1) as a standalone partition-based clustering approach that can handle outlier data points which achieves better results than the original K-Means algorithm with respect to runtime and SSE, or (2) as a prediction approach that is able to predict, for each data set, number of clusters which is required as an input to the K-Means algorithm such that the optimal $k$ lies in the range $[k_{pred} - 2\,, k_{pred} + 2]$. Additionally, when PROD is used as a prediction approach for the amount of clusters, it produces a prediction for the initial centroids to be used by the K-Means algorithm instead of the randomly initiated centroids, such that these predicted centroids enhance the performance of K-Means in terms of the runtime of the algorithm.

Moreover, through comparing the prediction performed by PROD with the prediction made by previously existing approaches, and specifically choosing the best existing approach: Improved K-Means, the results revealed that PROD: (1) predicts more accurate number of clusters, (2) produces clusters with higher quality as provided by SSE, (3) is faster by a factor of 26.5, i.e., has a lower runtime, and (4) saves in average 7 iterations of number of iterations required by K-Means.

# 8. Outlook

The Position-Based Prediction Using Voronoi Diagrams (PROD) approach, which was introduced in this thesis is superior, as shown previously, among the other three prediction approaches. However, PROD can be even more enhanced and extended in future work with respect to multiple aspects.

Currently, PROD can only be performed for two-dimensional data sets. Therefore, one of the extensions that could be applied to this approach is to make it applicable for more than two-dimensional data sets, i.e., for *n*-dimensional data sets. This can be done by mainly adjusting the space partitioning step as well as applying some modifications for all other steps performed by PROD.

Moreover, the performance of PROD can be more enhanced through utilizing the capabilities of the modern parallel and distributed computing systems. More precisely, these distributed systems allow not only higher scalability but also higher performance in terms of better runtime as well. For instance, one important framework that is used for distributed data processing called MapReduce [46, 47]. This framework is for processing large scale data sets in a distributed computer cluster, and it can be used to allow for a distributed processing of PROD. Furthermore, Apache Spark is another important framework which also can be used to allow for a parallel and distributed processing of PROD [48].

Additionally, finding working prediction for DBSCAN and OPTICS is an important aspect on which future work can focus. This can be done through testing another approach which may produce useable results. On one hand, this can be either in the fine tuning step, i.e., through finding and testing another approach for the inspection and merging of the Voronoi regions, or in the prediction step, i.e., through finding another approach for producing the final prediction. On the other hand, this can also be in using another space partitioning algorithm.

Furthermore, another aspect on which future work can focus, is to extend the prediction step made by PROD in order to enable it to produce predictions for input parameters for other mining algorithms. More clearly, there might be mining algorithms, other than DBSCAN and OPTICS, for which the resulted Voronoi diagram can be used to predict their input parameters in addition to predicting the K-Means parameter. Therefore, future work can focus on finding other mining algorithms, e.g., CURE [49] and DENCLUE [50], and evaluating how the Voronoi diagram can be used to predict input parameters for these algorithms.

Additionally, another aspect for which PROD can be enhanced is to use some other space partitioning approaches. More precisely, future work can focus on choosing a space partitioning approach, other than Voronoi diagrams, and evaluate the performance of PROD using the other space partitioning approach. More precisely, since PROD tackled the problem with a visual approach, therefore, future work can focus rather on algorithmic approaches such as KD-Trees [38] or R-Trees [39].

Furthermore, these other space partitioning approaches might work better for the purpose of predicting input parameters for more than one mining algorithm. More clearly, other space partitioning approaches might produce better results that can be used to predict input parameters for multiple mining algorithms in addition to predicting the number of clusters for the K-Means.

Therefore, future work can focus on: (1) making PROD applicable for *n*-dimensional data, (2) distributing PROD in a parallel processing cluster, (3) finding working prediction for DBSCAN and OPTICS, (4) evaluating other space partitioning approaches, and finding how and for which mining algorithms these approaches can be used to perform a correct prediction of the input parameters.

## 9. References

[1] Kudyba, S., & Kwatinetz, M. (2014). Introduction to the Big Data Era. *Big Data, Mining, and Analytics: Components of Strategic Decision Making*, 1.

[2] Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). "The KDD process for extracting useful knowledge from volumes of data," Communications of the ACM (39:11), ACM, pp. 27–34 (doi: 10.1145/240455.240464).

[3] Gartner Press Release. https://www.gartner.com/newsroom/id/3570917.

[4] Han, J., & Kamber, M. (2000). Data Mining: Concepts and Techniques. Data Mining: Concepts and Techniques, 3–26. https://doi.org/10.1002/1521-3773(20010316)40:6<9823::AID-ANIE9823>3.3.CO;2-C.

[5] Garner, S. R. (1995). Weka: The waikato environment for knowledge analysis. *Proceedings of the New Zealand Computer Science Research Students Conference*, 57–64.

[6] Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., and Leyton-Brown, K. (2016). "Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA," *Journal of Machine Learning Research* (17), pp. 1–5.

[7] Aurenhammer, F., & Klein, R. (1998). Voronoi diagrams. In J.-R. Sack and J. Urrutia, editors, Handbook of Computational Geometry, Elsevier Science Publishers B.V. North-Holland, Amsterdam.

[8] Andritsos, P. (2002). Data clustering techniques. Rapport Technique, University of Toronto. Department of Computer Science, 1–34. Retrieved from ftp://ftp.cs.toronto.edu/cs/ftp/pub/reports/csri/443/depth.pdf.

[9] Zhang, Z. (2012). K-means Algorithm, 1–16. Cluster Analysis in Data Mining. Retrieved from http://user.engineering.uiowa.edu/~ie_155/lecture/K-means.pdf.

[10] He, Z. (2006). Farthest-Point Heuristic based Initialization Methods for K-Modes Clustering, (2), 7. Department of Computer Science and Engineering, Harbin Institute of Technology. Retrieved from http://arxiv.org/abs/cs/0610043.

[11] Arthur, D., & Vassilvitskii, S. (2007). K-Means++: The Advantages of Careful Seeding. Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 8, 1027–1025. https://doi.org/10.1145/1283383.1283494.

[12] Huang, Z. (1998). Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values. Data Mining and Knowledge Discovery, 2(3), 283–304. https://doi.org/10.1023/A:1009769707641.

[13] Mining, D. (2011). DBSCAN A Density-Based Spatial Clustering of Application with Noise. Linköpings Universitet - ITN 2011-11-30, 1–8.

[14] Sander, J., Ester, M., Kriegel, H. P. P., & Xu, X. (1998). Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. Institute for Computer Science, University of Munich, 194(1), 169–194. https://doi.org/10.1023/A:1009745219419.

[15] Ester, M., Xu, X., Kriegel, H., & Sander, J. (1996). Density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of The ACM SIGKDD International Conference On Knowledge Discovery and Data Mining, pages, 226–231.

[16] Ankerst, M., Breunig, M. M., Kriegel, H.-P., & Sander, J. (1999). Optics. Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data - SIGMOD ’99, 49–60. https://doi.org/10.1145/304182.304187.

[17] Nelsen, R. (2006). Springer Series in Statistics Springer Series in Statistics. An Intro. https://doi.org/10.1007/978-0-387-98135-2.

[18] Israel, G. (1992). Determining Sample Size. University of Florida Cooperative Extension Services, Institute of Food and Agriculture Sciences, 85(3), 108–113. https://doi.org/10.4039/Ent85108-3.

[19] Birodkar, V., & Edla, D. R. (2014). Enhanced K -Means Clustering Algorithm using A Heuristic Approach. Journal of Information and Computing Science, 9(4), 277–284.

[20] Zhang, H., Yu, H., Li, Y., & Hu, B. (2015). Improved K-means Algorithm Based on the Clustering Reliability Analysis. International Symposium on Computers & Informatics (Isci), 2516–2523.

[21] Huang, S. L. T., & Tan, Y. (2011). Research of clustering algorithm based on k-means. *[J] COMPUTER TECHNOLOGY AND DEVEL*, 21(7), 54-57.

[22] Lei, F. L. X., Xie, K., & Xia, Z. (2008). An efficient clustering algorithm based on local optimality of k-means. *[J] Journal of Software*, 19(7), 1683-1692.

[23] Likas, A., Vlassis, N., & Verbeek, J. (2003). The global k-means clustering algorithm. Pattern Recognition. Pattern Recognition, Elsevier, 36 (2), 451-461. Retrieved from http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V14-45TTJY0-7&_user=6068188&_coverDate=02.

[24] Kodinariya, T. M., & Makwana, P. R. (2013). Review on determining number of Cluster in K-Means Clustering. *International Journal of Advance Research in Computer Science and Management Studies*, *1*(6), 2321–7782.

[25] Yan, M. (2005). Methods of Determining the Number of Clusters in a Data Set and a New Clustering Criterion, 120. Blacksburg, Virginia. Retrieved from https://pdfs.semanticscholar.org/8492/87862e01639378d2301fe5489378df4adf59.pdf.

[26] Caramanis, L. (2013). Lecture 3 — January 22 k-means Algorithm, 1–6. EE 381V: Large Scale Learning. Retrieved from http://users.ece.utexas.edu/~sanghavi/courses/scribed_notes/Lecture_3_Scribe_Notes.pdf.

[27] Pelleg, D., Pelleg, D., Moore, A. W., & Moore, A. W. (2000). X-means: Extending K-means with efficient estimation of the number of clusters. *Proceedings of the Seventeenth International Conference on Machine Learning Table of Contents*, 727–734. https://doi.org/10.1007/3-540-44491-2_3.

[28] Myung, J. (2004). Model Selection Methods. Amsterdam Workshop on Model Selection, Ohio State University. Retrieved from http://faculty.psy.ohio-state.edu/myung/personal/model%20selection%20tutorial.pdf.

[29] Sugar, C., & Gareth, J. (2003). Finding the number of clusters in a data set: An information theoretic approach. *Journal of the American Statistical Association*, *98*, 750–763. https://doi.org/10.1198/016214503000000666.

[30] Fortune, S. (1987). A Sweepline Algorithm for Voronoi Diagrams. *Algorithmica*, *2*(2), 153–174.

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

[32] Shape Sets. https://cs.joensuu.fi/sipu/datasets/.

[33] https://www.uni-marburg.de/fb12/arbeitsgruppen/datenbionik/data?language_sync=1.

[34] A fast Java Implementation Fortune's Voronoi algorithm. http://ageeksnotes.blogspot.de/2010/11/fast-java-implementation-fortunes.html.

[35] Sourceforge: Simple Voronoi Library. https://sourceforge.net/projects/simplevoronoi/.

[36] SPMF an Open-Source Data Mining Library. http://www.philippe-fournier-viger.com/spmf/.

[37] Devulapalli, R., Quist, M., & Carlsson, J. G. (2013). Spatial partitioning algorithms for data visualization, 90170V. Industrial and Systems Engineering, University of Minnesota, Minneapolis, USA. https://doi.org/10.1117/12.2042607.

[38] Dolci, C., Salvini, D., Schrattner, M., & Weibel, R. (2010). Spatial Partitioning and Indexing. Geographic Information Technology Training Alliance. Retrieved from http://www.gitta.info/SpatPartitio/en/text/SpatPartitio.pdf.

[39] Balasubramanian, L., & Sugumaran, M. (2012). A State-of-Art in R-Tree Variants for Spatial Indexing. International Journal of Computer Applications, 42(20), 35–41. https://doi.org/10.5120/5819-8132.

[40] Gold, C. M. (1990). Advantages of the Voronoi Spatial Model. (4 18) 656-3308, 1–10. Research Center in Gtomatic, Laval University, Quebec, Canada. Retrieved from https://pdfs.semanticscholar.org/13db/35133b613038ceb635e7e367749d87a94a9a.pdf.

[41] Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., & Wirth, R. (2000). Crisp-Dm 1.0. CRISP-DM Consortium, 76. Step-by-step data mining guide. https://doi.org/10.1109/ICETET.2008.239.

[42] Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., and Sculley, D. (2017). "Google Vizier: A Service for Black-Box Optimization," in *KDD '17*, p. 10. https://doi.org/10.1145/3097983.3098043.

[43] Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms". *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*, p. 847. https://doi.org/10.1145/2487575.2487629.

[44] Chen, K. K -means Clustering, 1–22. COMP24111 Machine Learning. The University of Manchester. Retrieved from http://syllabus.cs.manchester.ac.uk/ugt/2017/COMP24111/materials/slides/K-means.pdf.

[45] http://lith.me/code/2015/06/08/Nearest-Neighbor-Search-with-KDTree/. Last accessed on 3th of April 2018.

[46] Gillick, D., Faria, A., & DeNero, J. (2006). MapReduce : Distributed Computing for Machine Learning. *Icsiberkeleyedu*, 1–12. https://doi.org/10.1.1.111.9204.

[47] Chen, H., Vasardani, M., & Winter, S. (2017). Geo-referencing Place from Everyday Natural Language Descriptions, 0(0). ACM Transactions on Spatial Algorithms and Systems. Retrieved from https://arxiv.org/pdf/1710.03346.pdf.

[48] da Silva Morais, T. (2015). Survey on Frameworks for Distributed Computing :, 95–105. Proceedings of the 10th Doctoral Symposium in Informatics Engineering - DSIE'15. https://paginas.fe.up.pt/~prodei/dsie15/web/papers/dsie15_submission_7.pdf.

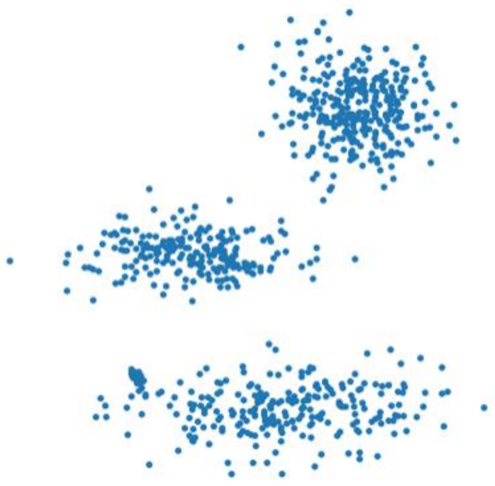
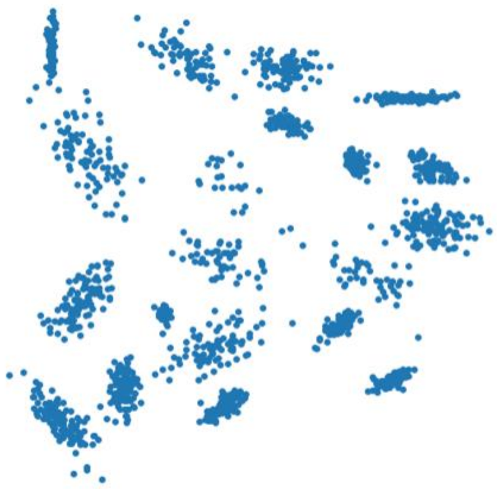
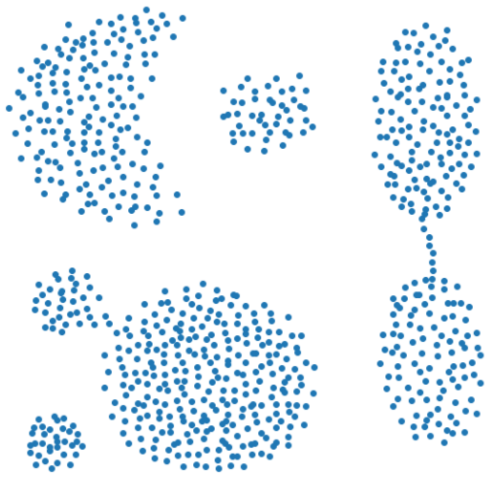
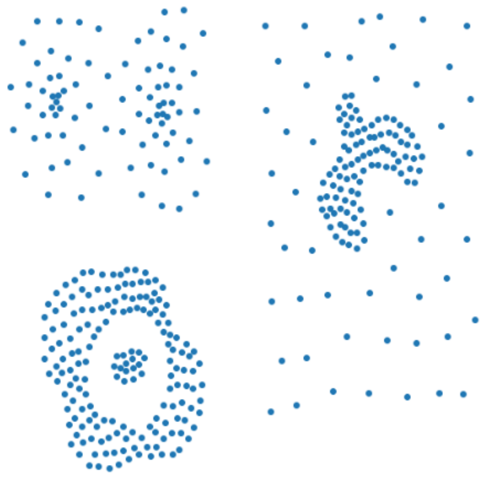
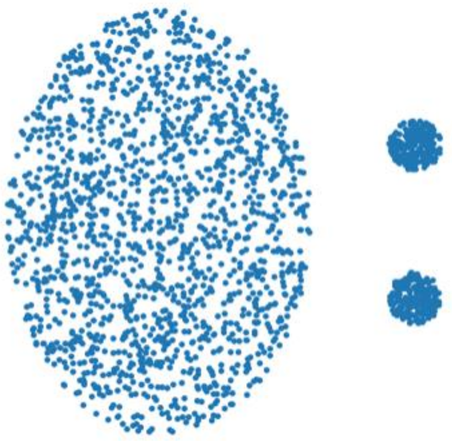
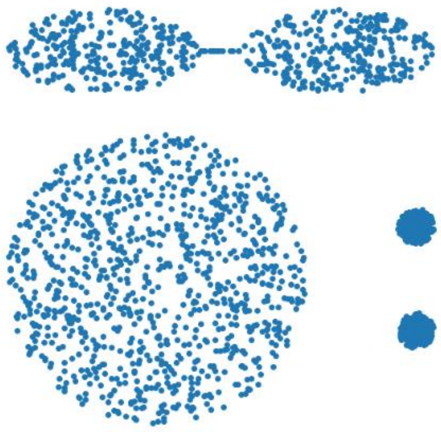
[49] Guha, S., & Rastogi, R. (2001). Cure: an efficient clustering algorithm for large database. Information Systems, 26(1), 35–58. Published by Elsevier Science Ltd. https://pdfs.semanticscholar.org/0788/f8a7feaa84aac29100e668b29ad19ab57e16.pdf.

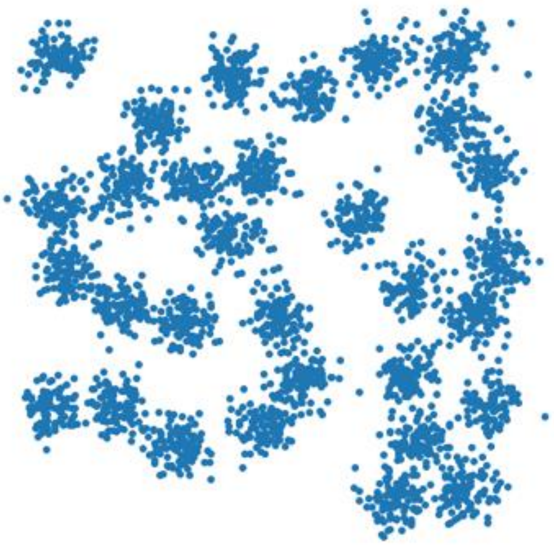
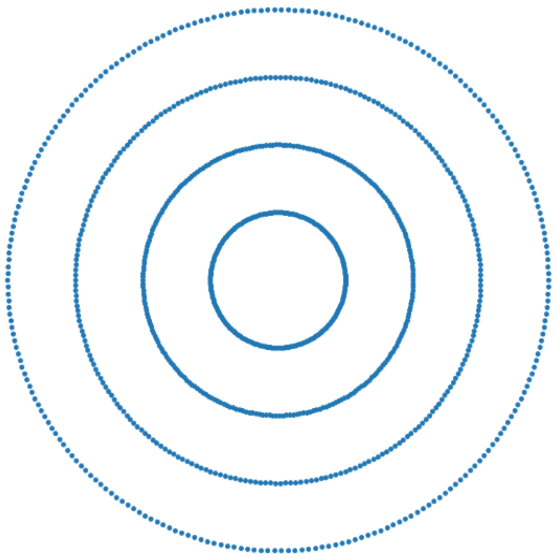
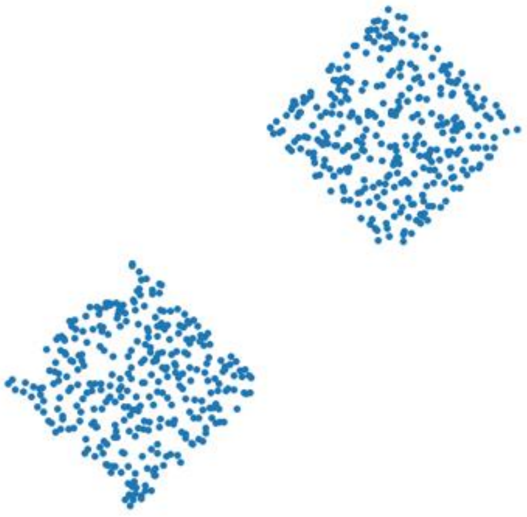
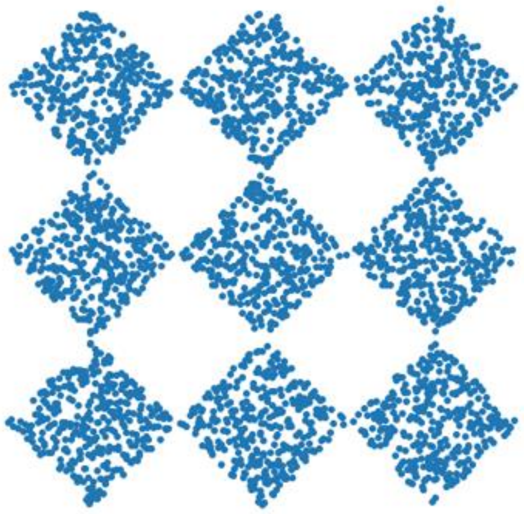
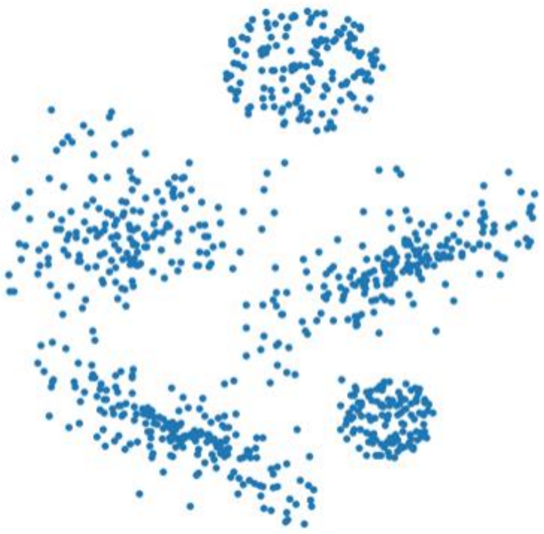
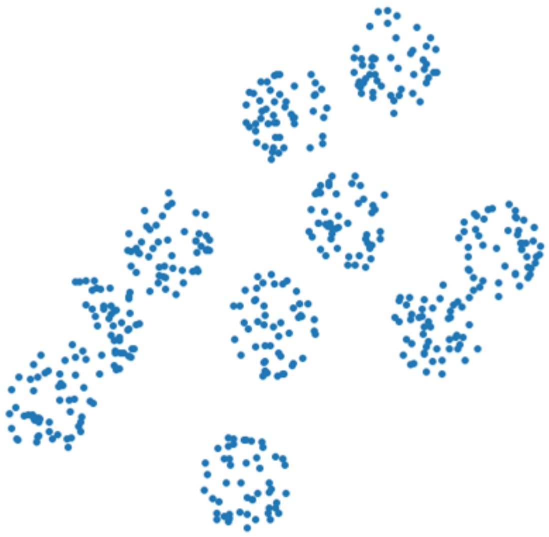
[50] Hinneburg, A., & Gabriel, H. H. (2007). Denclue 2.0: Fast clustering based on kernel density estimation. In Proc. 2007 Int. Conf. Intelligent Data Analysis (IDA'07), 70–80. https://doi.org/10.1007/978-3-540-74825-0_7.
[51] https://github.com/ArlindNocaj/power-voronoi-diagram.
[52] https://github.com/serenaz/voronoi.
[53] https://github.com/aschlosser/voronoi-java.
[54] https://github.com/Rogach/jopenvoronoi.
[55] https://github.com/d3/d3-voronoi.
[56] MOA Machine Learning for Streams. https://moa.cms.waikato.ac.nz/.
[57] ELKI: Environment for Developing KDD-Applications Supported by Index-Structures. https://elki-project.github.io/.
[58] https://deeplearning4j.org/.

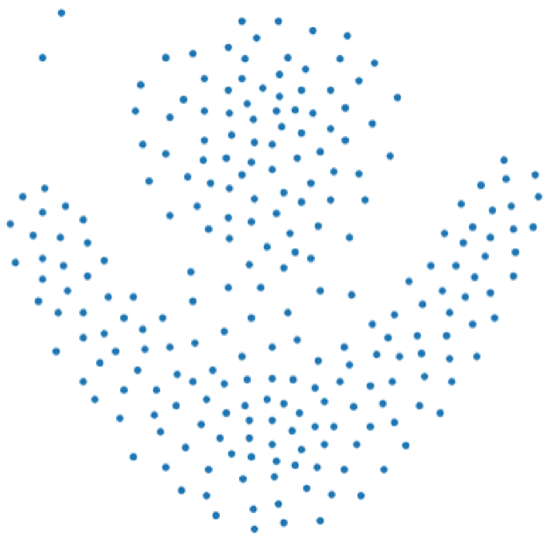
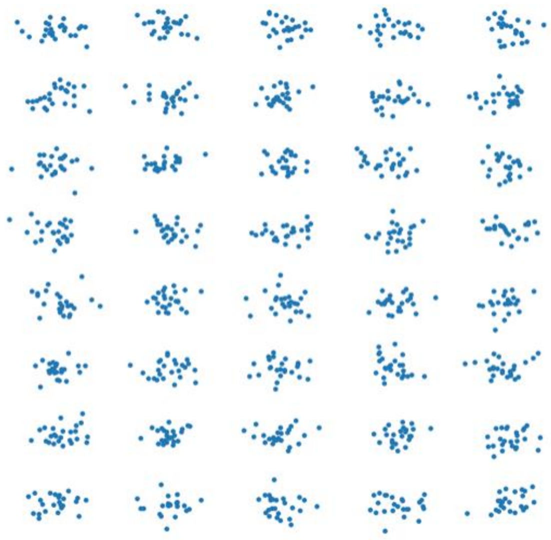
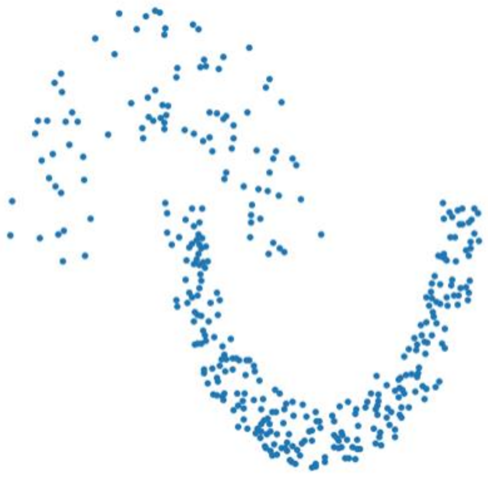
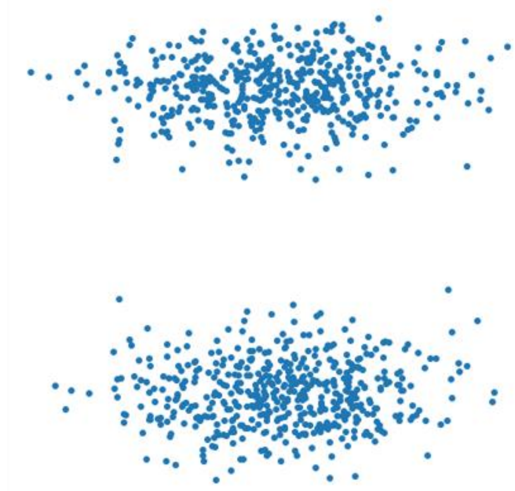
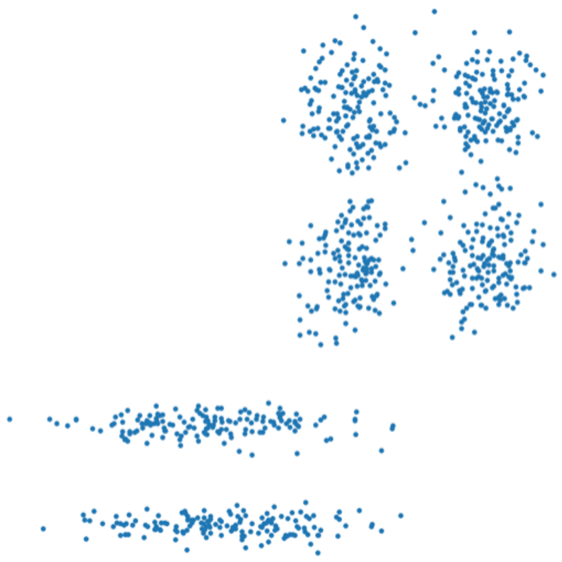
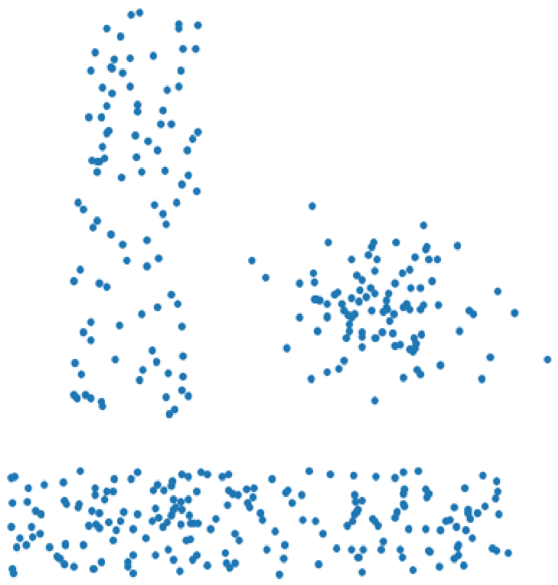
All websites were last accessed on 6th of April 2018.

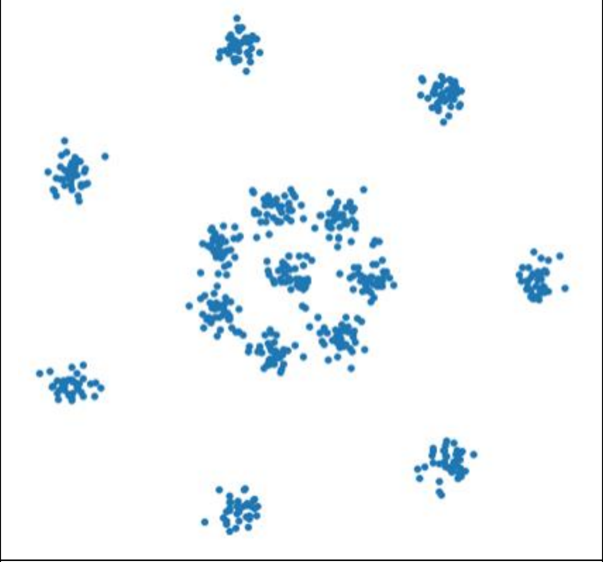
# 10. Appendix

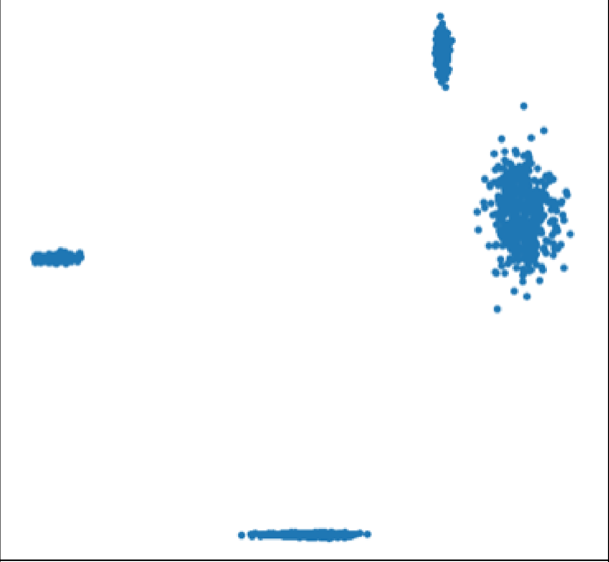
This section provides a screenshot for each of the twenty-eight data sets which are used in all experiments. The screenshot of each data set is provided in the table below.

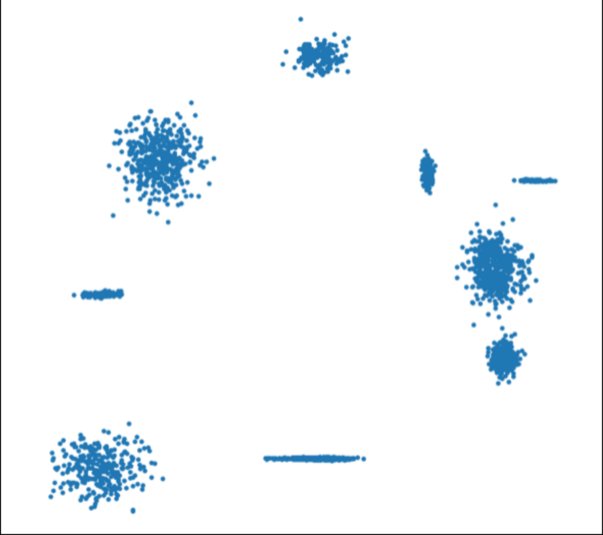
| **4C** | **20C** |
|---|---|
|  |  |
| **Aggregation** | **Compound** |
|  |  |
| **Curet0** | **Curet1** |
|  |  |

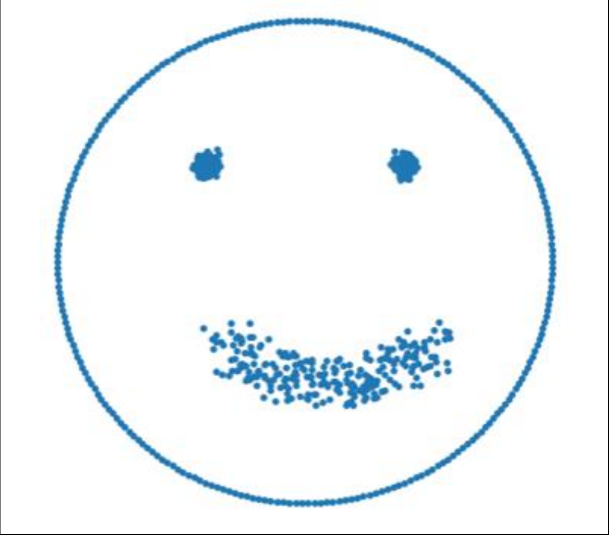
| | |
|---|---|
| **D31**  | **Dartboard**  |
| **Diamond2**  | **Diamond9**  |
| **DS**  | **Elliptical**  |

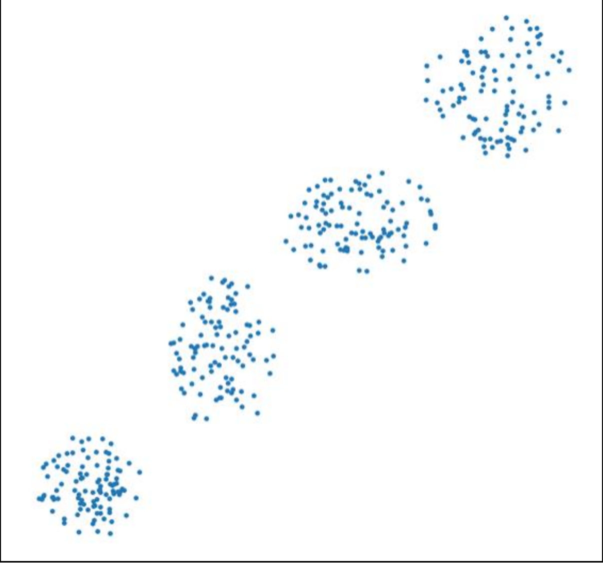
| | |
|---|---|
| **Flame**  | **Fourty**  |
| **Jain**  | **Long1**  |
| **Longsquare**  | **Lsun**  |

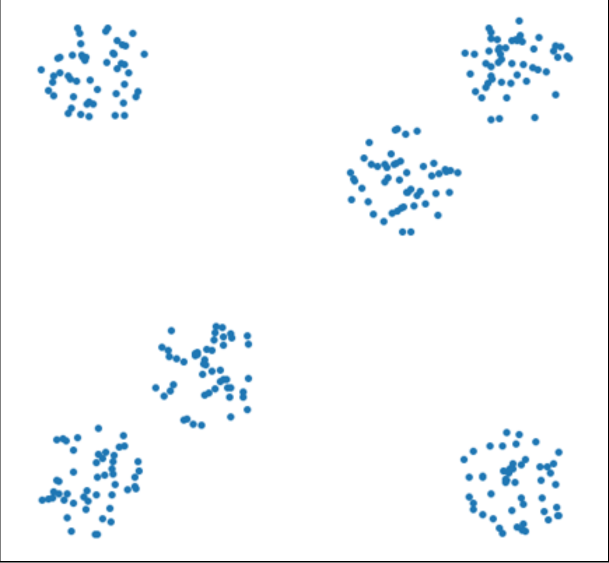

R15

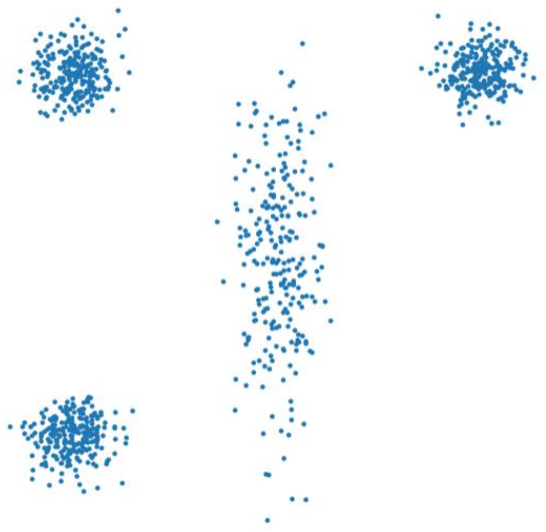
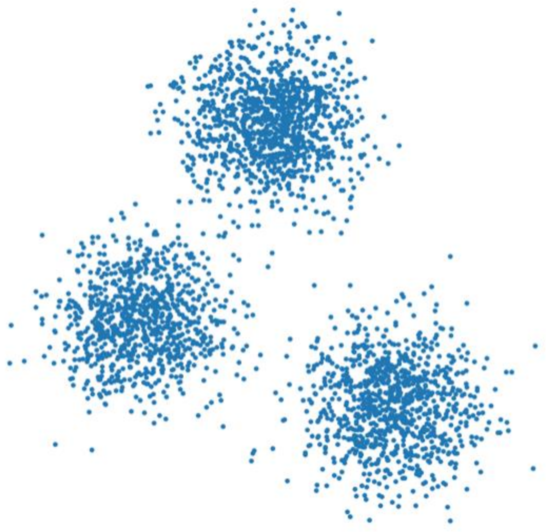
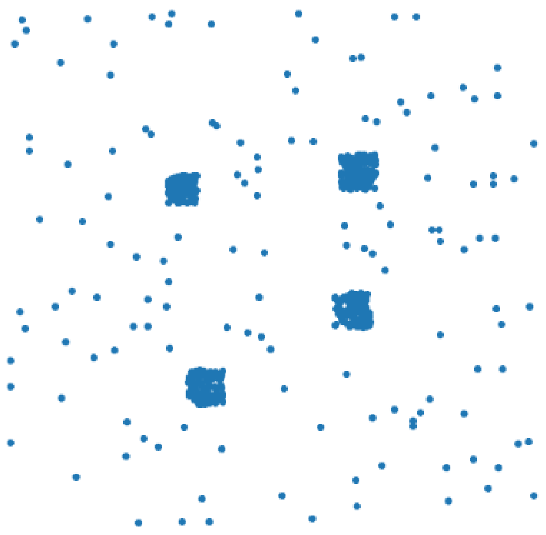
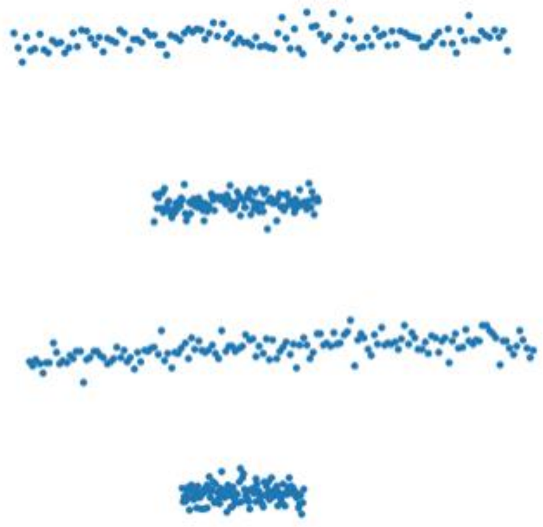

Shapes4


Shapes9


Smile1


Spherical4


Spherical6

| | |
|---|---|
| **Triangle**  | **Xclara**  |
| **Zelnik4**  | **Zelnik5**  |

## Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Date and Signature:

17.04.2018 [illegible]