New partition-based and density-based approaches for improving clustering

Nabil El Malki

▶ **To cite this version:**

Nabil El Malki. New partition-based and density-based approaches for improving clustering. Other [cs.OH]. Université Toulouse le Mirail - Toulouse II, 2021. English. NNT : 2021TOU20007 . tel-03716266v2

HAL Id: tel-03716266

https://theses.hal.science/tel-03716266v2

Submitted on 7 Jul 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 2 - Jean Jaurès

Présentée et soutenue par

Nabil EL MALKI

Le 13 janvier 2021

Nouvelles approches basées sur la partition et la densité pour l'amélioration du regroupement

---

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
IRIT : Institut de Recherche en Informatique de Toulouse

Thèse dirigée par
Olivier TESTE et Franck RAVAT

Jury

Mme Fadila BENTAYEB, Rapporteure
M. Ladjel BELLATRECHE, Rapporteur
Mme Mothe JOSIANE, Examinatrice
M. Olivier TESTE, Directeur de thèse
M. Franck RAVAT, Co-directeur de thèse
M. Pascal PONCELET, Président

Capgemini

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 2 Jean Jaurès (UT2 Jean Jaurès)*

Présentée et soutenue le *13/01/2021* **par :**

Nabil EL MALKI

New Partition-based and Density-based approaches for improving clustering

---

JURY

Fadila BENTAYEB	Professeur, Université de Lyon 2	Rapporteure
Ladjel BELLATRECHE	Professeur, École Nationale Supérieure de Mécanique et d'Aéronautique (ENSMA)	Rapporteur
Pascal Poncelet	Professeur, Université de Montpellier	Président du jury/ Examinateur
Josiane Mothe	Professeur, Université Toulouse 2	Examinatrice
flore mounier	Docteur,	Invité
franck ravat	Professeur, Université Toulouse UT1	Co-directeur
olivier teste	Professeur, Université Toulouse 2	Directeur

École doctorale et spécialité :
EDMITT - Informatique et Télécommunications

Unité de Recherche :
IRIT : Institut de Recherche en Informatique de Toulouse

Directeur(s) de Thèse :
Olivier TESTE et *Franck* RAVAT

Rapporteurs :
Fadila BENTAYEB et *Ladjel* BELLATRECHE

Je dédie cette thèse à la mémoire du cher Jean-François Cesarini pour son soutien actif et son indéfectible confiance en la capacité de réussite... Merci JF pour ton soutien ; Merci JF pour l'être que tu as été...

Remerciements

Je dois une gratitude particulière à mes directeurs de thèse Olivier Teste et Franck Ravat pour leur profonde préoccupation, leurs encouragements perpétuels, leurs excellents conseils, leur coopération aimable et leur soutien incontestable pendant mes travaux de recherche. Aussi, je les remercie chaleureusement pour leur rigueur scientifique, leur encadrement de qualité et la confiance qu'ils ont accordé à ces travaux de thèse. Je tiens à rappeler jusqu'à maintenant qu'ils sont de très loin mes meilleurs encadrants et enseignants que j'ai pu avoir.

Mes sincères remerciements vont également aux membres du jury qui m'ont accordé l'honneur de participer à l'évaluation de ma thèse. Je remercie particulièrement les rapporteurs Mme Fadila Bentayeb et M. Ladjel Bellatreche d'avoir accepté de rapporter soigneusement mon travail. Je remercie également Mme Josiane Mothe et Mr. Pascal Poncelet pour avoir accepté avec grand intérêt d'examiner mes travaux de thèse.

J'exprime mes remerciements pour Robin Cugny pour son aimable collaboration durant mes travaux de recherche, pour son sérieux et sa persévérance incontestables et surtout pour les nuits blanches passées ensemble à rédiger avant les deadlines.

Je remercie chaleuresement Capgemini et ses collaborateurs de m'avoir accueilli. Je remercie également ma responsable de Capgemini Flore Mounier pour son excellent encadrement et son accompagment durant toute la période de ma thèse. Je remercie également Luis de Juan pour ses précieux conseils. Je remercie aussi Daniel Sandrine, responsable du pôle scientifique, pour tout le soutien qu'elle m'a apporté et son souci de me faciliter le déroulement de mes travaux de thèse.

Je remercie chaleuresement l'IRIT et l'équipe SIG de m'avoir accueilli. Je remercie l'ensemble de mes collègues pour leur conseils et aide et particuloèrement Lejeune, Abdelghani, Ramiandrisoa, Neptune, Ullah, Panta sans oublier les anciens Qodseya, Ben Hamadou... et les nouveaux arrivants De casanove, Li, Lazzara, Yewgat...Mes remerciements à l'ensemble du personnel de l'IRIT pour leur disponibilité, leur aide et leur gentillesse.

Je tiens à exprimer ma reconnaissance à Lahouari, Hakima Ait Al Cadi et Abdeljalil pour toute l'expérience qu'ils m'on apportée. Je ne les remercierai jamais assez...

Enfin, je voudrais exprimer ma gratitude à tous les membres de ma famille qui m'a apporté son soutien moral pour mener à bien ce travail de recherche. Je remercie infiniment et profondément mes parents pour l'excellente éducation qu'ils m'ont accordée, leur sacrifice à mon égard, leur souci perpétuel de mon bien être, leur support continu et l'amour sans faille qu'ils m'ont accordé. En effet il n'y a pas de mots pour les remercier comme il se doit. Je remercie mes frères et sœurs pour les bons moments passés ensembles et particulièrement mon grand frère Mohammed qui a été mon soutien depuis de très nombreuses années. Je suis également très reconnaissant aux encouragements et au soutien de ma belle-famille. Je tiens à remercier chaleureusement

Khalid et son épouse Nadia pour leur conseils et enrichissants échanges. J'exprime ma sincère gratitude à mon épouse pour la confiance qu'elle m'a accordée, son inestimable patience, ses précieux encouragements et conseils. Je la remercie pour m'avoir aidé à supporter les difficultés que j'ai rencontrées durant cette thèse. Je la remercie ainsi que notre bel enfant du bonheur qu'ils me font vivre.

Je remercie tous mes chers amis et toute personne ayant laissé une trace positive sur mon parcours. Je leur dédie mes réussites passées et à venir.

Abstract

Clustering is a branch of machine learning consisting in dividing a dataset into several groups, called clusters. Each cluster contains data with similar characteristics. Several clustering approaches exist that differ in complexity and efficiency due to the multitude of clustering applications.

In this thesis, we are mainly interested in centroid-based methods, more specifically k-means and density-based methods. In each approach, we have made contributions that address different problems.

Due to the growth of the amount of data produced by different sources (sensors, social networks, information systems...), it is necessary to design fast algorithms to manage this growth. One of the best-known problems in clustering is the k-means problem. It is considered NP-hard in the number of points and clusters. Lloyd's heuristic has approximated the solution to this problem. This is one of the ten most used methods in data mining because of its algorithmic simplicity. Nevertheless, this iterative heuristic does not propose an optimization strategy that avoids repetitive calculations. Versions based on geometric reasoning have partially addressed this problem. In this manuscript, we proposed a strategy to reduce unnecessary computations in Lloyd's version and the versions based on geometric reasoning. It consists mainly in identifying, by estimation, the stable points, i.e., they no longer contribute to improving the solution during the iterative process of k-means. Thus, calculations related to stable points are avoided.

K-means requires a priori, from users, the value of the number of k clusters. It is necessary for k to be the closest to the ground truth. Otherwise, the result of partitioning is of low quality or even unusable. We proposed Kd-means, an algorithm based on a hierarchical approach. It consists in hierarchizing data in a Kd-tree data structure and then merging sub-groups of points recursively in the bottom-up direction using new inter-group merging criteria that we have developed. These criteria guide the merging process to estimate k closest to real and produce clusters with a more complex shape than sphericity. Through experimentation, Kd-means has clearly shown its superiority over its competitors in execution time, clustering quality and k estimation.

The density-based approach's challenges are the high dimensionality of the points, the difficulty to separate low-density clusters from groups of outliers, and the separation of close clusters of the same density. To address these challenges, we have developed DECWA, a method based on a probabilistic approach. In DECWA, we proposed 1) a strategy of dividing a dataset into sub-groups where each of them follows its probability law; 2) followed by another strategy that merges subgroups, similar in probability law, into final clusters. Experimentally, DECWA, in high-dimensional spaces, produces a good quality clustering compared to its competitors.

Keywords: Clustering, Partition-based clustering, Density-based clustering, Cal-

culation time acceleration

Résumé

Le clustering est une branche de l'apprentissage automatique consistant à diviser un ensemble de données en plusieurs groupes appelés clusters. Chacun des clusters contient des données avec des caractéristiques similaires. Plusieurs approches de clustering existent qui diffèrent en complexité et en efficacité, en raison de la multitude d'applications du clustering.

Dans cette thèse, nous nous intéressons essentiellement aux méthodes basées sur les centroïdes plus spécifiquement les k-moyennes et aux méthodes basées sur la densité. Dans chaque approche, nous avons apporté des contributions qui répondent à des problèmes différents.

En raison de la croissance de la quantité de données produite par différentes sources (capteurs, réseaux sociaux, systèmes d'information...), il est nécessaire de concevoir des algorithmes rapides pour gérer cette croissance. L'un des problèmes les plus connus en clustering est celui des k-moyennes. Il est considéré NP-difficile en nombre de points et de clusters. La solution de ce problème a été approximée par l'heuristique de Lloyd. Celle-ci est l'une des dix méthodes les plus utilisées en fouille de données en raison de sa simplicité algorithmique. Néanmoins, cette heuristique itérative ne propose pas de stratégie d'optimisation qui évite des calculs répétitifs. Des versions basées sur le raisonnement géométrique ont répondu en partie à ce problème. Dans ce manuscrit, nous avons proposé une stratégie visant à réduire les calculs inutiles dans la version de Lloyd ainsi que dans les versions basées sur le raisonnement géométrique. Elle consiste principalement à identifier, par estimation, les points qui sont stables, c'est-à-dire, qui ne contribuent plus à l'amélioration de la solution lors du processus itératif de k-moyennes. Ainsi, les calculs liés aux points stables sont évités.

K-moyennes requiert a priori, de la part des utilisateurs, la valeur du nombre de clusters k. Il est nécessaire que k soit la plus proche de la vérité-terrain, sinon le résultat de partitionnement est de mauvaise qualité voire inutilisable. Nous avons proposé Kd-means, un algorithme basé sur une approche hiérarchique. Elle consiste à hiérarchiser les données dans une structure de données du type Kd-tree puis à fusionner des sous-groupes de points récursivement dans le sens bas-haut via de nouveaux critères de fusion inter-groupes que nous avons développé. Ces critères guident le processus de fusion à estimer k le plus proche du réel et de produire des clusters ayant une forme plus complexe que la sphéricité. À travers les expérimentations, Kd-means a nettement montré sa supériorité, face à ses concurrents, en temps d'exécution, en qualité de clustering et en estimation de k.

Les défis de l'approche des méthodes basées sur la densité sont la grande dimensionnalité des points, la difficulté à séparer les clusters de faible densité des groupes de

points aberrants ainsi que la séparation des clusters proches de même densité. Pour y répondre, nous avons développé DECWA, une méthode basée sur une approche probabiliste. Dans DECWA, nous avons proposé 1) une stratégie de division d'un ensemble de données en sous-groupes où chacun d'eux suit sa loi de probabilité ; 2) suivie d'une autre stratégie qui fusionne des sous-groupes, similaires en loi de probabilité, en clusters finaux. Expérimentalement, DECWA, dans des espaces de grandes dimensions, produit un clustering de qualité par rapport à ses concurrents.

Keywords: Regroupement, Regroupement basé sur les centroides, Regroupement basé sur la densité, Accélération du temps de calcul

Contents

List of Figures

List of Tables

Chapter I

Introduction

Contents

I.1 Research context

I.1.1 Classification

Classification of objects is an important human activity. Indeed, every day, classification is part of our learning process; e.g., the child learns to differentiate man from woman, cat from dog, truck from car etc. In computer science, this classification is automated and constitutes a sub-domain belonging to machine learning (Hastie et al., 2009). The latter cuts across several scientific fields such as biology, speech processing, image processing etc. It has enabled them to gain new fundamental knowledge and thus has enabled them to solve significant problems.

There are three main types of classification. Supervised classification (Alpaydin, 2010) classifies objects while knowing a priori their nature. In a supervised vehicle classification, each object's label or nature is known in advance; e.g., motorbike, car. In this category, the task consists in establishing a model that best represents the data, and then use it to make predictions about labels of new unlabelled objects not yet seen. In unsupervised classification (Berkhin, 2006), objects are not labeled. It consists of finding underlying structures that best represent the nature of different objects. In semi-supervised classification (Zhu, 2008), there are labeled and unlabelled objects. Those that are labeled are used, via strategies, to improve unsupervised classification and to allow a supervised classification alternative when not all objects are labeled. The advantage of this category is that it provides some knowledge in advance about the properties of certain structures that are useful for the first two types of classification. It also reduces the human task of expert manual labeling in the process of supervised classification.

This manuscript focuses on unsupervised classification and, more specifically, its important sub-domain, named *clustering* (Aggarwal and Reddy, 2013).

I.1.2 Clustering

Clustering is one of the most useful data analysis tasks to discover interesting groups and patterns in the underlying data. In clustering, the objective is to partition a set of points (also called objects, observations or simply data points) into groups (clusters) so that the points in a cluster are more similar to each other than the points in different clusters (Figure I.1). Each point is described by attributes that could be called dimensions. To get a better idea of how clustering has been used in practice, below are two different lists of use cases. In the first one clustering is generally a pre-processing tool for other data processing methods (1), while in the second one it is an essential solution in its own right in several application areas (2).

(1) Clustering as an pre-processing tool:

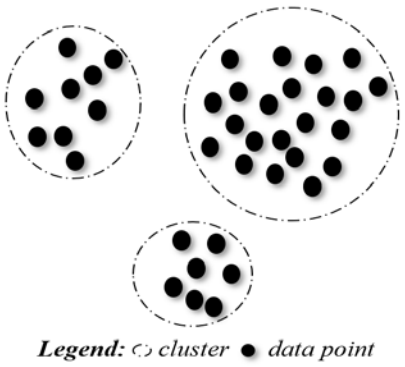


Figure I.1: An example of three clusters, each containing data points

- data summarization (Kleindessner et al., 2019): data compression to give compact and easier to interpret representations. Also, compression leads to a reduction in the use of memory space. Different strategies are implemented more or less explicitly to ensure the level of compromise between the loss of relevant information and memory gain;

- labeling (Trivedi et al., 2015): the user may have to opt, for a specific task, for a supervised classification method. However, this requires the data are labeled. The unsupervised classification is integrated into a pre-processing phase in which it puts the points in groups. From then on, each group is associated with a label. After the end of the pre-processing phase, the data and their associated labels are consumed by the supervised classification method;

- outlier detection (Hodge and Austin, 2004): outliers are a portion of data, which, due to their characteristics, are abnormally different from the majority of other data in a dataset. These outliers can lead, the methods processing the dataset, to biased results. Clustering is called upstream of these methods to clean the outliers from the dataset. However, others tasks are developed specifically to identify outliers that are not considered as errors or biases but rather as phenomena in their own right; e.g., fraudulent transactions. In this case, clustering is not a cleaner but rather a detector of rare phenomena.

(2) Clustering in different application areas:

- genetics (Kiselev et al., 2019). with the massive collection of genomic data due to new biological techniques developed in recent years, different diseases have been categorized according to different gene expression levels. In this way, a disease could be rapidly detected early on via information obtained by the group;

- marketing (Huang et al., 2007). given an e-commerce site, its customers are grouped into segments according to their characteristics. Each segment is a group of customers with similar tastes. These segments are targeted according

to the product that the site is trying to sell with maximum gains. This technique leads to the facilitation or even abstraction of the conversion step from hesitant or uninterested customers to buying customers.

There are many other application domains using clustering, such as health (Thanh et al., 2017), aeronautics (Yang and Villarini, 2019), information retrieval (Chifu et al., 2015), document classification (Amine et al., 2008), databases (Bentayeb and Favre, 2009) etc.

I.2 Research problems

Clustering is a powerful and useful tool for a wide range of use cases. However, it has to face the different challenges to be successful for the areas in which it is called. Despite the development of successful clustering algorithms, they are gradually becoming insufficient for various reasons; i.e. there is a continuous technological advancement, the amount of data collected by modern companies is rapidly increasing, and the data is becoming more and more complex. Clustering is forced to adapt to these developments to meet the needs of data scientists and enterprises. There are two main aspects to consider when evaluating a clustering method: its execution time and data clustering quality. The quality depends essentially on the algorithm used as well as the users' expectations. These two aspects guide the way to meet the challenges. These are diverse and each of them is treated in a specific way. Below are the challenges:

- number of points. Clustering algorithms must produce a result in a reasonable amount of time relative to the number of points contained in the dataset. This is even more true when the method is used as a pre-processing tool for more expensive tasks such as deep learning tasks;
- number of dimensions. This data characteristic is important because it could make a clustering method ineffective when the number of dimensions is relatively large. Indeed, data analysis becomes too complicated because the distances between points tend to become more uniform. In addition, each dimension brings its share of errors and implies biases in the clustering algorithm used.
- data distribution. The way in which points are arranged in a data space brings involved different issues and, consequently, the development of different clustering methods. These issues are also diverse:
  - outliers. datasets are not always clean, especially for real ones. They may contain errors such as input errors, malfunctions of measuring devices, or data collection. In addition to these errors, some data points have particular characteristics compared to other data and errors. They could correspond

to a different nature or even be isolated from the majority of the data. Their identification may be necessary for the success of certain tasks. Therefore, algorithms should be able to work in the presence of outliers, at best, to differentiate between errors and rare phenomena relevant to data scientists;

- cluster imbalance. clusters of points in the dataset can be of different sizes. They are a source of difficulty for algorithms because small clusters can be confused with outliers or even absorbed by large clusters. In this sense, knowledge or detection of certain phenomena could be neglected. Besides, the results may be biased and not reflect the data's natural reality and the way the data was generated;
- cluster forms. Depending on the source that generated the clusters, each cluster of data has a particular form to it or common to other clusters in the same dataset. The form in itself is a significant difficulty in clustering because it implies three assertions: a priori from the launch of a chosen clustering algorithm, the shapes of the clusters are known, not known, or partially known. In the case where the clusters' forms are not known, robust strategies with the fewest possible assumptions about the data must be implemented to best approach the cluster shape's true nature;
- cluster number. many algorithms require the number of clusters to be filled in before the algorithm is run. This information is difficult to obtain and requires a deep knowledge of the dataset. This is often not the case when the algorithm is used as a tool to explore the data before doing specific processing. Giving an incorrect cluster number compared to the data's reality leads to a partitioning of the data that is not of good quality or even not useful.

Generally, to meet these challenges with significantly more performance and efficiency, new alternatives to existing clustering algorithms suffering from specific difficulties should be implemented, or existing algorithms should be revised to make them more robust. This thesis makes contributions to both approaches.

Different clustering paradigms exist (Aggarwal and Reddy, 2013). Their difference is explained by the fact that data is not labeled. So, each paradigm brings its own rules to establish its cluster definition and its strategies for partitioning a dataset. In this thesis, the focus is essentially on the paradigms of centroid-based (Reddy and Vinzamuri, 2018) and density-based clustering methods (Kriegel et al., 2011). We tackle the two paradigms on different issues.

Centroid-based methods consider that a cluster has a centroid that best represents the points of the same cluster. Indirectly they minimize an objective function, i.e., they minimize as much as possible for each cluster the variance between the points and the

centroid (intra-cluster variance). They are based on an iterative approach, i.e., at each iteration, the intermediate solution is improved until convergence, i.e., the maximum possible minimization of the objective function. However, intermediate calculations are costly when a relatively large number of points, dimensions, and clusters are to be processed, especially when the number of iterations is large. Moreover, these methods are sensitive to the parameters set by the users and the characteristics of the data. In practice, these methods are run several times to choose the most optimal result. This makes the use of these methods even more expensive and slower. This problem is discussed in chapter III, section III.2. The contribution focuses on k-means because it is the best-known method in this category. Also, it is widely used in practice and implemented by the majority of known machine learning libraries.

Despite its algorithmic simplicity and wide use, k-means remains nevertheless dependent on input parameters, mainly the number of clusters k. As emphasized above, the k value must be within a range of possible and reasonable values or even a single potential value to produce a partitioning of the dataset that is as close as possible to the field truth. This problem has been addressed by proposing several heuristics. In their operation, they are based on multiple calls of k-means. However, they do not work in certain spatial configurations of the data, i.e, dataset with overlapping clusters, which lead to biased results and very long process time. Moreover, many of these methods estimating k require clusters to be strictly spherical, which is not always the case in reality. This strong apriori assumption about the cluster's shape also leads to a bad estimation of k if the assumption is not respected in the dataset. These problem are addressed in chapter III, section III.3.

Density-based methods adopt another definition of a cluster, unlike centroid based methods. A dense region, i.e., a subspace of data where the points are very close together, represents a potential cluster or part of it (Kriegel et al., 2011). Although these methods have less a priori assumptions about the data than the centroids-based methods, they are sensitive to the spatial distribution of the data. Indeed, they expect at the boundaries between clusters a variation in density. The intensity of this variance depends on the method used. Clusters of the same density with a strong overlap are potentially considered as a single cluster. Another problem is that they might consider a low-density cluster as a group of outliers, i.e., as points that do not provide useful information. However, this low-density cluster may represent a real phenomenon existing in the ground truth. Moreover, density calculations, with very large dimensions, tend to give a clustering far from the ground truth. This problem is called the curse of dimensionality (Donoho, 2000). These problems are dealt with in chapter IV.

I.3 Outline

In this manuscript, three contributions have been made. The first two are grouped in chapter III. They deal with problematics around k-means. The third contribution, presented in chapter IV, deals with a density-based method:

- Acceleration of k-means versions based on triangular inequality in a massive data context. A strategy for accelerating k-means, Sk-means, is proposed. It stores information on the movements of points during the k-means process (standard version and optimized versions based on geometric reasoning). This information allows not to apply useless calculations on certain points. In this way, k-means is accelerated in execution time, especially when running in large datasets;

- Kd-means. Proposal of a clustering method that automatically estimates the number of clusters present in a dataset. Kd-means is based on kd-tree, a hierarchical data structure used to organize data. In Kd-tree, we introduced fusion measures that use geometric properties used in a hierarchical process (bottom-up). These measures guide the aggregations between sub-groups of points to produce, at the process end, final clusters with a more complex shape than those found by k-means and by the mixture of Gaussian models. Kd-means has the advantage of producing quality clustering that is fast compared to competitors;

- Decwa. A clustering method capable of detecting irregularly shaped clusters. It is based on a probabilistic approach as well as optimal transport. It has the advantage of producing a quality clustering carried out on very large dimensions.

Chapter II

Related work

Contents

II.1 Introduction

Given $k \in \mathbf{N}^*$ and $X = \{x_1, ..., x_n\}$ a set of points where x_i is defined in the data space $\mathbb{R}^d$, the clustering aims to partition X into a set of clusters $C = \{C1, ..., C_k\}$ so that the points of a cluster are as similar as possible and the points of different clusters as dissimilar as possible. Note that a dimension corresponds to a characteristic of the data point, e.g., if a point represents a vehicle, its characteristics could be the color, the number of wheels, the registration, etc.

First, we overview the different clustering approaches (section II.2.2) and then identify their main differences. A focus is then essentially put on partition-based methods (section II.3) and density-based methods (section II.5) since our contributions have been made in both approaches. Another section is devoted to optimization strategies (section II.4) for k-means. Even if they can speed up the execution time of methods, these strategies could also improve the quality of partitioning, i.e., obtain even more homogeneous clusters.

II.2 Clustering methods

II.2.1 Introduction

Although clustering methods have to build homogeneous clusters, they do not have the same definition of the cluster nor on how to carry out partitioning because of the unsupervised aspect of these methods. The clustering methods are broadly classified into partition-based, hierarchical-based, density-based and grid-based(Aggarwal and Reddy, 2013). We will briefly discuss these paradigms.

II.2.2 clustering methods taxonomy

Partition-based methods[1]. The aim of this paradigm consists in dividing a dataset X into k clusters according to a particular objective function. Each cluster is represented by a centroid, also called a gravity center or center, which is not necessarily contained in the data set. For example, in the k-means algorithm (Lloyd, 1982b), the centroid is the arithmetic mean of all the cluster points' values. In the k-medoid algorithm (Kaufmann and Rousseeuw, 1987), the centroid is one of the points in the cluster called the medoid.

To produce data partitioning, the methods of this paradigm go through an iterative process. The partitioning is improved at each iteration compared to the previous iteration according to the objective function until convergence. The improvement in an iteration refers to the change of cluster for some points.

[1]This paradigm has another well-known name, centroid-based methods. In this manuscript, both names are used.

Hierarchical methods. In this paradigm, clusters are represented hierarchically, at different granularity levels, through a tree structure called a dendrogram. In this dendrogram, clusters are nested, i.e., they are made up of other smaller clusters. Different strategies exist to recover clusters from the dendrogram, e.g., by cutting the dendrogram at a given height. The recovery of a different set of clusters is possible by cutting the dendrogram at a different height without rebuilding the tree. The methods are divided into two groups:

- divisive: these methods proceed to a recursive dichotomy of the dataset. At each division, one or more clusters are obtained, each of these clusters is then divided and so on;
- agglomerative: these methods recursively agglomerate clusters. At the very beginning, each point is a cluster. The merging is done pair by pair of clusters. Often, a set distance is involved to calculate the similarity between two clusters.

Density-based methods. This group considers that clusters are dense regions of points separated by less dense regions. The latter are assimilated to outliers. A cluster is defined as follows: given a volume in a data space, the more data points there are in this volume, the denser the volume is. Clusters are arbitrary in shape, i.e. they can take practically any shape. Moreover, generally, there are no a priori assumptions about the shape of clusters.

Grid-based methods. Commonly, these methods first quantify the original data space into a finite number of cells. Partitioning operations are performed on the quantized space (grid stucture). The main feature of this approach is its ability to reduce processing time since similar data points fall into the same cell, and each cell is treated as a single point. This makes the algorithms independent of the number of objects in the dataset. However, the clustering quality is compromised by the size of the cells chosen. The definition of quality depends on the method used and the user. Moreover, the uniform grid (i.e. cells with the same size) may not be efficient for very irregular data distributions. Below the common successive instructions of the grid-based methods:

- Define a set of grid cells
- Calculate cell density
- Eliminate cells with a density lower than a threshold
- Forming clusters from contiguous cells

II.2.3 Conclusion

We discussed the five paradigms. These one correspond to very different approaches, although they have the same goal, i.e., to divide a dataset into several homogeneous groups (clusters) of data points. There are several aspects involved in the differentiation of the five paradigms. Without being exhaustive, below are some of them (Fahad et al., 2014):

- what type of cluster is produced? 1) nested cluster, i.e., a cluster composed of smaller clusters; 2) flat cluster, i.e., a whole cluster containing no sub-clusters;
- what kind of cluster shape is produced? Spherical, elliptical or arbitrary shape;
- is inner processing based on data structures? e.g., tree (hierarchical approach), grid (grid-based approach), no structure needed for partition-based methods.

In this thesis, we focus essentially on the two following approaches: partition-based methods (also called centroid-based methods) and density-based methods. Despite this focus, we have used some tools from other paradigms in our contributions.

II.3 Partition-based clustering methods

II.3.1 Introduction

In this sub-section, we discuss partition-based methods. We recall that clusters each have a center that represents the points of the cluster in these methods. First, we focus on the best-known algorithm of this group of methods, k-means. Then we discuss its alternatives and the reason that led to the development of these alternatives.

II.3.2 K-means

II.3.2.1 Problem definition

K-means (Forgy, 1965, Lloyd, 1982a) is one of the most widely used data clustering algorithms. Due to its practicality in many fields of application, this algorithm has been popular from 1965 (Jain, 2010b, Liu et al., 2012). K-means implicitly seeks to optimize the following objective function, i.e. for each cluster, k- means minimizes the distance between the points and their centroid:

$$\sum_{j=1}^{k} \sum_{\forall x \in C_k} dis(x, G_k)^2 \tag{II.1}$$

where G_k is the centroid of the cluster C_k and dis is the distance function. It generally corresponds to the Euclidean distance. If another distance is used, convergence is not guaranteed unless it is proven.

K-means is considered an optimization problem. Nevertheless, the problem is combinatorial and computationally NP-Hard. So, Lloyd's algorithm (Lloyd, 1982a) (called standard k-means or simply k-means) is an iterative heuristic has been developed to solve approximately the k-means problem optimization. It converges to a local optimum; i.e., it affords a solution that is not necessarily the optimal one.

Lloyd's version consists of an iterative process that ends when centroids do not change between two iterations. This process consists of two steps:

- assigning each point to the nearest centroid using the euclidean distance;
- recalculating the centroids, i.e., the average of the point values of the newly formed clusters.

It has been shown that k-means can be seen as a probabilistic approach. Indeed it produces about the same results as Gaussian Mixture Model (GMM) (Titterington et al., 1985) under certain conditions. GMM considers that the points of each cluster are sampled from a Gaussian distribution. During its process, it learns the parameters of each Gaussian cluster via the Expectation-Maximation (EM) algorithm (Dempster et al., 1977). When GMM imposes that the Gaussian distribution is isotropic, then, in this case, the clusters it produces are strictly spherical, so GMM and k-means produce similar results.

II.3.2.2 Limitations

The quality of clustering and convergence depends on the choice of the number of clusters (k), the initial centroids and the distribution of data (Jain, 2010a). These three factors can lead to good or reasonable clustering or clustering making no sense in practice. In figure II.1, three different data sets highlight the limitations of k-means in the face of different data distributions. In figure II.1(a), k-means could not identify the two elliptical clusters. Indeed, k-means only identifies spherical clusters or tries to form spherical clusters even if they do not exist in the ground truth. In figure II.1(b), three clusters which do not have the same cardinality in points but which are separated. In this case, k-means failed to detect the clusters correctly. Indeed, an area with a high density of points leads to points around or a little far away being in the same cluster even if they are not in the ground truth. Moreover, isolated points also strongly influence the location of the centroids. As a result, very distant points can be found in the same cluster. In II.1(c), k-means unrolled on two clusters that do not have the same variance; i.e., they do not have the same diameter, resulted in one cluster absorbing another. K-means has difficulties when the clusters do not have the same variance or with very different relative variances.

It should be noted that even if the initialization of centroids is good, the difficulties discussed so far are not solved. Other strategies are needed to limit or solve these

problems.

Another limit concerns the centroid representation. Indeed, the centroid is the artihmetic mean of the point values of the cluster. This representation is not appropriate for use cases wherein the means are not meaningful or the data is not continuous and numeric (e.g. categorical data).

Even if k-means has a probalistic equivalent via GMM. This latter requires the user to specify the value of k, and its clustering quality is sensitive to this value. Moreover, the EM algorithm has difficulties converging towards an optimal solution in some cases, for example, when it faces medium-dimensional data. Consequently, to limit this difficulty, some constraints are imposed, such as limiting spatial orientations of clusters that can be identified.

The limitations will be discussed in more detail later in the chapter.

II.3.2.3 Versatility

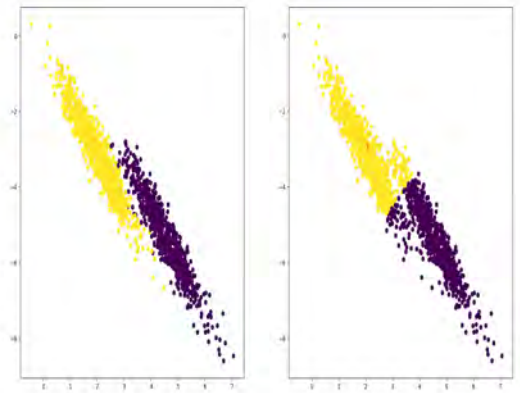
One of the advantages of k-means is its versatility, i.e., at the algorithmic level, that its subparts can be improved independently. Changing one sub-part does not necessarily have a negative impact on another part. We distinguish two types of changes that can be made on k-means (figure II.2). The first one is external. It consists in proposing the most optimal possible inputs for k-means to lead it to converge more quickly and to better optimize the objective function as opposed to providing random inputs, which often produces a poor final result. Possible improvements include determining a locally optimal initial set of centroids for k-means or identifying the actual number of clusters in the dataset. The second one is internal, it consists in deploying strategies 1) to improve the clustering quality in the sense of having a good correspondence between the result and the ground-truth and 2) to speed up the k-means execution time. Obtaining good clustering could imply an optimized objective function but not necessarily. The same logic applies in the sense optimized objective function to good clustering.

II.3.3 K-means alternatives

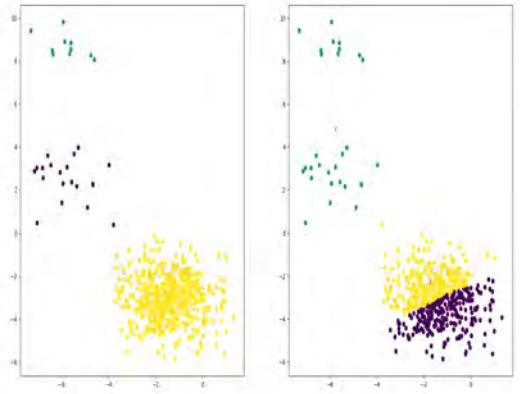
In this part, we discuss partition-based clustering methods inherited directly from k-means. Their development is due to k-means versatility.

II.3.3.1 K-median

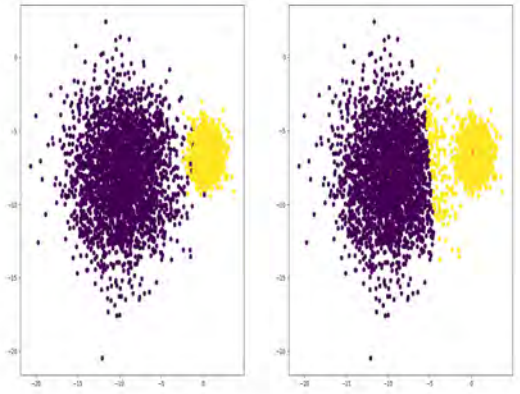
K-median (Bradley et al., 1996) differs from k-means in the definition of the cluster centroid. K-median uses the median instead of the mean to represent a cluster. Indeed, it is known that medians are less sensitive to extremes of a list of values. The purpose of k-median is to assign each point to the nearest median while minimizing the following



(a)



(b)



(c)

Figure II.1: To the left of each figure, the ground truth (one color for a cluster). On the right, an example of clustering by k-means. Crosses represent centroids.

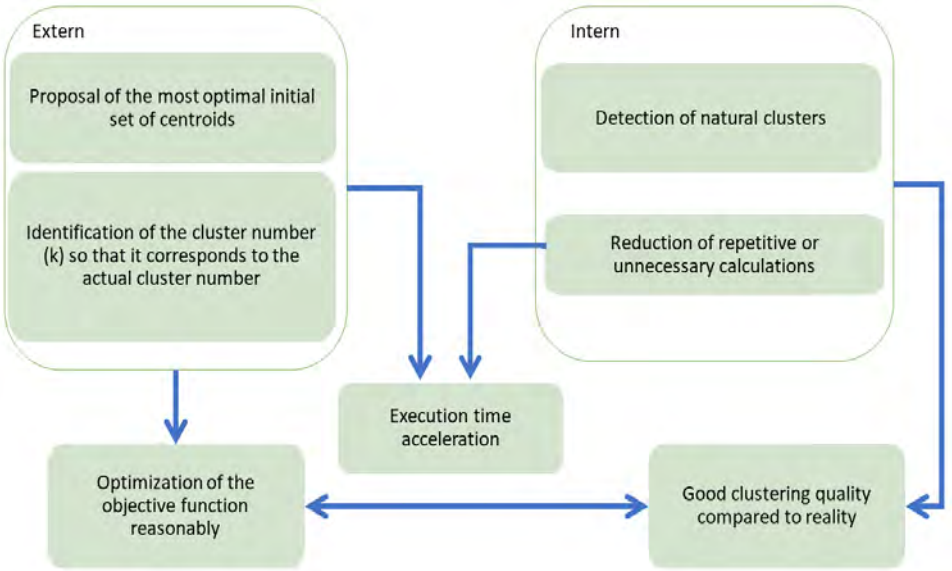


Figure II.2: K-means versality

objective function:

$$\sum_{j=1}^{k} \sum_{x_i \in C_j} |x_i - G_j|^2 \tag{II.2}$$

with $|.|$, for this function, is the Manhattan distance and G_j is the median of cluster C_j. Concretely, the result is a partitioning of the dataset into clusters, so that the distance between each point and its median is as small as possible. Instead of Euclidean distance, Manhattan distance is used to measure the distance between the points and the centroids. At the algorithmic level, k-median consists of two steps as k-means. The first one consists in assigning each point to the nearest median and the second one in calculating the median of each cluster. The k-mediane advantage is that it is more robust to outliers and noise.

II.3.3.2 K-medoids

In some use cases, the mean or median does not make sense in practice; e.g. if the points are telecom network stations, then the average of the stations is meaningless. In the k-medoid algorithm (Kaufmann and Rousseeuw, 1987), close to k-means, the centroid's value is taken from the data set. Therefore, the value is real because it is not derived from an aggregation of points, as is the case in k-means and k-medoid, where the centroid does not exist in the dataset. The centroid is called medoid. It corresponds to the most centralized point of the cluster. The other points of the cluster are non-medoid.

At each iteration, the points are assigned to the nearest medoid. By this, the algorithm minimizes the quadratic sum of the dissimilarities between each point and the medoid representing it. Indeed, the objective function is similar to that of k-medians except that the centroid is a medoid instead of a median. Then follows the

phase of updating the medoids, it corresponds to an iterative sub-process wherein each iteration, a test of replacement of each medoid by a non-medoid. By calculating scoring functions, this test verifies whether it is better to replace one medoid by another. This phase is costly. On the one hand, the solution search space is exponential, and on the other hand, $(k-n)k$ tests are carried out.

II.3.3.3 K-modes

The basic concept of k-means is based on mathematical calculations (averages, Euclidean distances). It is not suitable for categorical (qualitative) data. These are discrete and even non-ordered in some cases (e.g., first names). Therefore, clustering algorithms dedicated only to numerical data cannot be used to classify categorical data that exist in many real-world applications. One could think about transforming categorical data into numerical data and possibly apply k-means. However, k-means uses numerical distances, so it could consider two close data points that are far apart but have been assigned two close numbers. In clustering research, much effort has been put into developing new methods for clustering categorical data. One of them is the k-mode method (Huang, 1997), which is widely used in various applications.

k-modes is similar to k-means except that cluster centroids are modes and the distance calculation is adapted for categorical data.

Points and modes are vectors of $d \in \mathbf{N}^*$ dimensions (attributes). The mode of an attribute in a given cluster is the most represented or frequent value.

Formally, the objective function to be minimized is :

$$\sum_{j=1}^{k}\sum_{i=1}^{n} f_{i,j} dis(x_i, G_j) \tag{II.3}$$

with $f_(i,j)$ indicating whether or not the ith point belongs (binary value) to cluster j and G_j the mode of cluster j and $dis(x_i, G_j)$ is the dissimilarity between points x_i and G_j and it is computed as follows:

$$\sum_{l=1}^{d} \sigma(x_{i,l}, G_{j,l}) \tag{II.4}$$

where

$$\sigma(x_{i,l}, G_{j,l}) = \begin{cases} 1 & x_{i,l} = G_{j,l} \\ 0 & x_{i,l} \neq G_{j,l} \end{cases} \tag{II.5}$$

with $x_{i,l}$ and $G_{j,l}$ are values of attribute l of points.

II.3.4 Conclusion

Partition-based methods perform clustering while optimizing an objective function, which is the sum of squared errors, i.e., minimizing the squared distance between the points and the centroid in each cluster.

The difference between these methods is essentially focused on the type of cluster center (mean, median, medoid, mode) and the distance to compute the similarity between points and cluster centers. Nevertheless, k-means is still widely used because of its linear complexity as a function of the number of points and clusters. For this reason, we are particularly interested in this method in this manuscript. Moreover, the contributions made to k-means could be extended to other partition-based methods with some adaptations.

II.4 Optimization strategies for k-means

II.4.1 Introduction

In this section, we discuss the optimization strategies explicitly implemented for k-means. They are either designed for speeding up time execution or improving quality or both. These strategies respond to different limitations of k-means. First, we address the problem of estimating the number of clusters in the dataset. This number is a parameter whose value is left to the free choice of the users. This parameter is essential because it has a strong influence on the intensity of cluster homogeneity. Different strategies have been developed to answer this problem. However, they use k-means several times before arriving at an estimate. We discuss their concept and their limits in more detail in subsection II.4.1.1. Another problem concerns the initialization of the centroids (subsection II.4.1.2), which also affects the clusters' homogeneity and is also done manually by the users. Strategies are proposed to initialize the centroids. Some of them do not call k-means but provide their set of centroids that they consider optimal. They generally make k-means converge faster and improve quality. Others use k-means several times or are complex in computation. In this case, they generally initialize the centroids and partition the dataset into clusters. Due to this complexity, the quality is generally better than the first centroid initialization strategies, but less so in terms of execution time. Finally, the last problem we discuss is the high cost of computations in k-means in a large amount of data environment. We present three fundamentally different strategies (subsections II.4.1.3, II.4.1.4,II.4.1.5) that accelerate the execution time of k-means. This time, optimization is totally internal to the k-means algorithm.

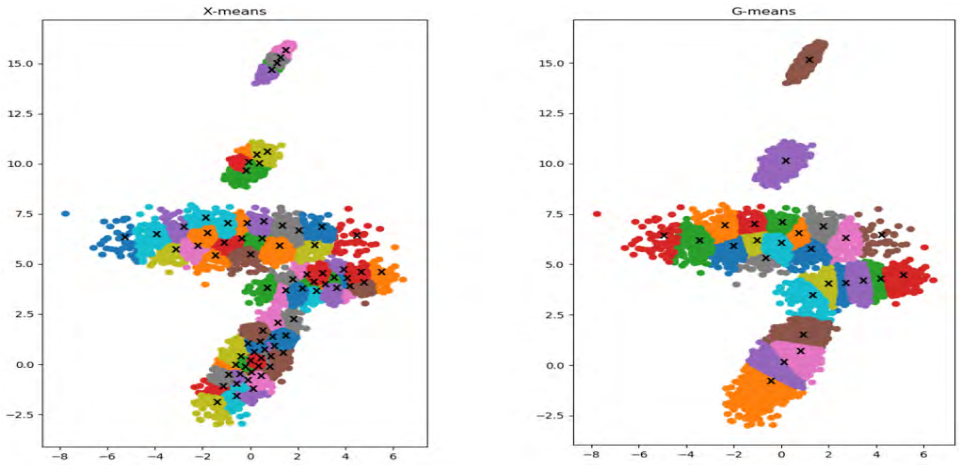


Figure II.3: Estimation of k by each of the both algorithms when clusters overlap. The dataset contains 5 true Gaussian clusters. The respective estimates of X-means and G-means are 67 and 26. The centroids are represented by black crosses. The horizontal and vertical axes refer to both dimensions of the used dataset.

II.4.1.1 K estimation

Due to the unsupervised clustering nature, estimating the number of clusters (k) is a major problem. In the case where the clusters are all spherical and well distinguishable (without overlapping), if the user does not provide the actual value of k then the clustering result is unsatisfactory. In reality, this simple configuration of cluster distribution is rare. Indeed, clusters can be of different shapes, sizes, orientations, and localities, making the task of estimating k more complex. This complexity is accentuated when clusters overlap in the data but correspond to different phenomena.

This problem has been studied for many years (Halkidi et al., 2001). Among the most representative works are X-means(Pelleg and Moore, 2000) and G-means (Hamerly and Elkan, 2003). They are based on information theory and probability tools. We study them in more detail in this part. These algorithms not only estimate the number of centroids but also propose a partitioning of the dataset. Indeed, they correspond to hybrid solutions wrapping the k-means algorithm.

II.4.1.1.1 X-MEANS X-means (Pelleg and Moore, 2000) is a k-means based algorithm that searches for the number of automatic centroids while ensuring that these centroids best match the dataset. Starting from a minimum k (k_{min}), generally equal to one, the algorithm iteratively adds new centroids if necessary until the maximum value of k (k_{max}) is reached. At each iteration, the algorithm identifies the subset of centroids that must be divided into two. This identification is made using a statistical criterion called Bayesian Information Criterion (BIC). The latter is used for model selection and is based on the likelihood function.

The X-means process consists of two operations that are iterated until completion: *Improve_param* and *improve_structure*. *Improve_param* is only the execution of

k-means with the current k as a parameter while $Improve\_structure$ researches where new centroids would be added. For each centroid and its associated region of points, three sub-operations follow each other:

- the corresponding BIC is calculated;
- k-means is executed with $k = 2$, two children's centroids are obtained;
- the BIC of the same region is calculated but taking into account the two children's centroids instead of their parent centroid.

If the previous BIC is smaller than the next BIC, then the parent centroid is replaced by its two children's centroids, or the parent centroid is retained.

II.4.1.1.2 G-means This algorithm (Hamerly and Elkan, 2003) wraps k-means and adopts almost the same approach as X-means. Instead of the BIC criterion, the gaussianity test (at the level of α confidence defined by the user) on clusters is used to decide whether to divide a centroid in two. The G-means algorithm starts with a small number of centroids and increases the number of centroids following a hierarchical process.

In fact, at each iteration, each centroid is assigned points, forming a cluster. Then a statistical test of gaussianity is applied to this cluster. If the test validates the hypothesis that the cluster is not Gaussian, the corresponding points must be represented by several centroids. Hence the division of the centroid in two using a strategy proposed by the authors using Principal Component Analysis (PCA) (Wold et al., 1987) . Otherwise, if it is Gaussian, the points are represented by this centroid. Between each division cycle, k-means is executed on all dataset and all centroids to refine the current solution (equivalent to the $improv\_param$ step of x-means).

II.4.1.1.3 Limitations The limitations of the above methods are related to the difficulty of dealing with overlapping clusters, the quality of clustering (and therefore also the value of k) they produce as well as the execution time to provide a result.

Each of the both algorithms can identify only a limited number of more or less spherical cluster types. X-means is suitable for data whose clusters are strictly spherical (Hamerly and Elkan, 2003). If other forms of clusters are present, then the estimate of k could be overestimated. On the other hand, G-means could identify clusters whose spherical shape is less strict than that identified by X-means. If the clusters are well spaced apart, G-means could provide the relevant value of k (Hamerly and Elkan, 2003). In addition to this distribution, the strict sphericity requirement must be added to the clusters for X-means to have the same performance. If the clusters overlap, then none of them estimates the correct value of k. Under these conditions, X-means and G-means tend to overestimate k.

These cases are illustrated in Figure II.3. Five Gaussian clusters have been generated but with more or less different sphericity. Two clusters are well separated from each other, the other three overlap. The two algorithms were run on this set of 20,000 data points. The k_{max} has been set at 140. G-means identified 26 clusters but detected the 2 clusters well separated from the others. However, there are overfitting on the remaining three clusters and an overestimation of 24 clusters instead of only estimating 3. X-means estimated k at 76. Overestimation is slightly higher for overlapping clusters than for separate clusters.

In terms of execution time, the two algorithms are not suitable when the data are massive. It is even more accentuated when the estimated value of k tends to be significant.

II.4.1.2 Centroid initialization methods

The partitioning of the dataset is dependent on the centroid initialization step, which by default, consists of randomly selecting k centroids from the points in the dataset (Celebi et al., 2013). A wrong choice of centroids leads to a partitioning of low quality or even meaningless for the users. Depending on the applications using k-means, centroid initializations' level of efficiency and accuracy could be very strict.

Algorithms are available for initializing k centroids. However, they do not necessarily lead to the same result or the optimal solution. An initialization algorithm's choice influences two aspects: clustering quality and the execution time. A compromise is involved, and its balance is defined according to the type of data analysis project. Some algorithms have a higher probability of proposing an initialization that leads to a correct result than others, i.e. a result that could be exploited by the users. In the same way, some are faster than others without necessarily being better in quality. We list some of them for more details but without being exhaustive.

One of the most widely used methods is Forgy's method (Forgy, 1965). It randomly takes k centroids from the dataset and then assigns the remaining points to the clusters with the closest centroid. Macqueen's method (MacQueen, 1967) follows the same logic except that the way of assigning points to clusters is different. Each time a point is assigned to a cluster, its centroid is recalculated while considering the new point. The justification behind these methods based on a quasi-random strategy is that random selection is likely to select points in dense regions, i.e., regions that are likely to be final clusters or contribute strongly to them. According to these randomized methods, the points of dense regions are considered to lead to correct results. However, in practice, these methods are likely to take outliers or points that are too close together as centroids. This reality is due to the strategy based on chance without any regularisation to avoid falling into these scenarios. Besides, several executions are necessary to choose the best method to get the best result.

The Bradley and Fayyad method (Bradley and Fayyad, 1998) is a method that reduces the random effect compared to the above methods, starts by randomly partitioning the dataset into J subsets. These subsets are clustered using k-means initialized by the MacQueen method, producing J sets of intermediate centroids, each with a cardinality of k. These sets of centroids are combined into a superset, which is then clustered by k-means J times, each time initialized with a different set of centroids. The group of centroids that minimizes the objective function of k-means the most is then taken as the final centroid group. Besides this method, $kmeans++$ (Arthur and Vassilvitskii, 2007), a method recognized for its performance and efficiency. Its goal is to identify the k centroids furthest away from each other. Gradually, one by one, it selects a new centroid among the points of the dataset such that this centroid is the furthest from the other centroids possibly already existing.

The methods discussed so far are of linear temporal complexity. Other methods have been proposed with greater complexity but more complex in operation. Among them, the binary division method (Linde et al., 1980) takes $G_1 = mean(X)$ as the first centroid. At iteration $t \in \{1, 2, ..., log_{2k}\}$, each of the existing $2^{(t-1)}$ centroids is split into two new centroids by subtracting and adding a fixed perturbation vector e, i.e. $G_j - e$ and $G_j + e$ ($j \in \{1, 2, ..., 2^{(t-1)}\}$). These 2^t new centroids are then refined using k-means. There are two main disadvantages associated with this method. Firstly, there is no indication of the choice of an appropriate value for e, which determines the split direction. Secondly, the method is computationally demanding since, after each iteration, k-means must be run for the whole dataset. The value of e could be determined by the PCA method (Huang and Harris, 1993). However, the calculation requirements are higher because of the generation of eigenvectors in PCA. Another known method, the global k-means method (Likas et al., 2003), takes $G_1 = mean(X)$ as the first centroid. At the iteration $i \in \{1, 2, ..., k-1\}$ it considers each of the n points as a candidate for the ith $(i+1)$ centroid and runs k-means with $(i+1)$ centroids on the data set. This method is computationally prohibitive for large datasets because it involves $n(k-1)$ executing k-means on the dataset.

In short, expensive methods are deterministic and lead to a better result. They also improve the convergence of k-means. Indeed, they correspond to hybrid and complex approaches. However, for applications requiring a fast response time, they are not practical. On the other hand, relatively cheaper (compared to previous methods) methods are faster (linear complexity) and could offer correct results. Nevertheless, it is advisable to repeat the initialization method several times to get a reasonable quality one. This is especially true for methods based on quasi-random approaches. Moreover, non-expensive methods are sensitive to outliers or even to the order of points, such as Macqueen's method (MacQueen, 1967). K-means++ (Arthur and Vassilvitskii, 2007) presents a relatively good compromise between quality and execution time. Given its

reasonable practicability, it is implemented in several reputable libraries of statistics and machine learning.

II.4.1.3 Parallelization

Despite the simplicity of k-means at several levels, such as implementation, application to various domains, and interpretation of results, it remains problematic for its execution time in the context of large data. At several levels of the algorithm, the computation time increases when its parameters increase. In the first step of k-means, at each iteration, the more the number of points n and the number of clusters k increase, the more the number of point-centroid distances increases. In the second phase, the more n and k increase, the more expensive the centroids' calculation is.

One way to reduce these costs is to parallelize computations. Various works (Kucukyilmaz, 2014) have focused on parallelizing k-means to save execution time; e.g. versions based on the MPI platform (Dhillon and Modha, 2000, Zhang et al., 2013), or the MapReduce paradigm (Zhao, Weizhong; Ma, Huifang; He, 2009). These works are generally based on one of the two following architectures:

- shared-memory architectures: the processors have direct access to common data located in the main memory. However, this architecture has a small number of processors because the bus interconnection's physical network has a limited data transfer capacity. The connection between the processors and the memories becomes a bottleneck since all the processors share the same bus.

- shared-nothing architectures: the processors share neither the memory nor the disk. Data is shared across the processors. The processors communicate by passing messages through an interconnection network. This architecture is scalable because each processor is independent of the other processors, and communication is iter-processors could be expensive.

  In the k-means version, due to its iterative aspect and the dependencies between the two phases of k-means, the exchange of intermediate results between processors is important. Therefore k-means can be parallelized in both architectures. However, these architectures have to face the communication cost, which increases when the amount of data and the number of calculations increases.

II.4.1.4 Tree-based k-means versions

The purpose of parallelized versions of k-means is to store data on distributed memory spaces and parallelize calculations to save execution time. Indeed, they do not seek to propose strategies for optimizing k-means algorithmically, i.e., avoiding unnecessary calculations.

We discuss other k-means optimization strategies that effectively deal with this aspect of avoiding useless computations. The first of them is based on indexing data in a hierarchical data structure. The underlying idea of these versions is, during the phase of assigning the point to its nearest centroid, not to calculate the distance between the point and all other centroids. Hence the saving in execution time.

In this sense, the authors of (Kanungo et al., 2002, Pelleg and Moore, 1999) have proposed tree-based versions of k-means. Basically, these versions work as follows: before the k-means process, the points are indexed in a tree. Each node refers to a set of points and its children refer to subsets whose union represents their father. Then during the k-means process, when assigning each point to its closest cluster, the list of centroids in the tree is propagated in the up-down direction. During the propagation, nodes reject some centroids from the list because they are considered far enough from their points. As soon as the list contains only one centroid for a given node, all points of this node (including recursively nodes and children) are assigned to this single centroid. In this way, the calculation of the distances between centroid points is accelerated.

However, these versions are not suitable for large dimensions (usually > 20 dimensions). In (Moore, 2000), the author proposes Moore's Anchors Hierarchy algorithm which uses a tree built with a middle-out technique. It is effective for much larger dimensions.

In practice, these methods are much less efficient than versions of k-means based on geometric reasoning (number of dimensions ≥ 8) (Hamerly, 2010). These versions are discussed in the rest of this sub-section.

II.4.1.5 K-means versions based on geometric reasoning

K-means versions based on geometric reasoning remain among the most efficient and effective accelerated versions of k-means (Lloyd version). It is based on geometric tools and specifically and very mainly on triangular inequality, to avoid as much as possible the computation of useless point-centroid distances in the first phase of k-means (i.e., the phase of searching for the nearest centroid of each point). This strategy is useful, especially if the points do not change or change very few clusters through the iterations of k-means. Several methods, in this sense, have been proposed.

Let us discuss on the basis of the methods based on geometrical reasoning, which is the triangular inequality. Let three points a, b, c defined in R^d and dis a metric distance defined as $R^d \times R^d \rightarrow R$, the triangular inequality (figure II.4) results in this relation $dis(a, c) <= dis(a, b) + dis(b, c)$ (we assume that $dis(u, v) = dis(v, u)$ for any points u and v).Geometrically there is no polygonal path between a and c greater than the line that directly connects them.

Note that these methods do not avoid all point-centroid distances. The rate of

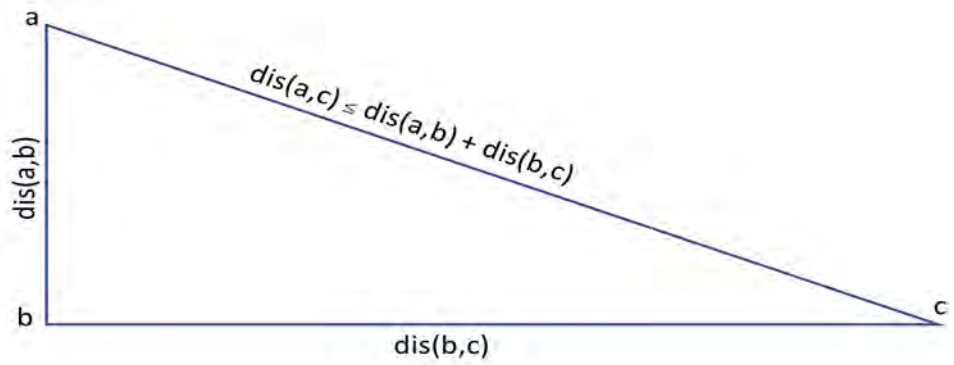


Figure II.4: Triangle inequality

calculation avoidance depends on the method and the spatial characteristics of the data. Thus, before clustering, the dataset's characteristics guide the users to choose the most appropriate version of the accelerated k-mean based on geometry reasoning. So in the remainder of this part we discuss chronologically the best known methods belonging to this category.

In the following, by language misuse, the algorithms could be called by their name or that of their author without being preceded by the word algorithm (e.g., Elkan instead of Elkan's algorithm).

II.4.1.5.1 Compare-means Compare-means (Phillips, 2002) is an accelerated version of the Lloyd version (Lloyd, 1982a). During the phase of assigning points to their nearest centroid, compare-means states that if centroids are spatially located very far from the centroid representing a given point, then those centroids are also far from that point. The idea is not to calculate the distance between this point and the centroids, which are judged to be far enough away.

Let $x_i, G_a, G_b \in R^d$, x_i a point, G_b the actual centroid of x_i and G_a another centroid, compare-means asserts that if the distance between G_a and G_b is strictly greater than twice the distance between the point x_i and its centroid G_b then the distance between x_i and G_a should not be calculated because G_a is indeed far enough from x_i in relation to G_b. This assertion is summarized in the inequation II.6 and proved via the triangular inequality in the appendices (section .1.2).

$$2 \times dis(x_i, G_b) < dis(G_b, G_a) \tag{II.6}$$

When looping the list of centroids for each point (first phase of k-means), the inequation validity II.6, for the given point and the current centroid, is checked. The actual distance is only calculated between the point and the centroid if the verification has been validated. Then, the same procedure is repeated for the next centroid. Note

that all inter-centroid distances are computed, at the beginning of the first phase of k-means and each iteration, to verify the inequation.

II.4.1.5.2 Sort-means Sort-means (Phillips, 2002) is an improvement of compare-means. Indeed, for each centroid G_j, the other centroids are sorted in the increasing order based on their distance from G_j. Thus, when looping the sorted list of centroids, as soon as the inequality validity (same as compare-means) is verified for a centroid, the centroids following it are not processed. Since these are more distant than the current centroid, so their processing is useless.

II.4.1.5.3 Elkan's version Although compare-means and sort-means reduce the calculation rate of point-centroid distances, other methods, developed later, reduce it even more under certain conditions. Among the most well-known of these methods is the Elkan version (Elkan, 2003). It uses the triangular inequality tool several times and in different ways to avoid unnecessary calculations. It introduces, via this tool, bounds on the point-centroid distances. For each point x_i it maintains 1) an upper bound u_i on the distance between x_i and the centroid G_{v_i} to which it is assigned (inequation II.7, v_i is the index of the cluster containing the point whose index is i), and 2) k lower bound, where each lower $l_{i,j}$ is on the distance between point x_i and a centroid G_j (inequation II.8).

$$dis(x_i, G_{v_i}) \leq u_i \tag{II.7}$$

$$l_{i,j} \leq dis(x_i, G_j) \tag{II.8}$$

These inequalities are exploited in the first phase of k-means. In Elkan version, this phase is subdivided into two parts: the outer test and the inner test. The outer test checks whether the point should change cluster without looping through the list of centroids. Concretely, it checks if $u_i \leq s_{v_i}$ is true (s_{v_i} is half the distance between G_{vi} and its nearest centroid). If it is true, it means that there is no other centroid closer to x_i than G_{v_i}. In this case, the same test is repeated on the next point. Otherwise, the inner test is called and loops the centroids list to determine if it is necessary to calculate $dis(x_i, G_j)$. If $u_i \leq l_{i,j}$ or $u_i \leq \frac{dis(G_{v_i}, G_j)}{2}$ then G_j cannot be closer to x_i than G_{v_i} and the distance between x_i and G_j is useless.

II.4.1.5.4 Hamerly's version Hamerly version (Hamerly, 2010) is a modified version of Elkan. It seeks to minimize the use of inner tests in favor of outer tests. Instead of k lower limits for each point, it sets just one. However, this does not have the same meaning as Elkan. Instead of limiting the distance between x_i and a centroid G_j, it limits the minimum distance between x_i and its nearest second centroid. The upper limit remains unchanged from Elkan.

In each iteration and for each point x_i, the following test must be checked:

$$u_i < max(l_i, s_{v_i}) \tag{II.9}$$

If (II.9) succeeds, then it implies that no centroid is closer to x_i than G_{vi}. In this case, we move on to the next point. If it fails, then u_i is updated with the distance between x_i and its current centroid. The test is re-performed taking into account the new value of u_i. If the test fails again, then the distances between x_i and all k centroids are recalculated.

II.4.1.5.5 Annulus's version Annulus (Hamerly and Drake, 2015b) is a variant of Hamerly, implemented to optimize the assignment phase. When Hamerly fails in its test (II.9), it must calculate the distances between point x_i and all centroids. This is where Annulus brings its touch. Instead of visiting all the centroids, only some of them are involved in calculating the distances with the point x_i.This approach is a reduction of the search space on the centroids.

$$dis(x_i, G_j) - ||x_i|| \leq ||G_{j'}|| \leq dis(x_i, G_j) + ||x_i|| \tag{II.10}$$

The inequation (II.10) is the Annulus representation. The norms $||x_i||$ and $||G_{j'}||$ correspond respectively to the distance from x_i and $G_{j'}$ to the origin. The Annulus has the center at the origin, and its mathematic model is derived from the inverted triangular inequality. Geometrically, the Annulus is the difference between two circles with the same center as the origin. The radius of the first circle is $dis(x_i, G_j) - ||x_i||$. The second larger circle has a radius of $dis(x_i, G_j) + ||x_i||$. So the Annulus width is twice $dis(x_i, G_j)$. Indeed the centroids $G_{j'}$ are in the Annulus. So when looping the centroids, only the centroids $G_{j'}$ are visited. The others are considered to be quite far from x_i.

II.4.1.5.6 Exponion's version The Exponion algorithm (Newling and Fleuret, 2016) is a geometric improvement of the Annulus algorithm. In practice, it is faster than the Annulus algorithm. The difference between them lies in the geometric tool used to reduce the search space of the centroids. Instead of Annulus centered on the origin, hyper-balls (generalized multi-dimensional circles) centered on centroids are chosen. Geometrically, the centroid closest to x_i is located on the hyper-ball centered on G_{v_i} (its closest centroid in the previous iteration) of radius u_i (upper bound on $dis(x_i, G_{v_i})$. The two centroids closest to x_i are in the hyper-ball of centroid G_{v_i} with a radius of $2 \times u_i + \delta_i$ with $\delta_i = dis(G_{v_i}, G_{j'})$ with $G_{j'}$ the closest centroid to G_{v_i}.

II.4.2 Conclusion

In this part of state-of-the art, the different strategies that improve k-means efficiency and performance are discussed (table II.1). So, optimization concerns the execution time acceleration as well as the clustering quality or the objective function optimization.

We first discussed methods for automatically estimating the number of k clusters in a dataset. More precisely, the methods (G-means and X-means) which are based on a hierarchical process and statistical tests or tests derived from information theory. We have shown experimentally that these estimation methods do not work properly when real clusters overlap and that X-means does not support non-strictly spherical clusters. These data characteristics induce them to an overestimation of k and poor clustering quality. We also discussed methods for initializing centroid k. Some are time-consuming but deterministic, and others are less costly but not deterministic and the objective function of k-means is less optimized. K-means++ stands out from the crowd by providing a good compromise between optimizing the objective function and the cost in time. It is de facto a standard in many automatic classification libraries. We also discussed the k-means parallelization. There are different parallelization paradigms, one of which is MapReduce, for distributing data and parallelizing computations. However, they have to deal with the network communication management of intermediate computation results, which can be costly when k et the number of points are large. We then focused only on accelerating the execution time of k-means (more precisely the Lloyd version) while maintaining the same clustering result. Among these accelerations, the tree-based versions avoid a certain amount of point-centroid distance computation in the first phase of k-means. In practice, when there are more than eight dimensions, they are less performant than the k-means versions based on geometric reasoning. We have studied their geometric tools for reducing the rate of calculation of unnecessary distances and their various methods among the most well known. Nevertheless, these versions do not avoid all unnecessary distances. They also generate maintenance costs to update the limits and inter-center distance matrices at each iteration.

II.5 Density-based clustering

II.5.1 Introduction

In this part, we discuss the paradigm of density-based methods (Kriegel et al., 2011). Contrary to centroid-based methods, it does not assume a priori assumptions on the distribution of points. Thus, it does not expect that real clusters are necessarily spherical or elliptical but can have any shape. It does not assign all points to clusters,

type	advantages	limitations
k estimation	no need to propose a value of k	overestimation of k and poor clustering quality for some spatial data configurations
centroid initialization methods	accelerates the convergence of k-means	the optimal k-means clustering solution is not guaranteed
parallelization	accelerates k-means by storing data on a distributed data system and performing parallel calculations	transfer of intermediate results increasingly expensive as k and number of points increase
trees-based k-means versions	accelerates k-means by reducing the rate of calculating unnecessary distances.	not all unnecessary distance calculations are avoided
acceleration techniques based on geometric reasoning		

Table II.1: Advantages and drawbacks of k-means optimization strategies

some of them are considered as outliers. This is useful if the data is noisy. In addition, users do not need to specify the number of clusters in advance.

We first discuss the basic concept of this paradigm and then its most representative methods.

II.5.2 Density-based concepts

Density-based clustering is based on the exploration of dense regions of points in the dataset (Aggarwal and Reddy, 2013). A dense region corresponds to a subspace of data where many points are relatively close to each other. Thus, a cluster in data space is a contiguous region of points with a high density of points, separated from the other clusters by less dense regions. (Kriegel et al., 2011).

The fundamental pillar of this paradigm is the consideration of a dataset as coming from an unknown probability density function ($p.d.f$). This function describes how the dataset was generated. Indeed, it corresponds to a representation of the source of the data. In this paradigm, through this function, one is not only interested in assigning each point to its most appropriate cluster but also in the generator of the cluster points to be able to subsequently generate new points in this cluster not yet existing in the dataset. The approximate estimation of this function is carried out by parametric or non-parametric statistical approaches. Parametric approaches consider that each cluster follows a probability law known a priori and whose parameters must be estimated. On the other hand, the density-based paradigm uses more or less

explicitly tools of the density function's non-parametric approaches. As mentioned above, the cluster has an irregular shape, which implies that the density function of the dataset or, more specifically that of the cluster, does not follow a pre-defined law.

Nevertheless, these methods do not integrate the notion of centroids, which problematic for applications requiring representatives. Moreover, the cluster definition of density-based paradigm implies no minimization of variance in each cluster, contrary to clustering methods based on centroids. This element differentiating them is an important factor orienting users on which clustering method to choose.

II.5.3 Dbscan

Dbscan (Ester et al., 1996) is, by far, one of the best-known algorithms in this category. It uses two concepts, essential to its operation, connectivity between points, and local density in a point's neighborhood. These concepts are controlled by two parameters entered by the user, $eps \in \mathbb{R}^+$ and $mp \in \mathbb{N}^*$. The basic principle is to build clusters according to the values of eps and mp. To define a cluster in Dbscan, the introduction of specific properties is essential, and these properties constitute the theoretical basis of this paradigm for the algorithms that followed Dbscan :

- Let a point $x_b \in X$, x_b is an interior point if in its neighbourhood (of radius eps) it has at least mp points, more formally $|N_{eps}| \geq mp$ with $N_{eps} = \{x_i \in X | dis(x_b, x_i) \leq eps\}$ the points in the x_b neighbourhood;
- Let $x_a \in X$ and $x_b \in X$ an inner point, x_a is directly density-reachable from x_b if $x_a \in N_{eps}(x_b))$ and $|N_{eps}| >= mp$;
- Let $x_a, x_b \in X$, x_a is density-reachable from x_b if there is a chain of points $x_1, ..., x_t$ with $x_1 = x_a$ and $x_t = x_b$ where x_i is directly density-reachable from x_{i-1};
- Let $x_a, x_b \in X$, x_a and x_b are density-connected if there is an third inner point so that x_a and x_b are density-reachable from that point.

So a region of points, R, is a cluster if only two conditions are met :

- Maximality: $\forall x_a, x_b$ with $x_b \in$, if x_a is density-reachable from x_b then $x_a \in R$;
- Connectivity: $\forall x_a, x_b \in R$, x_a and x_b are densiy-connected.

Dbscan has a certain sensitivity concerning border points (points that are not interior points but are in a neighborhood of an interior point). Indeed, if Dbscan is executed several times with the same parameters, the interior points remain unchanged, whereas the border points could change cluster through different Dbscan calls. This is due to the sorting order of the points. Also, it is not able to detect clusters of different

densities or nested clusters of different densities. It could also produce a lot of noise in many configurations of parameter values.

II.5.4 Denclue/Dbclasd

Denclue (Hinneburg and Keim, 1999) is an algorithm based on a probabilistic approach. It consists in modeling the probabilistic density function f of X from the aggregation of influence functions. Then use f directly to get the final clusters.

Each point is associated with an influence function. This one characterizes the influence of the point on its neighborhood. Indeed, the probability density function at a given point $p \in X$ is the sum of the influence functions of all the points at p, more formally $f(p) = \sum_{i=1}^{|X|} y_i(p)$ with y_i the influence function of the point x_i. The type of influence function is defined and parameterized by the user before the clustering process. It can be, e.g., Gaussian or square wave.

From f, particular points are identified, called attractors. They have a density higher than a predefined threshold and are indeed local maxima of f. A heuristic called climbing hill is applied to assign it to its most suitable attractor for each remaining point. The points around each attractor constitute a sub-cluster. Clusters are formed by the agglomeration of sub-clusters whose distance is less than another predefined threshold.

In addition, to access the neighborhood of a point more efficiently and make calculations locally, Denclue uses a grid-based data structure.

Denclue is presented as generalizing from Dbscan, k-means and hierarchical approaches. Nevertheless, it inherits from these also their difficulty in detecting clusters of very close density. A Denclue 2.0 (Hinneburg and Gabriel, 2007) version has been proposed to automate the step size in the climbing hill and improve the quality of points assignment to attractors. Still, it keeps almost the same solution as its predecessor.

Dbclasd is another algorithm based on a probabilistic approach of the density. Dbclasd assumes that clusters follow a uniform probability law allowing it to be parameter-free Xu et al. (1998). However, it has difficulties to detect non-uniform clusters because of this strong assumption. Also, it is dependent on the order in which the points are processed. Clustering results can vary significantly from one order to another. Dbclasd also has difficulty to separate nested densities, especially those close in density.

II.5.5 Optics/Hdbscan

Indeed, the Dbscan parameters do not allow identifying different densities, especially those that are imbricated. If the mp parameter is set, then very high-density clusters

are absorbed by less dense clusters because high-density clusters require a smaller *eps* radius than that of less dense clusters to cover the same minimum number of points. In this way, we lose information about the particularity of denser clusters.

Optics (Ordering Points To Identify the Clustering Structure) (Ankerst et al., 1999) has been proposed to address this problem. The main idea is to order the points according to local point densities to identify denser clusters. OPTICS does not produce clustering directly but a visual hierarchical structure, called reachability-plot, allowing users to extract clusters.

The reachability-plot is a sequence of valleys and peaks. The x-axis refers to the points, ordinate so that the same cluster points are practically contiguous. The y-axis is the reachability-distance, a distance introduced by OPTICS. In reachability-plot, a cluster is a valley, i.e., points with a low reachability-distance and the peaks correspond to cluster separations or noise points. To extract clusters, a horizontal line, called a cut-off, is drawn at a given reachability-distance. By reading the plot from left to right, valleys and peaks are identified from where are clusters are derived.

In short, Optics offers a visual solution where a cut is a Dbscan solution. Indeed, it offers a multitude of Dbscan solutions. However, the automatic extraction of clusters from the reachability-plot remains a complicated task. Hdbscan (Campello et al., 2013, McInnes and Healy, 2017) is an algorithm that also provides a visual solution with a hierarchical cluster structuring. Compared to Optics, it facilitates better automation of cluster extraction. Instead of a reachability-plot, it produces a condensed cluster tree, in which clusters have a hierarchical relationship (father-son clusters). It induces a new mutual reachability distance quite close to the reachability-distance of Optics. As a result, they could space out points in their new measurement space that should not be and therefore consider points as outliers and be absorbed by other clusters.

Although these approaches solve part of the problem of varying density clusters, they suffer from unevenly distributed density and high-dimensional datasets. They still mismanage low-density clusters by tending to consider them as outliers or to merge them into a higher-density cluster.

II.5.6 Conclusion

The density-based methods paradigm is based on the density notion (in the general sense of a number of points in a given volume of data space). This concept allows identifying any form of clusters without specifying the number of clusters a priori. The Dbscan, Denclue, Dbclasd methods have difficulties in finding imbricated clusters with different densities. Optics and Hdbscan have been proposed to meet this need. Nevertheless, they have difficulties in working in high-dimensional data and dealing with unevenly distributed densities.

Common problems with existing density-based clustering approaches are related

to the difficulty of dealing with low-density clusters, clusters close to similar densities, and high-dimensional data. The other limitation relates to their inefficiency in dealing properly with nested clusters of different shapes and uneven distribution of densities.

II.6 Discussion

We discussed the methods of two paradigms. First, we reviewed partition-based methods with a particular focus on k-means. Several limitations were identified regarding k-means and its environment:

- k-means and its associated reasoning-based versions are increasingly expensive in execution time as the data quantity increases;
- the computational cost of k automatic estimation methods makes them slow on large datasets;
- the k estimation methods deviate excessively from the real k cluster number in their estimation when clusters overlap or when the constraints they impose on real cluster forms are not respected.

In the density-based methods paradigm, the methods have difficulties working on high-dimensional spaces and correctly managing low density clusters or clusters close to the same density.

In the following chapters, we respond to the above-mentioned limitations with three different contributions, one chapter with two contributions responding to the limitations of k-means. The other chapter responds to the limitations of density-based methods.

Chapter III

Partition-based clustering methods optimization

Contents

III.1 Introduction

In this chapter, our contributions focus on the k-means algorithm. Improvements concern both execution time and partitioning results.

In the literature, two main limitations related to k-means have been identified. The first one is the execution time of k-means, which becomes longer and longer as datasets get more massive. So, k-means has difficulties in scaling because it has a temporal complexity proportional to knt with t the number of iterations, k number of clusters and n number of points (Jain, 2010b). Indeed, the calculation cost in a single iteration increases significantly as n and k increase. This limitation remains true for accelerated versions of geometry-based k-means even though they perform fewer calculations than the standard version. Typically k-means is used for pre-processing applications. Its usage time must be fast. In a large data environment, this property could be lost.

The other limit consists in requiring the user, for k-means, to predefine the value of k. Thus, determining inaccurately value of k has a direct negative impact on the clustering quality. Different solutions have been proposed to estimate the relevant number k of clusters (Hamerly and Elkan, 2003, Pelleg and Moore, 2000). Among the limitations of these solutions, we show three major ones:

- they call k-means several times on the same data. Consequently, in a large data context, repetitive access to data caused by the repetitive use of k-means on the same data make them computational time-consuming or unusable for applications that require applications that require a relatively fast response time;
- they hardly support overlapped clusters. This problem can lead to two different cases. i) overestimating the number of clusters exaggeratedly compared to the actual number, thus producing a very poor clustering quality. ii) significantly increasing the processing time;
- they pose a priori a strong hypothesis on the shape of the real cluster. Thus, X-means seeks to form isotropic Gaussian clusters (i.e. strictly spherical clusters where the dimensions have the same variance). G-means seeks to form Gaussian clusters (i.e. spherical or elliptical). In reality, clusters may not be Gaussian and could have a complex shape integrating more or less spherical sub-forms.

In response to the limitations mentioned, we propose two different contributions to k-means:

- sk-means: a solution designed to meet the first limitation, i.e., the one related to execution time. Sk-means aims to accelerate the execution time of standard k-means (Lloyd's version) and its versions based on geometric reasoning while having a near similar partitioning result. Indeed, the acceleration is due to the

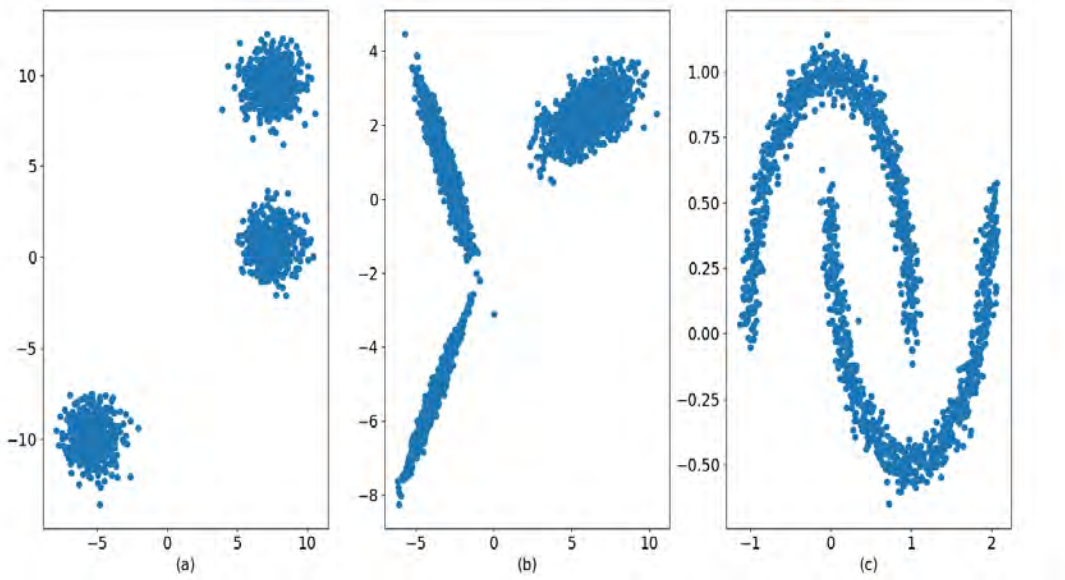


Figure III.1: Type of cluster forms: a) spherical clusters b) clusters of different ellipticity and orientation, c) clusters more complex than elliptical clusters.

exploitation and storage of information on the movements of the points through different iterations to avoid repetitive and useless calculations;

- Kd-means: a solution addressing the limitations related to the a priori estimation of the number of clusters, i.e., the limitations related to the estimation execution time, the overlap of real clusters, and the strong assumption on the shape of clusters. Kd-means estimates the number of k clusters for large datasets. Indeed, Kd-means is an algorithm based on the KD-tree(Bentley, 1975) data tree structure and a hierarchical process. In the first step, the data and their corresponding metadata are hierarchized in Kd-tree. These data, organized in groups in the Kd-tree leaves, are merged progressively from bottom to top to form final clusters. To guide point group merges, we have defined several new inter-cluster merge criteria to support complex and overlapping clusters more effectively. Kd-means is better suited than its competitors for large datasets and pre-processing tasks for data analysis projects requiring a reasonable response time.

In this chapter, the sections III.2 and III.3 are devoted respectively to Sk-means and Kd-means. Each of these sections is divided into two parts: 1) the presentation and explanation of the solution process; 2) and the solution robustness experimental validation against its competitors.

III.2 SK-means

Symbols	Meaning
n	number of data points
X	$X = \{x_1, \ldots, x_n\}$ is a dataset
k	number of clusters
C	$C = \{C_1, \ldots, C_k\}$, is a set of clusters,
G	$G = \{G_1, \ldots, G_k\}$, is a set of centroids, each cluster C_j is associated with its centroid G_j
$dis(x_i, x_j)$	dis is a distance between the points x_i and x_j
s	$s = \{s_i \mid i = 1...n\}$ set of stability levels, each point x_i is associated with its stability level s_i.

Table III.1: Symbols

III.2.1 Introduction

In the standard version of k-means (Lloyd's version), and especially in its assignment phase, $n \times k$ distances are calculated at each iteration with n the number of points and k the number of clusters to be identified. This phase is formulated as a problem of finding the nearest neighbor for each point, except that the neighbor of a point is not any point but a centroid. Each point is then assigned to its neighboring centroid cluster. In this context, the larger n and k are, the higher the computational cost. This cost is even higher if the distance function is also computationally expensive.

As we have seen in the literature, several accelerated versions of k-means have been proposed. Among them, the versions based on geometric reasoning whose most essential tool is the triangular inequality. These versions focus precisely on the k-means first phase. Their goal is to identify each point's closest neighbor while being computationally less expensive than Lloyd's version.

In order for versions based on geometrical reasoning to generate run-time accelerations, they rely on new variables they have introduced and calculations associated with them. It should be noted that the gain (in execution time) should absorb these additional costs. In some cases, these costs may exceed the gain.

Each point is assigned variables, called bounds. These variables' function is to ensure the validation of tests using triangular inequality to minimize the number of point-centroid distance calculations. Nevertheless, they must be maintained and updated at least once at each iteration. If the tests fail, further updates may be required. E.g., in the Elkan version, each point is associated with an upper bound and k lower bounds, which means that a total $n + n * k$ limits must be managed at each iteration for the whole dataset. Updates result in either addition or subtraction adjustments or distance calculations.

One of the reasons for these costs is that the information on the data points' long-term movements during the iterations is not exploited for the following iterations(note that a movement of a point means that it changes cluster). Whether in k-means or its accelerated versions, there is no information saved on the long-term dynamicity of the points during the process (i.e., during a sufficiently long number of iterations). Instead, the accelerated versions focus more on the nearest neighbor (centroid) problem for a given iteration without having a strategy with a global view on all iterations where information can be exchanged even between two inconsequential and distant iterations. In short, the information of one iteration is lost for the following iterations and, more particularly, the iterations that are a bit distant.

The objective of our contribution in this section is to propose a novel strategy to reduce the additional computational cost of Lloyd's version and the versions based on geometric reasoning while keeping the partitioning almost identical. This strategy can be integrated into the standard version of k-means (Llyod version) and its accelerated versions based on geometric reasoning. The integration of the strategy in these versions always allows them to work with the external environment. Indeed, they can always be coupled with the same centroid initialization methods as if they did not integrate the strategy and support the same distance metrics, including Euclidean distance.

III.2.2 Optimization strategy

In this sub-section, we introduce the strategy, as well as elements containing it, whose goal is to accelerate standard k-means and its geometry-based accelerated versions.

During the k-means process (including its geometry-based versions), there are several point categories defined by the movement degree of the points (the cluster change of a point in an iteration is a movement):

- points move from one cluster to another in a recurrent way. These points are very dynamic;
- points that move less frequently, they are less dynamic;
- points, which during a series of consecutive iterations, change clusters, then from a given iteration, they remain attached to a cluster until the process end;
- points that do not move from the first iteration, they are assigned to a cluster to which they will remain fixed until the process end.

The points of the first group are very dynamic, they are the ones that keep the process going. So, as soon as the points in this category no longer change cluster then the k-means process ends. So, we focus mainly on the last three categories.

Our goal is to identify the least dynamic points, i.e. belonging to one of the last three categories. Indeed, these points represent a significant cost in time because they

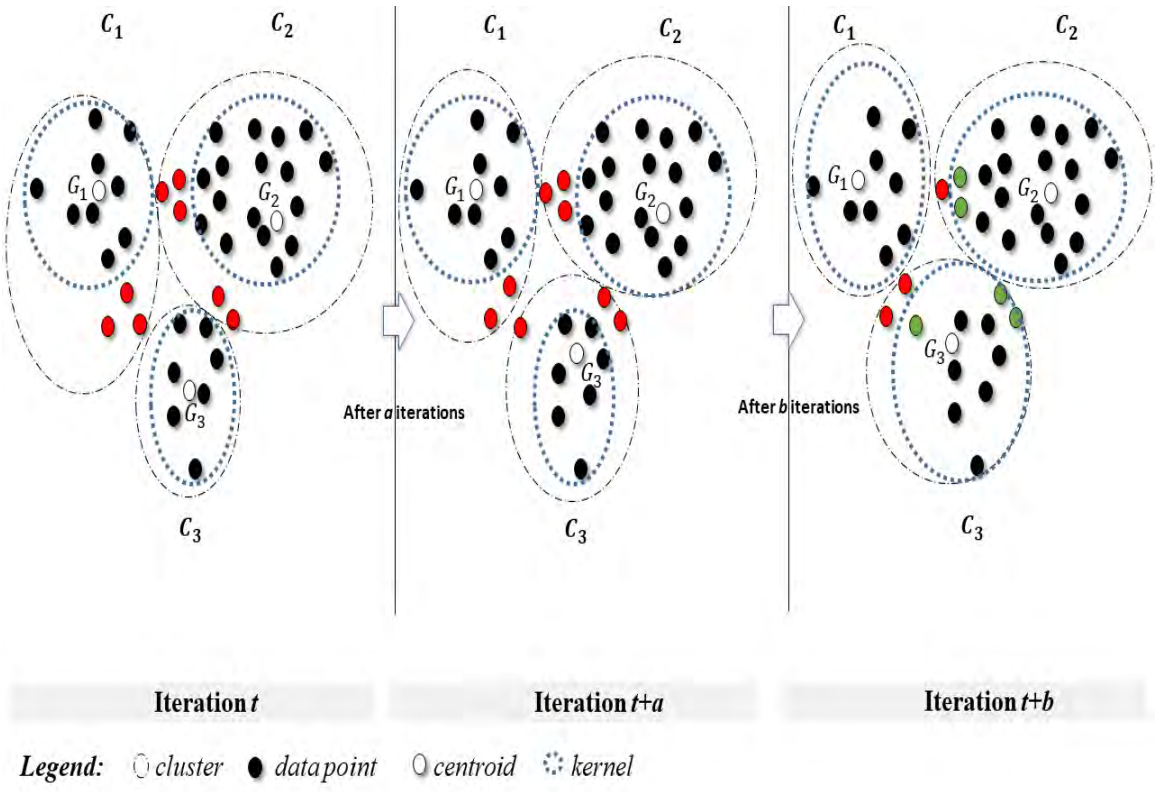


Figure III.2: K-means process of three clusters. Illustration of the clusters and their kernels evolving according to the point movements.

are involved in unnecessary calculations, tests, and comparisons. We propose to track the stability of each point. Let $s = \{s_i | i = 1...n\}$ be the set of values on the point stability. The value of s_i indicates the number of successive iterations a point has remained fixed to a cluster; i.e., it has not changed clusters during this number of iterations. This set stores information on the points' stability over the long term, which allows to see the evolution of each point individually and to classify each point according to the three categories mentioned above.

As a consequence of the introduction of s, we define for each cluster C_j two different parts corresponding to two data disjoint subsets: the kernel $IC_j \subseteq C_j$ and the outer part $C_j \backslash IC_j$. The kernel consists of points fixed to the cluster C_j until the process end. The outer part is the point set that are likely to change cluster in the following iterations. Before the process begins, $IC_j = \emptyset$, while the other part is equal to C_j. By using the information about the points' stability in s, the IC_j kernel is gradually fed during the process. Each time a point is considered to be stabilizing, it is moved into the kernel. We call this point as passive. In this example III.2, we will show how points are moved between the kernels and the outer parts through different iterations. In the example, a k-means process of three clusters is schematized. At iteration t ($t > 1$, which does not correspond to the process beginning), a kernel is already constituted in each cluster. In clusters C_1 and C_2, there are still points (red points) that can

change clusters. In cluster C_3, the kernel is equivalent to the whole cluster. However, it could receive new points from the other clusters in the following iterations. At iteration $t + a(a > 1)$, clusters C_1 and C_2 lose points, which were not in the kernel, to C_3. On the other hand, cluster C_3 considers these points unstable (red) and, therefore, not integrated into its kernel. At iteration $t + b(b > a)$, C_3 integrates the points into the kernel, gained at iteration $t+a$, because they are considered passives. At the same time, it gains two other points from C_1, but it considers them dynamic. Among the three points considered as dynamic in C_2 at iteration $(t+a)$, C_2 considers two of them as passive at iteration $t + b$. Nevertheless, the rest remained dynamic. The question that arises is why this point has not been given the same consideration as the other two, whereas from iteration t to iteration $t + b - 1$, they were all dynamic. There are two possible answers to this. The first one indicates that if the dynamic point is fixed to the C_2 cluster from iteration t to $t + b$, then the two other passive points are fixed to the C_2 cluster before their dynamic neighbor, i.e., before the iteration t until their integration in the C_2 kernel. The second considers that if the third points are set to C_2 from t, then the dynamic point has at least once changed cluster between t and $t + b$, and the others have not.

During all these iterations of the example, s was updated and used to integrate points into kernels. Indeed, before the process was launched, all s_i is equal to zero. At each iteration, for any point x_i that does not belong to a kernel and does not change cluster during two successive iterations, then s_i is incremented. Otherwise, it is reset to zero.

As soon as a point integrates a kernel, it is no longer treated in the following iterations in the calculation of point-centroid distances because it is considered passive. Consequently, most likely, its current centroid remains closest to it compared to the other centroids until the end of the process.

To assert that a point is passive and therefore sufficiently justified to integrate it into a kernel, so we defined the following condition setting the minimum stability threshold of a point according to its cluster changes :

Definition III.1. Given $r \in \mathbb{N}^*$, $x_i \in X$ a data point, x_i is **passive** if at least during r successive iterations, it has not changed cluster.

As soon as the point x_i is considered stabilizing, i.e., $s_i \geq r$, it is passive because it is integrated into the kernel and is no longer the object of calculations in the subsequent iterations. On the other hand, active points (points outside the kernel) continue to be involved in calculations until they are passive. The value of r must be large enough to ensure that the kernel centroid containing x_i is always the closest after the qualification of x_i as passive.

The value of r is calculated from an estimator E that we have developed. It depends on several characteristics of the data. The relationship between r and these

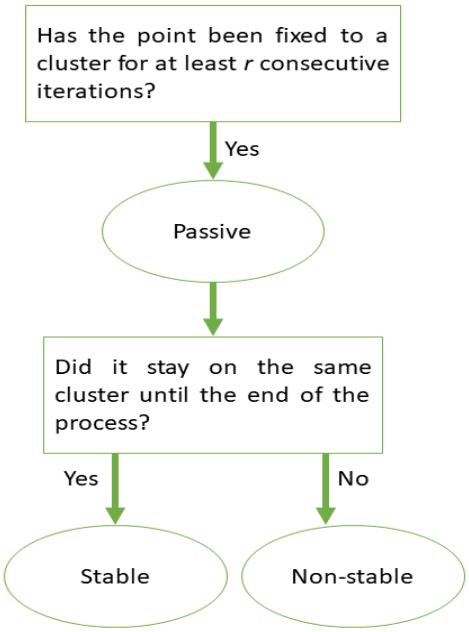


Figure III.3: The description of the point state during a k-means process integrating Sk-means. Circles refer to the point state. once the passive point, it is no longer involved in the calculations in the following iterations.

characteristics is shown in the experimental part III.2.5. In this same part we infer the model of E.

III.2.3 Intact convergence

In the following, we discuss the convergence of k-means and its versions that integrate the Sk-means strategy. However, before detailing this aspect, we introduce the following definition (see also figure III.3):

Definition III.2. Given $x_i \in X$ a given point, x_i is stable if it 1) passive (i.e. it respects the definition III.1) and 2) its centroid (that of the iteration in which x_i became passive) remained the closest until the k-means process end. If the second condition is not met then the point is passive non-stable.

The convergence property of k-means (Lloyd's version and the versions based on geometrical reasoning) is maintained even if they integrate the Sk-means strategy. Convergence means that the values of all centroids do not change during two successive iterations. This means that the objective function has been locally optimally optimized according to the initial centroids (the objective function result differs according to the spatial positions of the initial centroids). In our case, convergence, in addition to its first meaning, could also be due to the change of all active points into passive points.

A first element contributing to the k-means convergence is that there is a finite number of ways to divide a set of n points into k clusters. This number is an a priori

known integer equivalent to the Stirling number $st = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$ (Graham et al., 1994). So, it depends only on n and k. This number remains unchanged even if some data points become passive. Moreover, in a k-means (integrating or not sk-means) run, only a few of st possible solutions are produced (one possibility by iteration). However, this first element in itself is not enough. It is necessary to show that k-means, iteratively, improves the partitioning more and more according to the objective function until reaching the final partitioning. From the optimization view point, this consists in minimizing the k-means objective function (sse) monotonically at each iteration until the minimum possible local solution is obtained. Note that both phases of k-means contribute to this minimization. So we will show that Lloyd's version and the versions based on geometric reasoning integrating Sk-means converge. We show this in the first phase and then in the second phase.

Let $a = \{a_i | i = 1...n\}$ with a_i the index (natural number) of the cluster to which point x_i has been assigned. Let $a' = \{a'_i | i = 1...n\}$ where a'_i, corresponds to the new index of the centroid of point x_i, replacing a_i.

First phase. Let G be the set of centroids where G_j is the centroid of cluster C_j, the objective function (also called intra-cluster variance or sum of squared errors(sse)) that k-means minimizes is defined as follows:

$$P(X, G, a) = \sum_{i=1}^{n} dis(x_i, G_{a_i})^2 \tag{III.1}$$

If the objective function P must decrease monotonically during iterations, so the following equation must be respected:

$$P(X, G, a') - P(X, G, a) = \sum_{i=1}^{n}(dis(x_i, G_{a'_i})^2 - dis(x_i, G_{a_i})^2) \leq 0 \tag{III.2}$$

If we reduce P to the individual level of each point, we get:

$$dis(x_i, G_{a'_i})^2 - dis(x_i, G_{a_i})^2 \leq 0 \tag{III.3}$$

When the point is active, the inequation III.3 is necessarily respected. Indeed, if x_i changes cluster, i.e. $a_i \neq a'_i$ then $G_{a'_i}$ is necessarily closer to x_i than G_{a_i} because the first phase of k-means looks for the nearest centroid for each point. If x_i does not change cluster, III.3 is true because the difference in its left part is zero.

If the point is passive, whether it is stable or not, then III.3 is null. It acts on P as if it was active but except that it does not change cluster (i.e., $a_i = a'_i$). Thus, removing points from the point-centroid distance calculations in the k-means first phase, this phase still reduces the intra-cluster variance between the points and the centroid of each cluster.

Second phase

Let G' be the set of new centroid values calculated from G. We show that the intra-cluster variance of each cluster (local *sse* of the cluster) is reduced from one iteration to another. We show that this variance is minimized optimally at each iteration by updating centroids using the arithmetic mean. We reason with the Euclidean distance for this phase, but the underlying idea remains the same for other distances, i.e., showing that the intra-cluster variance decreases monotonically.

Let the intra-cluster variance $P(X, G'_j, a') = \sum_{\forall x_i \in C_j} dis(x_i, G'_j)^2$ (we consider that C is up to date with respect to a'). Since $P(X, G'_j, a')$ is quadratic (and therefore convex) (Jain, 2010a) then it is optimal when the derivative of $P(X, G'_j, a')$ is zero:

$$\frac{dP(X, G'_j, a')}{dG'_j} = \sum_{\forall x_i \in C_j} 2(x_i - G'_j) = 0 \tag{III.4}$$

,which implies

$$G'_j = \frac{1}{|C_j|} \times \sum_{\forall x_i \in C_j} x_i \tag{III.5}$$

, hence

$$P(X, G'_j, a') - P(X, G_j, a') = \sum_{\forall x_i \in C_j} dis(x_i, G'_j)^2 - \sum_{\forall x_i \in C_j} dis(x_i, G_j)^2 \leq 0 \tag{III.6}$$

The active, stable passive and non-stable passive points, in this phase, are not differentiated because their values, at a given iteration, in a given cluster C_j, are taken into account in the calculation of the new arithmetic mean, the centroid average G'_j.

Note that the arithmetic mean is not the value that minimizes the second phase at best for other distances; e.g, for the Manhattan distance, the median is chosen instead of the average. In this case, another version is obtained, k-medians instead of k-means.

III.2.4 Integration

In this part, we show how Sk-means fits into the different versions of k-means. First of all, we recall that Lloyd's version and the versions based on geometric reasoning are made of two main steps repeated in an iterative process. The first step consists in assigning each point to the cluster whose centroid is closest. The second step recalculates the centroids of the clusters.

At a given iteration and at a visit of a given point iteration in the first phase, we differentiate two other steps named outer test and inner test. The outer test step allows deciding whether to iterate to the next point because the current point does not change cluster at the current iteration. Before the decision, in the outer test, some tests are run via variables called bounds framing the point-centroid distances. If the

tests pass, then calculating the distances between the point and the k centroids is avoided. If not, the inner test is then activated. The latter conducts additional tests, via bounds, in order not to calculate all the distances between the point and the other centroids, i.e., some distances must be calculated while others must not.

We have integrated Sk-means in four algorithms: Lloyd's (Lloyd, 1982a), Compare-means (Phillips, 2002), Sort-means (Phillips, 2002) and Exponion (Newling and Fleuret, 2016) versions (they are called original algorithms). The latter resulted, via integration, in optimized versions. The algorithm 1 describes how Sk-means integrates with the original versions. The algorithm is general and adapted to these versions. The adaptation of this algorithm to one of the original versions results in the optimized version. Among the inputs of this algorithm is the value of r. This value is either calculated using our E estimator or the user provides it. Then the set of s is initialized to 0 for each of the points of X. In each iteration, in the first phase, we iterate on the points' indexes. Then, the outer test and inner test are eventually triggered. Indeed, depending on the k-means version, the outer test and the inner test are both triggered, only one of the both, or neither of the two. If at least one is used, the tests associated with the original version are used. After these tests, we check if the point has changed the cluster. If it is the case, we check if the point is likely to become passive using the value of r. If it becomes passive, then its index is removed from the list of point indices. In other words, at the next iteration, we no longer iterate on this point until convergence. Then follows the second step, which is the updating of the clusters' centroids. Then other updates of the variables specific to the original version are carried out, i.e., calculations of new values of the bounds associated with each point. In this case, when the point is passive, the bounds of the point are no longer updated.

Lloyd's version does not intrinsically have a computational optimization strategy compared to algorithms based on geometric reasoning. So it does not have the inner test and the outer test. Consequently, it is the version that takes the most advantage of the Sk-means strategy. Compare-means and Sort-means do not adopt the outer-test and therefore focus on the inner test. Exponion exploits the outer test and the inner test. The effect of Sk-means on versions based on geometrical reasoning is twofold. First, the outer test and inner test for passive points are removed, which means that bounds calculations and point-center distance calculations are avoided. Secondly, the updating of the passive point bounds, which requires additional calculations, is also removed.

We ran Sk-means on a sample dataset to show its benefits. The dataset used has 100,000 two-dimensional points and consists of 11 real clusters, some of which are contiguous. The purpose of k-means is then to identify these clusters. We have launched on the dataset the versions benefiting from Sk-means (OLloyd, OSort, OCompare and OExponion) and the versions not benefiting from Sk-means (Lloyd, Sort, Compare

Algorithm 1 Sk-means

Input: k,G,X,r
Output: C

```
1: s ← {0∀i = 1...card(X)}
2: convergence ← true
3: while convergence = true do
4:   {first step}
5:   for i in indexes do
6:     {outer test is conducted}
7:     if outer test fails then
8:       k_old ← a_i {with a_i the index of the cluster to which point x_i is closest}
9:       {inner test}
10:        {tests, specific to a given version of k-means, are performed in order to
         modify a_i or not}
11:      if a_i ≠ k_old then
12:        s_i ← s_i + 1
13:        if s_i ≥ r then
14:          indexes ← indexes\i
15:        end if
16:      else
17:        s_i ← 0
18:      end if
19:    end if
20:  end for
21:  {second step}
22:  {centroids update}
23:  {update of the boolean variable convergence}
24:  {updates of variables specific to a given version of k-means}
25: end while
```

and Exponion). The r stability value was estimated by our E estimator.

To see Sk-means' benefits, we used three results produced by each of the eight versions: data partitioning, execution time, and the number of centroid-point calculations performed. In figure III.4, we compare each version of k-means to its optimized version (i.e., integrating Sk-means) in terms of execution time and the number of performed calculations. In terms of the number of performed calculations, the optimized versions avoid on average 66% calculations than the original versions. This results into the fact that Optimized versions are significantly faster by two to more than three times than their original associated versions. As an example, the optimized version of Lloyd's avoided 71% of calculations compared to the original, which resulted in the optimized version being 3.4 times faster. Even if the optimized versions are faster, the clustering results proposed by the optimized and non-optimized versions are similar (figure III.5). This shows that removing the points from the distance calculation operations, considered passive by Sk-means, does not really influence the data partitioning.

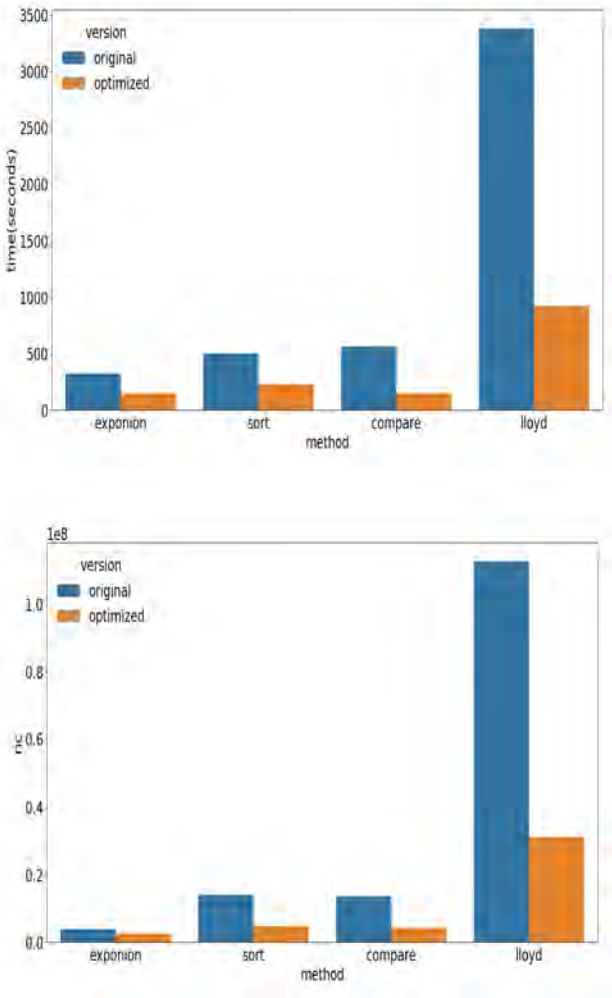


Figure III.4: On the left, the optimized versions (Sk-means) and the original associated versions are compared in execution time. On the right, these versions are compared in the number of distance calculations performed.

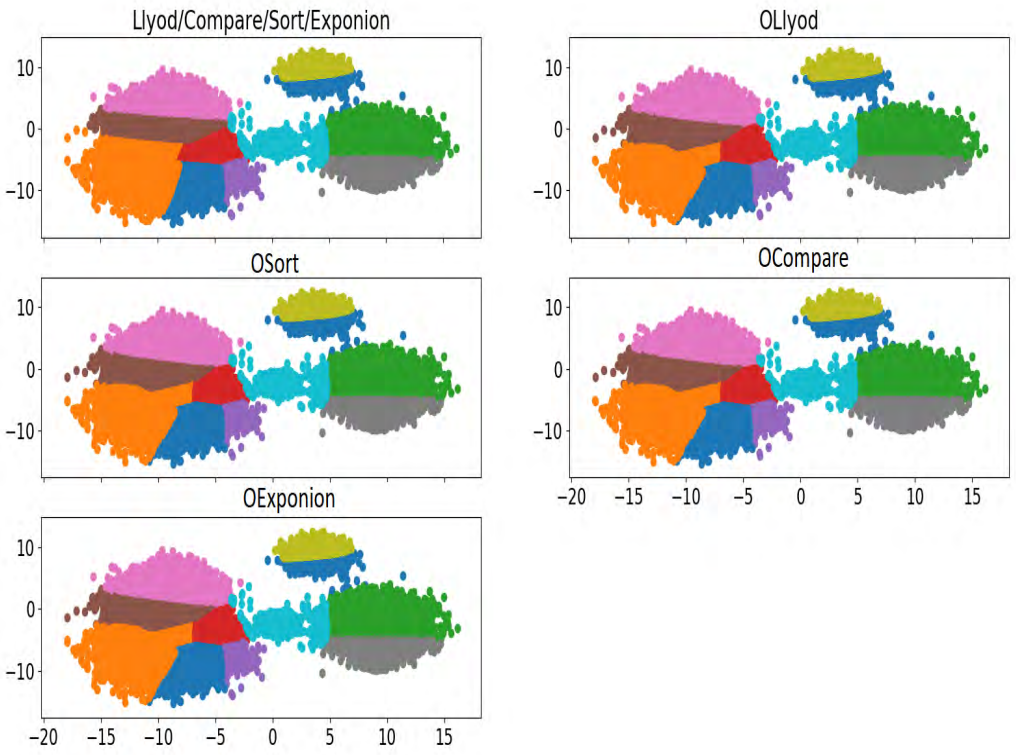


Figure III.5: Clustering results proposed by the different versions of k-means. Each color represents a cluster. Names beginning with O correspond to the names of the optimized versions. The optimized versions produce the same clustering result.

III.2.5 Experimental assessments

III.2.5.1 Introduction

In this part, the Sk-means robustness against competitors is assessed.

First of all, we will also study how the E estimator was constructed experimentally (part III.2.5.2). Then, using this estimator, we will show that the algorithms that benefit from the Sk-means strategy are significantly faster than the corresponding versions that do not benefit from it while having almost identical partitionings (part III.2.5.4). Various data sets have been used to show this gain.

III.2.5.2 Experimental protocol for the estimator E

In part III.2.2, we discussed about the estimator E estimating the minimum number of successive iterations (noted r) during a point must remain attached to a cluster to be considered a passive point. In this section, we explain the experimental approach that led to this estimator.

Before starting experiments, we postulate that the value r depends on the number of points, the dimension, the number of clusters, and the spatial distribution of the initial centroids. In other words, this dependence means that the value r is defined according to the data characteristics (controlled by parameters). To verify this postulate, three steps were performed. First, different synthetic data sets were generated. Then on each of them, k-means was run. Then, statistical tests were applied to the launches' results, which led to the validation of the postulate and the E estimator's inference.

Data generation. This step aims to generate datasets that are varied enough for better representativeness of the synthetic and real data encountered in practice. The data sets are generated in hyper-cubes (generalized multi-dimensional cubes). A hyper-cube, associated with a single dataset, allows to frame the dataset and control the data's spatial distribution. To obtain varied data sets, the values of different parameters characterizing the datasets have been varied (see table III.3). As an example, the inter-cube distance is one of the parameters for obtaining various data sets. Its influence is clear in the figures III.6 and III.7 showing two data sets of eight clusters of the same dimension (d) and each having the same number of points (n). By having an inter-cube distance of 0.2, the dataset (figure III.6) is easier to cluster (under certain conditions), in the sense that the same clusters could be found as they are in the ground truth. With a distance of -0.2, the clusters are overlapping (figure III.7), which makes the dataset much more difficult to cluster. Between the two levels of difficulty, other values of inter-cube distance make it possible to have data sets that are more or less complicated to cluster.

In addition to the choices of the data characteristics, there are different types of

Parameter	Value set
dataset cardinality($n = \|X\|$)	$\{2 \times 10^3$,1×10^4,2×10^4,4×10^4,5×10^5,1×10^5, 2×10^5,4×10^5,6×10^5,$1 \times 10^6\}$
dataset dimension (d)	{2,4,6,8,10}
number of real clusters (c)	{4,8,12,16,20,25,30}
inter-cube distance	{-0.2,-0.1,0,0.1,0.2}

Table III.2: Parameter value range

expression:

- **centroid initialization method (which we refer to as im)**. Two methods have been chosen: 1) random choice of initial centroids, so there is no guarantee that the centroids are chosen correctly in order to have a reasonable local optimal solution according to the objective function of k-means; 2) k-means++, another alternative, is chosen because globally it proposes centroids which are sufficiently far from each other and which leads to k-means generally producing a better clustering than the random method. Nevertheless, in a concern of generality as underlined above, we wish to study the initialization method's influence on r. Consequently, a parameter (sp) is proposed to characterize the spatial distribution of the initial centr oids. It is calculated as follows: given l_{max} the maximum value of the pairwise distances(between centroids) and a hyper-rectangle encompassing these centroids, sp is equal to the ratio of l_{max} to the maximum length of the hyper-rectangle. Another type of calculation of sp has been tested, but it is less efficient as shown below in the experimental results;

- **the cluster number to be identified by k-means and its accelerated versions (k)**. Indeed, k is equal to the real cluster number in the dataset (i.e., $c = k$). Despite this, in practice, the variation of the parameters sp, inter-cube distance and im leads to challenging datasets in the sense that several centroids can be in the same cluster or have very close or overlapping clusters as outlined above, which makes their clustering difficult.

A total of 1750 synthetic datasets were generated, and 3500 experiments were carried out. The results are analyzed graphically before being statistically validated.

III.2.5.3 Experimental inference of estimator E

Visual results. Graphically a monotony of scatterplots defined on two variables (parameters) is expected to state that these variables are dependent. Concretely, monotony consists in the fact that the more one variable increases in value, the more the other variable studied increases or otherwise decreases (one direction only, not both).

In figure III.8(a), in general, the scatterplot tends to grow, i.e., the more the number of points increases, the more the number of iterations increases as well. In figure

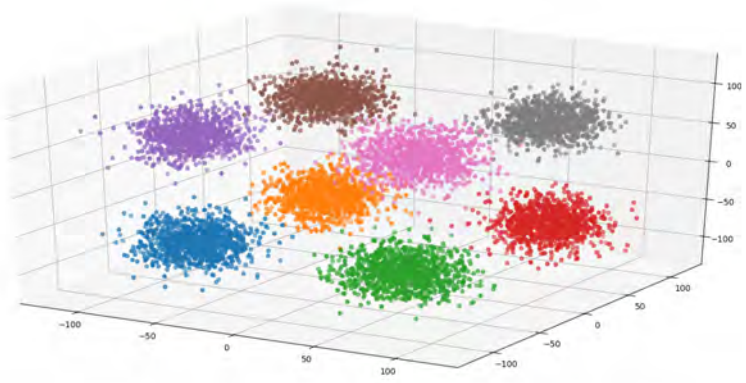


Figure III.6: a cube of eight well-separated clusters (inter-cube distance = 0.2).

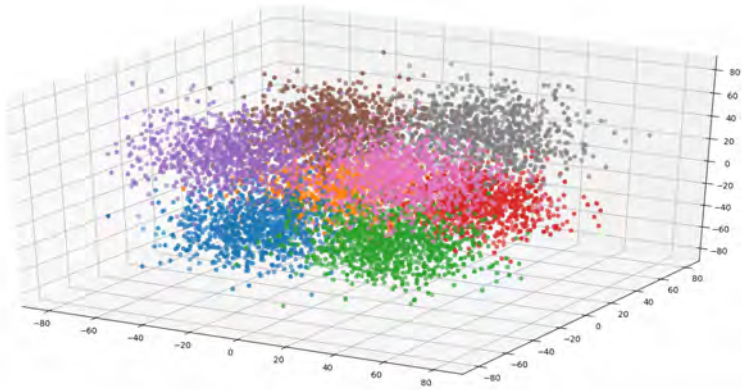


Figure III.7: A cube of eight overlapping clusters (inter-cube distance = −0.2).

III.8(b), the scatterplot, formed from the dimension (d) and the number of iterations, visibly tends to grow even though the tendency seems to be less than that of the previous scatterplot. If the scatterplot is cleaned from the few points that are far from the mass of the remaining points on each value of the dimension, the visual monotony is slightly stronger. In figure III.8(c), the scatterplot has a decreasing monotony but with a less pronounced slope than that of the scatterplot defined on n (figure III.8(a)). This decrease means that, globally, the greater k is, the fewer iterations there are. In figure III.8(d), a decreasing trend is noted. Even if noise is present around the majority of the points (dense region), it still maintains the general decreasing trend. We infer from this scatterplot that the closer the centroids(the smaller the sp) are to each other (i.e., the distance between two centroids is small compared to the inter-point distances of the whole dataset), the greater the number of iterations.

In summary, graphically, we conclude that there is monotony in the four scatterplots III.8(a,b,c,d). Therefore there is dependence between the number of iterations and the other parameters k, d, n and sp.

$$T(k,d,n) = log_2(\frac{f_1 * n * f_2 * d}{f_3 * k}) \tag{III.7}$$

The function $T(k,d,n)$ models these monotonies. Knowing that the scatterplots involving n and d have an increasing monotony, they are put in the numerator, whereas k is in the denominator because its corresponding scatterplot is decreasing. Nevertheless, the real numbers f_1, f_2, f_3 are used to weight the importance of influence on r of n, d and k, respectively. In practice, the number of iterations is generally very small or even negligible compared to the values of n,d and k. Intuitively, the binary logarithmic is adapted. Moreover, the factor f_4 is integrated to either increase or decrease the logarithmic.

$$E(k,d,n,sp) = f_4 * T(k,d,n) * (1 - sp) \tag{III.8}$$

From T, $E(k,d,n,sp)$ is inferred. The spatial distribution of the initial centroids has a non-negligible influence on the total number of iterations, hence its inclusion in E. The factor $(1 - sp)$ corresponds to an increasing version of the scatterplot III.8(d) to mean that the closer the centroids are to each other (i.e., the larger $(1 - sp)$ is), the greater $r = E(k,d,n,sp)$.

Figure III.8(e) shows the relationship between r (by extension E) and the number of iterations. The factor f_4 is set to 2 and f_1,f_2 and f_3 to 1. The monotony is much stronger compared to those of the other parameters. The scatterplot is more concentrated and less noisy. The monotony is increasing, which shows that the total number of iterations and E evolve in the same direction.

Statistical results. The *Spearman* test is a statistical measure used to confirm or disprove the dependence (or correlation) hypothesis between two variables. It produces two values: the correlation level and the p-value. The correlation is between -1 and 1, where 0 indicates independence of the variables, and the closer the value is to 1 or -1, the stronger the correlation. The p-value indicates the probability of making an error by rejecting the assumption (called the null hypothesis) that the two variables are independent. In a classical hypothesis test, a significance threshold sg is set to decide whether or not to reject the null hypothesis. Generally $sg = 0.04$, but in our study, $gs = 0.01$ is set to be more demanding. Note that even if the level of correlation is strong, but the p-value is higher than sg, then we declare that there is independence.

According to the table results III.3, the null hypothesis is rejected for r and n because the probability of error (p-value) is 0. Moreover, their correlation rate is in line with this probability of erroneous results. It is 0.88 for n and 0.93 for r, which repre-

Parameter	correlation	p-value
n	0.88	0.00
d	0.13	$1.60e^{-14}$
k	-0.28	$9.91e^{-65}$
$1 - sp$	0.47	$3.37e^{-190}$
r	0.93	0.0

Table III.3: Spearman's test results on the correlations between k,d,n, sp and r, and the total number of iterations that k-means took to converge.

sents a solid correlation. Concerning k, d and $1-sp$, the p-value is negligible compared to sg, and therefore, the null hypothesis is rejected. The dimension d has the least influence on the number of iterations, although it is correlated with it. The spatial distribution of the initial centroids, as underlined graphically, has an important influence on the number of iterations. Another type of sp calculation tested, as mentioned above, consists in dividing the average of the pairwise distances (between centroids) over the maximum length of the hyper-rectangle. So, the test on $1 - sp$ produced a p-value of $4.00 \times 10^{-}59$, which shows the existence of a correlation. Nevertheless, the correlation is 0.27. It is weaker than the first type of calculation of sp (0.47). The value of k is a negative correlation, which validates the decreasing monotony seen graphically. In sum, the null hypothesis is rejected for all parameters, and thus, the dependence between k, d, n, sp, and r, and the total number of iterations is validated.

III.2.5.4 Experimental protocol for comparing algorithms

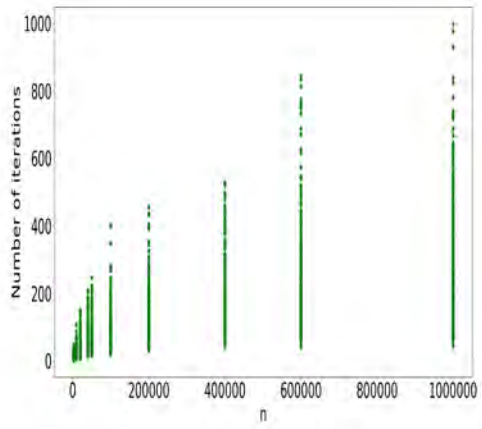
We evaluate the Sk-means strategy applied to Lloyd's (Lloyd, 1982a), Compare-means (Phillips, 2002), Sort-means(Phillips, 2002) and Exponion (Newling and Fleuret, 2016) versions. So, our optimized versions associated with these algorithms are Ollyod, OCompare-means, Osort, OExponion. The datasets (table III.4) used are of different cardinalities and dimensions and are of different types (images, sensory data, synthetic data). All algorithms have been coded in python 3.6.

An experiment consists in running each of the eight algorithms (the four versions of k-means and their optimized versions via Sk-means) on a given dataset whose entries are :

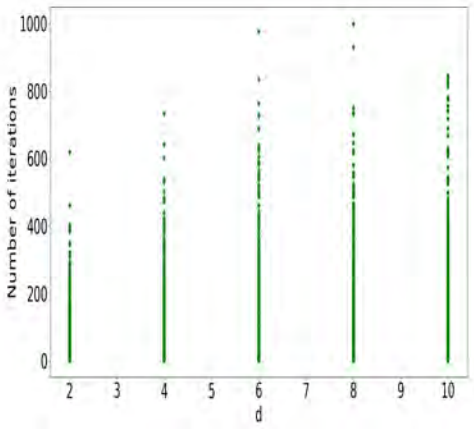
- $k \in \{8, 60, 200, 500\}$
- method of initialization (random, k-means++).

A total of 384 experiments (8 algorithms $\times$ 2 initialization methods $\times$ 4 values of k $\times$ 6 datasets) have been launched. In each experiment concerning an optimized algorithm, the stability value r is estimated.

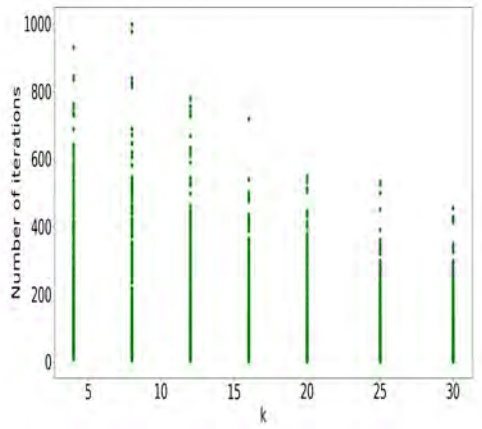
Through these experiments, execution time (including calculation execution time of r) and quality are evaluated. In this section, the **quality** concerns the **objective function minimization** of k-means (sse), the algorithm having best minimized the



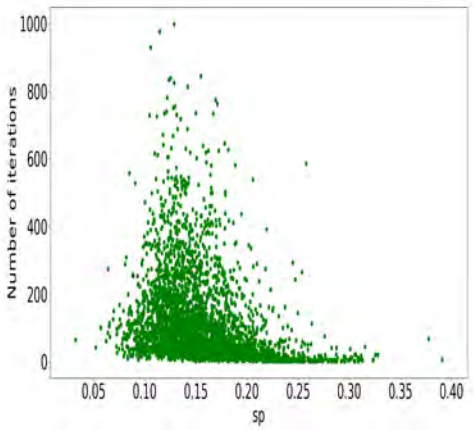
(a)



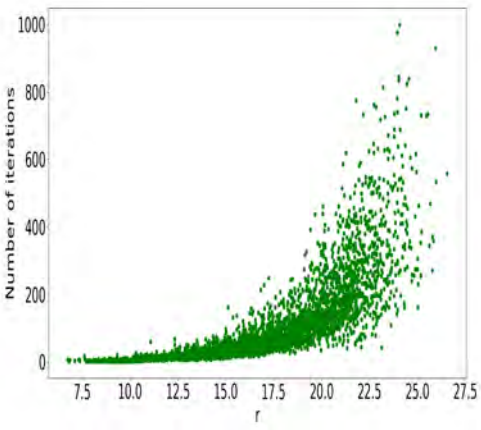
(b)



(c)



(d)



(e)

Figure III.8: Different scatterplots each involving the total number of iterations, which k-means took to converge, depending on another parameter (n, d, k, sp, r).

Data name	d	n	Description
birch	2	100000	a list of Gaussian clusters organised in a data space divided into 10x10 squares
covtype	54	150000	remote measurements of ground cover
cup98	56	95412	dataset proposed by KDD98
mnist50	50	60000	database of handwritten digits (MNIST), having undergone a random projection.
mnist784	784	70000	database of handwritten digits (original MNIST)
vehicle	100	98528	database of vehicles whose characteristics are obtained from seismic and acoustic sensors

Table III.4: Description of the datasets used to evaluate Sk-means

function is considered better. The LLoyd, Compare-means, Sort-means and Exponion algorithms all produce the same *sse* value.

III.2.5.5 Results analysis

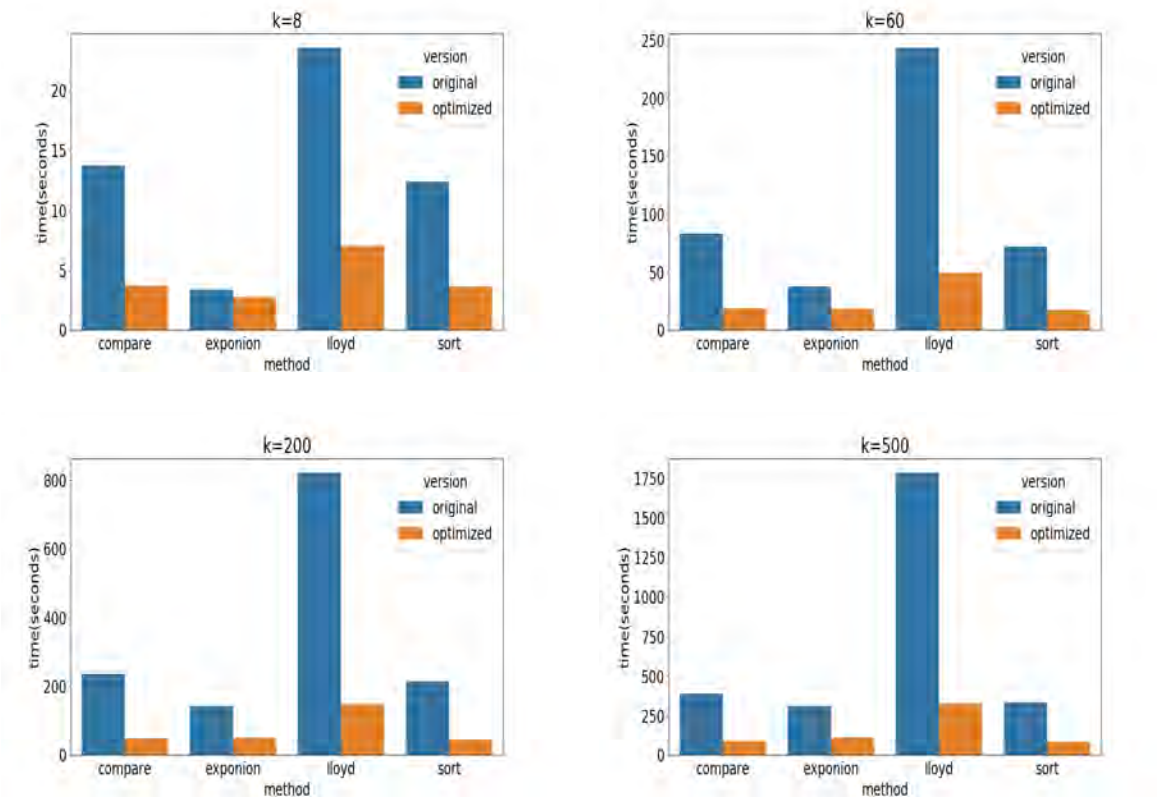


Figure III.9: Gain in execution time of the optimized version compared to the original version as a function of k.

In figure III.9, we show the gain brought by the optimized version compared to its original version. A first observation is that the larger k is, the more expensive the algorithms are. This is expected because the computational complexity of k-means increases when k increases. On all values of k, the original Lloyd's version is clearly the most expensive in execution time because it has no intrinsic optimization strategy.

Nevertheless, the gain brought by the Lloyd optimized version compared to its original version varies from 3.4 to 5.4 times. The original version of Exponion is the least expensive. The gain is from 1.2 to 2.8 times, brought its optimized version. The original and optimized versions of the compare-means and sort-means are more expensive than the Exponion (original and optimized). However, they tend to have the same acceleration level as the latter, as the k increases. At $k = 500$, the optimized Exponion (time=111s) is overtaken by the optimized versions of compare-means (time=87s) and sort-means (time=85s) . The gains of these two algorithms, on different values of k, are stable as for the other algorithms, ranging from 3.7 to 4.8 for compare-means and from 3.4 to 4.8 for sort-means. All in all, the versions benefiting from Sk-means bring an overall gain that is not negligible, especially when k is large.

This table III.5 compares the different versions of k-means compared to Lloyd's version in terms of execution time. The experiments' results, launched on the same data set and at the same value of k but different by the initialization of the centroids, are averaged in this table (in fact, the average of two experiments).

Oexponion and Osort remain, on average, the favorites for practically all datasets. They have benefited the most from the Sk-means strategy. In small and large dimensions with a very high k value, the gains brought by the optimized versions can be very significant (191× for {Oexponion,birch,$k = 500$}, 149× for {Osort,covtype,$k = 500$}, 97× for {Osort,cup98,$k = 500$}. On two datasets, when $k = 8$, Exponion is better than OExponion. In all other datasets, Oxponion is better, and in many cases, very much better. The other optimized versions are, in all cases, better than their original associated versions. The optimized algorithms are always better than Lloyd, which is not the case for the original compare and sort (mnist784). For very large dimensions, these two algorithms have difficulty to outperform Lloyd except slightly at $k = 500$. In this configuration, Oexponion outperforms all the others.

The quality mentioned above is measured from the relative difference. Let A be an algorithm to evaluate and B be a reference algorithm, the relative difference is equal:

$$\Delta sse = \frac{sse_A - sse_B}{sse_B} \tag{III.9}$$

The smaller the difference, the closer A and B are in terms of clustering results. In all cases (table III.6), in absolute value, Δsse is less than 0.01 (i.e. 1%) except for birch at $k = 200$ which is at 0.012 exceeding very slightly the 1%. The difference is null in several cases or even negative (at cup98, $k = 8$) i.e., slightly better quality for the optimized versions. For each dataset and k value, the optimized versions have almost the same rate of difference. This shows that the optimized versions converge almost at the same speed in terms of the number of iterations. Moreover, it is experimentally observed that there are always fewer iterations in the optimized versions than in the original versions.

		k			
data name	method	8	60	200	500
birch	compare	3.55	13.36	19.45	21.35
	exponion	18.03	33.57	58.25	80.46
	lloyd	1.00	1.00	1.00	1.00
	ocompare	23.14	95.93	133.62	113.08
	oexponion	18.96	76.29	**160.69**	**191.12**
	olloyd	7.03	7.96	7.17	5.41
	osort	**24.72**	**109.03**	135.64	105.60
	sort	4.13	26.12	61.31	75.38
covtype	compare	3.46	10.34	15.24	18.64
	exponion	**7.00**	16.63	23.90	32.32
	lloyd	1.00	1.00	1.00	1.00
	ocompare	5.10	30.88	73.04	111.16
	oexponion	4.02	23.63	53.04	92.80
	olloyd	1.56	3.12	4.91	6.31
	osort	5.65	**38.77**	**94.18**	**149.50**
	sort	3.79	15.90	32.85	61.07
cup98	compare	2.64	7.32	10.17	11.84
	exponion	4.66	4.46	5.83	7.74
	lloyd	1.00	1.00	1.00	1.00
	ocompare	7.05	22.11	57.07	87.23
	oexponion	5.81	7.61	19.20	31.11
	olloyd	2.77	3.39	6.21	8.03
	osort	**7.53**	**26.33**	**66.43**	**97.85**
	sort	2.88	9.59	14.97	22.71
mnist50	compare	4.54	5.89	6.58	7.21
	exponion	**16.65**	9.91	8.54	6.62
	lloyd	1.00	1.00	1.00	1.00
	ocompare	12.80	53.24	29.34	16.25
	oexponion	16.38	37.58	20.57	11.11
	olloyd	2.88	9.29	4.45	2.21
	osort	14.56	**60.25**	**32.82**	**18.05**
	sort	5.27	7.41	8.37	9.15
mnist784	compare	0.88	1.01	1.13	1.17
	exponion	2.90	4.03	3.12	1.99
	lloyd	1.00	1.00	1.00	1.00
	ocompare	1.75	5.18	5.34	4.78
	oexponion	**3.09**	**8.84**	**9.48**	**5.48**
	olloyd	1.83	5.33	6.05	4.03
	osort	1.74	5.23	6.22	4.78
	sort	1.04	1.19	1.13	1.19
vehicle	compare	0.98	1.47	1.66	1.88
	exponion	5.06	2.95	2.28	2.17
	lloyd	1.00	1.00	1.00	1.00
	ocompare	5.05	4.92	**7.64**	7.92
	oexponion	**10.51**	**5.50**	5.85	5.71
	olloyd	4.72	4.27	4.81	4.13
	osort	4.87	5.32	**7.64**	**7.99**
	sort	1.05	1.45	1.72	1.90

Table III.5: Number of times the algorithms are accelerated compared to Lloyd's version. Algorithms beginning with the letter o refer to algorithms that have benefited from the Sk-means strategy. The numbers in bold correspond to the best acceleration for a given dataset and k.

		k			
Data name	Method	8	60	200	500
birch	ocompare	0.003	0.009	0.012	0.008
	oexponion	0.003	0.009	0.012	0.008
	olloyd	0.003	0.009	0.012	0.008
	osort	0.003	0.009	0.012	0.008
covtype	ocompare	0	0.002	0.003	0.005
	oexponion	0	0.002	0.003	0.005
	olloyd	0	0.002	0.003	0.005
	osort	0	0.002	0.003	0.005
cup98	ocompare	-0.005	0.001	0.003	0.006
	oexponion	-0.005	0.001	0.003	0.006
	olloyd	-0.005	0.001	0.003	0.006
	osort	-0.005	0.001	0.003	0.006
mnist50	ocompare	0	0.008	0.002	0.001
	oexponion	0	0.008	0.002	0.001
	olloyd	0	0.008	0.002	0.001
	osort	0	0.008	0.002	0.001
mnist784	ocompare	0	0.001	0.001	0.001
	oexponion	0	0.001	0.001	0.001
	olloyd	0	0.001	0.001	0.001
	osort	0	0.001	0.001	0.001
vehicle	ocompare	0.001	0.001	0.001	0.001
	oexponion	0.001	0.001	0.001	0.001
	olloyd	0.001	0.001	0.001	0.001
	osort	0.001	0.001	0.001	0.001

Table III.6: Relative difference in SSE between each optimized algorithm and the LLoyd version.

III.2.6 Conclusion

K-means (Lloyd's version) is one of the best known and most used partitioning algorithms.

Nevertheless, it becomes more and more expensive to calculate as the number of points (n) and the number of clusters (k) increase. This is particularly true in the k-means first phase, which requires calculating the distance between each point and the k centroids. This is impractical in a large dataset environment. Therefore, k-means versions based on geometric reasoning have been developed to reduce point-centroid distance calculations in this phase. However, their computational complexity also increases as the n and k increase. Moreover, for each point, the variables (bounds) are updated at each iteration.

Therefore, we have developed Sk-means, a strategy that further reduces the number of distances between points and centroids. Its objective is to detect stable points as quickly as possible, i.e., points that will not change the cluster before the process end. For each cluster, we defined two different parts (kernel and outer part) to separate the dynamic points (i.e., very likely to change cluster) from those that are not or slightly dynamic. Indeed, the core concentrates the stable points and the passive points (points in the process of stabilization). On the other hand, the outer part contains dynamic

points. We have also designed an estimator (E) to decide whether the point is passive or dynamic.

The advantage of this strategy occurs when the kernel absorbs a point. In this case, this point is no longer involved in point-centroid distance calculations until the process end. This means that k distances are avoided at each iteration for this point. In addition, in versions based on geometric reasoning, the bounds associated with this point are no longer maintained and updated. The higher the number of points in the kernels, the more advantageous our strategy is.

Experimentally, k-means versions integrating Sk-means are up to 191 times faster than Lloyd's version. A version with Sk-means can be up to 5 times faster than its original non-optimized version.

Moreover, overall, the relative error between partitionings is below 1%. Therefore, the objective functions of k-means versions with Sk-means and those without Sk-means provide almost the same local solutions, i.e. i.e. the differences in terms of clustering results are negligible.

III.3 KD-means

Symbols	Meaning
n	number of data points
d	dimension of data points
X	$X = \{x_1, \ldots, x_n\}$ is a dataset, with $x_i \in \mathbb{R}^d$
k	number of clusters
C	$C = \{C_1, \ldots, C_k\}$, is a set of clusters,
G	$G = \{G_1, \ldots, G_k\}$, is a set of centroids, each cluster C_j is associated with its centroid G_j
$dis(x_i, x_j)$	dis is a distance between the points x_i and x_j
w	number of leaf nodes in the Kd-tree data structure
υ	index used to refer to a leaf node, or to iterate on a list of leaf indices annotated ϑ
h_j	it corresponds to a hyper-rectangle encompassing the points of cluster C_j

Table III.7: Symbols

III.3.1 Introduction

As we have pointed out, three limitations have been identified concerning estimators for estimating the number of clusters k, namely:

- the overlap of clusters in the ground-truth leads estimators to deviate exaggeratedly from real k;
- the execution time is long because k-means is called at each iteration, and this time is even longer when there is an overlap of clusters;
- estimators constrain clusters to have a particular shape (spherical, elliptical).

So, we propose Kd-means to address these limitations. First, we give a global overview of the steps of the solution in III.3.2. Then, we detail these steps corresponding mainly to the Kd-tree data structure and the information stored in it (III.3.3), the construction of the data structure (III.3.4), the initialization of Kd-tree leaves (III.3.5), the hierarchical merging of clusters (III.3.6) and the cluster regularization (III.3.7).

This is followed by experimental validation of Kd-means on large and varied datasets in III.3.8. The Kd-means robustness against its competitors is demonstrated in terms of k estimation, the quality of clustering (similarity between partitioning and ground-truth) and execution time.

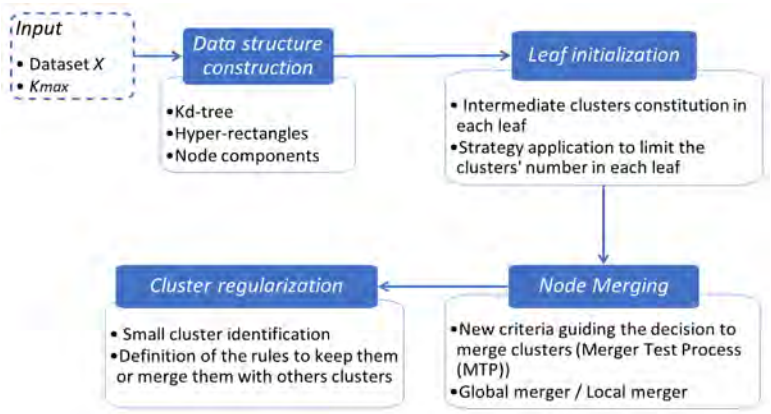


Figure III.10: Overall solution for estimating the value of k and clustering.

III.3.2 Overview

The solution is composed of three main parts (figure III.10):

- the storage and organization of the dataset X provided by the user in a data structure (Kd-tree);
- the processing is done on this structure to estimate k (Merging task);
- optional improvement of the results of the previous step (Cluster regularization).

We opted for the Kd-tree (Bentley, 1975) data structure to fulfill the roles of storage and data organization. A Kd-tree is a binary tree representing a hierarchical subdivision of space using splitting planes that are orthogonal to the dimensions [1]. In Subsection III.3.3.1, we discuss the advantages of Kd-tree as well as the method that builds the Kd-tree in III.3.3.

In the processing part, first of all, each leaf's points are processed independently from the points on the other leaves (see III.3.5). This process results in several clusters in the leaf. As a result, in each leaf, several clusters are constituted. Then the clusters of the nodes are merged recursively from the leaves to the root according to rules and algorithms defined in part III.3.6. These are built to manage cluster shapes as well as cluster overlap.

The final step is a regularization phase (part III.3.7). It detects small clusters and decides whether they remain separate from other clusters, or they should be merged with other clusters. This phase avoids having an overestimation of k due to small clusters.

[1] do not confuse the k of Kd-tree, which is just the dimension of the data stored in the tree while the k of k-means which corresponds to the number of clusters

III.3.3 Data structure elements

III.3.3.1 Kd-tree

Kd-tree puts points that are spatially close together in the same node (Bentley, 1975). This property is exploited by several machine learning methods to accelerate calculations related to point space. One of the best-known cases is the k-closest neighbor's algorithm (Cover and Hart, 2006). This property is advantageous to us in two cases. First of all, it partly addresses the problem of k-means because it groups the most similar points possible in the same node. Second, it provides an optimized spatial subdivision of space to accelerate data processing. It recursively splits the whole dataset into partitions, like a decision tree acting on numerical datasets (Gan et al., 2007). The root node represents the whole data space, while the leaves are the smallest possible partitions of the data space. So, a node is either a leaf, i.e., a node without child nodes, or an internal node, i.e., a node with two child nodes.

III.3.3.2 Node components

To allow the node merging step (III.3.6) to run, all nodes each have at least one cluster. Note that the clusters are formed in the leaf nodes during the leaf initialization step (III.3.5). On the other hand, in the internal nodes, they are dynamically constituted during the node merging step. Each cluster is associated with:

- the indexes referring to its points. Therefore, the data points are not stored in the node (by extension in the whole tree) but are accessible via the indexes;
- the arithmetic mean of its points, representing its centroid;
- the sum of the squared deviations (between its points and its centroid), also called sse;
- its hyper-rectangle, a geometrical object used because of its advantages in terms of set calculation and set representation in Kd-means. This object is detailed in the part III.3.3.3. Note that in each node, we annotate ζ its list of hyper-rectangles.

The internal nodes also have, in particular, the splitting dimension index sd as well as the associated value sv. These two values are used in the construction phase of the Kd-tree (part III.3.4).

III.3.3.3 Hyper-rectangle

As outlined above, each cluster is associated with a hyper-rectangle. The latter is a necessary element for the node merging step. Formally, it corresponds to the smallest

possible rectangle (generalized to n dimensions) covering all the data points of a set. It is defined by the following equation:

$$H = \{x_i | \forall i \; x_{mn} \leq x_i \leq x_{mx}\} \tag{III.10}$$

where, mn and mx respectively represent indexes of the lower ($mins$) and upper ($maxes$) bounds of the hyper-rectangle. Indeed, $mins$($maxes$) is a vector corresponding to the minimum (maximum) values of each dimension of a cluster's points(see figure III.11).

In our case, we consider a hyper-rectangle as a geometric object to approximate the overall spatial distribution of a set of points contained in the node. In other words, each of the clusters of a node is geometrically represented by a hyper-rectangle. From this representation results, in KD-means, in time-saving on the calculations that involve sets (a set is a cluster of points). For example, to calculate a distance between two sets or perform a set operation involving at least two sets, their corresponding hyper-rectangles could be used. Therefore, instead of visiting all the data points of the sets, it is only necessary to use the $mins$, and $maxes$ vectors of the hyper-rectangles for the above calculations. This greatly saves much time on large amounts of data.

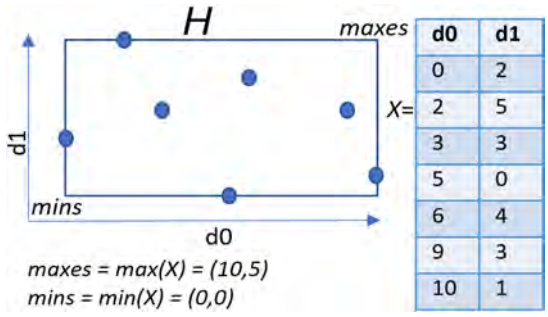


Figure III.11: A 2-dimensional hyper-rectangle H and the associated bounds $maxes$ and $mins$ located on the corners (upper right and lower left). The dataset has two dimensions ($d0$ and $d1$). The circles represent the points of the dataset. The max (resp. min) function returns the maximum (resp. minimum) of the values in each dimension. Although the example deals with a two-dimensional space, the hyper-rectangle can also be defined in a space of more than two dimensions.

III.3.4 Data structure construction

In this part, we focus on the strategy we adopted to build Kd-tree.

The Kd-tree is built from top to bottom. In the beginning, Kd-tree contains only one node (root), then from this one, two child nodes are formed. Then the nodes are recursively divided until the leaves are created. Each child node represents a sub-space of points. The union of the sub-spaces of both child nodes corresponds to the point space of their father. In the following, we focus on the splitting strategy of the internal nodes.

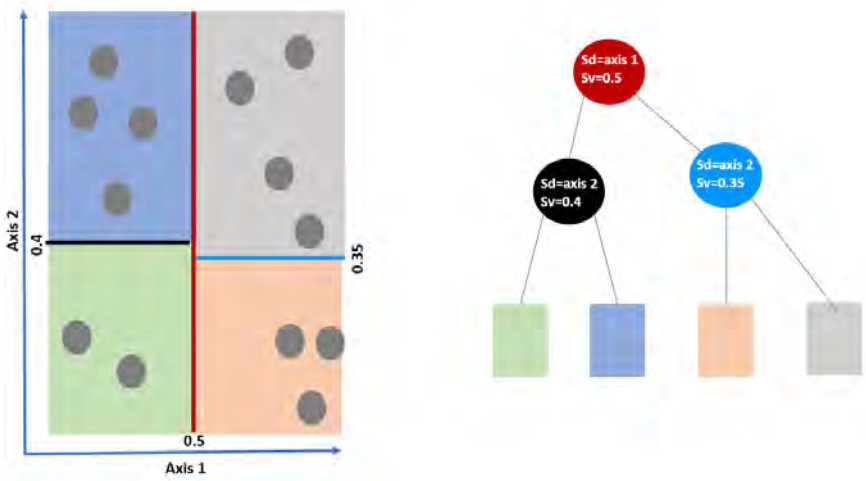


Figure III.12: Example of the Kd-tree instantiation method on a two-dimensional space. The method is also applicable to spaces of more than two dimensions.

The partitioning of a data space of an internal node (i.e., not the leaf) is performed mainly based on two cutting elements that must be specified: a given dimension (sd) of the node data space and a value of this dimension (sv). Thus the points whose value at the dimension sd is less (resp. greater) than sv, then they are assigned to the child node, which is called *lesschild* (resp. *greaterchild*). This process is carried out recursively from the root to the leaves.

The figure III.12 illustrates this process through an example. On the left of the figure, a two-dimensional data space (axis 1 and axis 2, axis and dimension are used interchangeably) and the right the associated Kd-tree. At the beginning of the process, Kd-tree contains only the root. To divide it in two, the dimension "axis 1" is assigned to sd and the value 0.5 to sv. As a consequence, the data space is divided into two at the vertical red line. A *lesschild* node (resp. *rightchild* node) is instantiated to represent the points which are to the left (resp. to the right) of the red line (i.e., their value on axis 1 is less (resp. greater) than 0.5). At this level, *lesschild* represents only the subspace on the left and *rightchild* only the subspace on the right. If we focus at the level of the left child of the root, $sd = axis2$ and $sv = 0.4$, the corresponding subspace is divided in two at the black horizontal line. From this node, two child nodes result. The same procedure is used for the right thread of the root.

Several rules for splitting nodes are proposed to choose the optimal dimension and associated value for correct data separation. Among them, we opted for *sliding midpoint splitting rule* (Samet, 2005) because it provides an optimized data organization than other classical rules (Maneewongvatana and Mount, 1999). This performance on others is explained because:

- it does not produce empty nodes or nodes whose data space is very sparse (i.e., a very large space, specifically at the sides, compared to the data it represents when it should be small);

- it is less expensive. Indeed, the classical rules choose the median as the cutting value sv. In contrast, the sliding midpoint splitting rule chooses the middle of the points $((max + min)/2)$, which is less expensive. The dimension sd chosen is the one that is the longest $(max - min)$.

Note even with this splitting rule, theoretically there is no guarantee on the limit of the depth that Kd-tree can have, the trees could be very deep (so the time of tree construction is increased). As a result, we have added two stopping conditions to avoid deep trees. These two conditions are performed at the beginning of the method that partitions the node in two. If one of the conditions is verified then the node is not divided and it is considered as a leaf:

- the depth of the leaf is limited to $log(n)$ to save construction time compared to the normal time taken if the depth is not limited. The root has a depth of 1;

- each node is limited by a minimum number of points annotated ψ. It is not interesting to have leaves with a number of points close to one in a context of massive data. This would result in a very long Kd-tree construction time and therefore make our solution unsuitable for near real-time applications. Besides, if the sum of the number of points of the node is less than $2 * \psi$ then it is a leaf. Indeed, two cases are possible if this limit is not defined. Either the two children will be leaves if their number of points (size) is less than ψ. Either the size of one will be greater than ψ and therefore the other is a leaf. In the latter case a difference in depth will occur which brings a certain imbalance of the tree. This condition limits the number of small leaves and the depth of the tree, and contributes to the tree's balance.

III.3.5 Leaf initialization

After building the Kd-tree, we estimate in each leaf (we annotate υ the leaf node's index) the number of clusters. At the same time, the clusters are produced. For this we have adapted G-means (Hamerly and Elkan, 2003) called *estimateK* (see Algorithm 2). It takes into account the problem related to G-means concerning the exaggerated overestimation of the number of clusters in the case of overlapping. Then, the *estimateK* algorithm produces a limited number of clusters in the leaves. Indeed, at each iteration t, *estimateK* produces k_t clusters. We introduce ρ a threshold that controls the maximum number of clusters to produce. Let w be the number of leaves that Kd-tree has, ρ is then defined as follows:

$$\rho = \begin{cases} \sqrt{k_{max}}, & \text{if } w \geq \sqrt{k_{max}} \\ \frac{k_{max}}{w}, & \text{otherwise} \end{cases} \tag{III.11}$$

Algorithm 2 InitializeLeaf

Input: k_{max},X,υ
Output: ζ

1: $LD \leftarrow x_\upsilon \subset X$
2: $\Gamma, k, I \leftarrow estimateK(LD, k_{max})$
3: { I is defined as $\{I_j | j = 1..k\}$ with I_j the set of point indices of cluster j and Γ set of centroids.}
4: {For each cluster, a hyper-rectangle is instantiated as well as its sse is calculated.}
5: **for** $j = 1$ to $j = k$ **do**
6: $\quad \zeta_I^j \leftarrow I_j$
7: $\quad \zeta_G^j \leftarrow \Gamma_j$
8: $\quad \zeta_{maxes}^j \leftarrow max(LD_{I_j})$
9: $\quad \zeta_{mins}^j \leftarrow min(LD_{I_j})$
10: **end for**

As soon as $k_t \geq \rho$ then k_t is chosen for the leaf. Note that k_t could be up to 2^t. This case occurs when all centroids have been divided into two new centroids because they have not been evaluated as Gaussian. Note that even if the total number of clusters in all leaves exceeds k_{max}, then our merging algorithms bring this number closer to the real k of the dataset.

If we assume that in equation III.11, only the second condition exists, this would cause ρ to tend towards a value of 1 or 0. Indeed for a given value of k_{max}, the greater w is the greater the ρ tends towards 1 if $w \leq k_{max}$. When $w > k_{max}$ then ρ tends towards 0. These cases could make the solution ineffective because they would underestimate the number of clusters in the leaf and, by extension, underestimate the number of clusters in the tree dataset. The first condition is essential to balance between having an underestimation of k if there is only the second condition and having an overestimation of k if there is no condition that would limit the value of k.

After estimating k in the leaf υ, each cluster produced is represented by a hyper-rectangle along with the associated aggregate calculations (arithmetic mean (G) and the sum of the squared differences between the points and G (sse)).

III.3.6 Node merging

The leaf initialization step produces clusters of points in each leaf. These clusters do not necessarily represent the final clusters, i.e., the clusters that the user has as output. In order to obtain the final clusters, the leaves' clusters are used as inputs in a hierarchical process from bottom to top.

In order to obtain final clusters, we have developed an algorithm called global merger. It consists in recursively merging clusters from the leaves to the root. The final clusters are indeed in the root. The nodes are processed according to the post-

fixed path, i.e., each node is processed after each of its children is processed. Indeed, processing a node consists in executing another algorithm, l̂ocal merger, on its clusters. The purpose of this algorithm is to merge as many clusters as possible in a given node. The l̂ocal merger result contains at least:

- new clusters resulting from the merge if at least two clusters have been merged;
- clusters that may not have been merged.

So, for a given node, when the results (clusters) are produced in its child nodes, then l̂ocal merger is applied on this node. In this case, the algorithm has as input the child results. This merger operation is repeated recursively from child nodes to the parent node. And this, up to the root (see the example III.13 for more clarity).

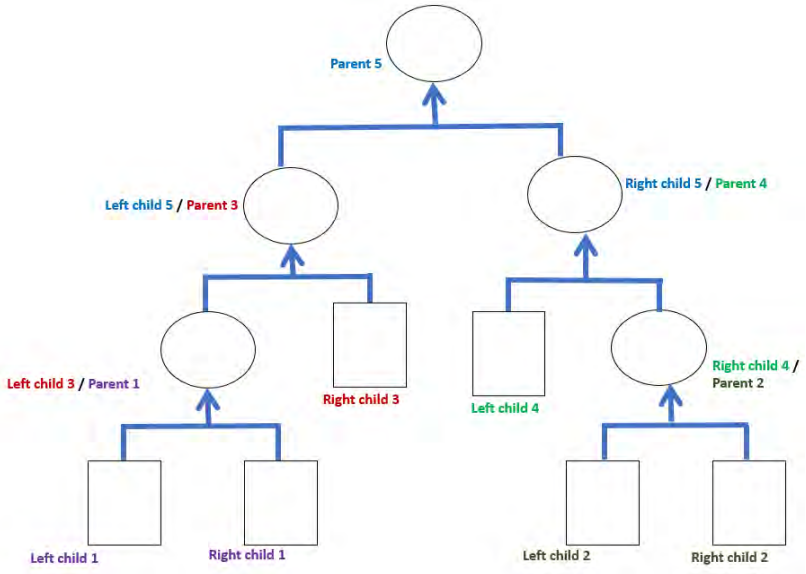


Figure III.13: Process of hierarchical bottom-up merging of clusters. Square shapes are leaves, while circle shapes are internal nodes. The merging process starts with the leaves and gradually and hierarchically reaches the root. The leaves only play the role of children while the internal nodes are first parents and then children. Each color, involving three nodes, represents a fusion operation. The child nodes are processed if they are not already, then the father node is processed in turn using the results of its child nodes.

To summarize, the ĝlobal merger algorithm visits the nodes according to the post-fixed path. At each visit, the l̂ocal merger algorithm is executed on a set of clusters. In this one, we apply the merger test process per cluster pair (figure III.14).

In the following, we introduce some definitions (part III.3.6.1) in order to detail the hierarchical merging process (part III.3.6.2).

In the following, we first introduce definitions used by the local merger algorithm (by extension also by the merge test process). Then we discuss the properties of the mergeability concept, i.e., the properties that guide the decision to merge or not a cluster pair. Then we detail the merge test process (using the mergeability concept) then the local merger algorithm.

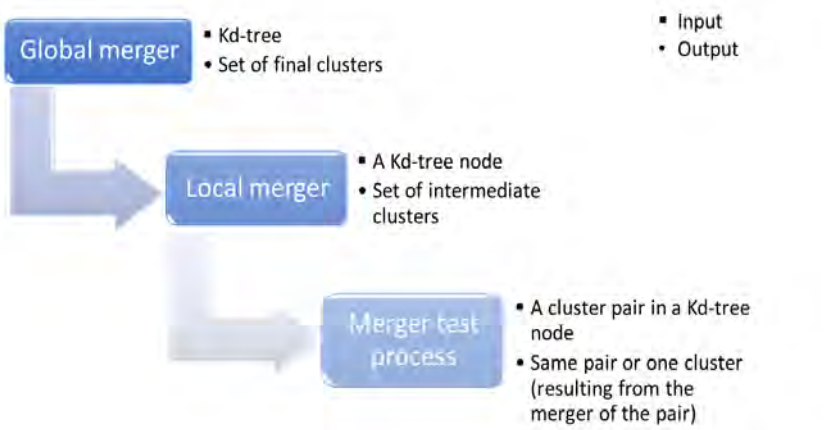


Figure III.14: Merger test process (MTP) is nested in the local merger algorithm, which in turn is nested in the global merger algorithm. MTP could be called multiple times during local merger execution in a node.

III.3.6.1 Definitions

Definition III.3. Let ζ be a set of hyper-rectangles, the *maxes* (resp. *mins*) associated with ζ is a vector where the ith element corresponds to the maximum (resp. minimum) value of the ith dimension of the *maxes* (respectively *mins*) vectors of *zeta*:

$$\theta(\zeta) = \begin{cases} mins = min(\{\zeta_h.mins | h \in [1...card(\zeta)]\}) \\ maxes = max(\{\zeta_h.maxes | h \in [1...card(\zeta)]\}) \end{cases} \tag{III.12}$$

with $card()$ a function that returns the number of elements of a set, and $min()$ (resp. $max()$) a function that returns the minimum (resp. maximum) value of a given set.

Definition III.4. Let $P = \{(u,v) | u,v \in \mathbb{N} \cap [1;k] \text{ and } u \neq v\}$ a list of index pairs of hyper-rectangles of a node. We consider the index pairs (u,v) and (v,u) equivalent and consequently P contains only one of them. The Average of the Pairwise Distances is calculated by the following function:

$$apd(\zeta) = \frac{\sum_{(u,v)\in P} ||G_u - G_v||}{card(P)} \tag{III.13}$$

with G_u and G_v the centroids, respectively, of the points of the hyper-rectangles h_u and h_v.

Definition III.5. The minimum distance between two hyper-rectangles h_u and h_v is calculated as follows:

$$minH(h_u, h_v) = ||0 - f|| \tag{III.14}$$

where $f = max(\{0, max(\{h_u.mins - h_v.maxes, h_v.mins - h_u.maxes\})\})$.

Lemma III.6. Given two datasets $L \subseteq \mathbb{R}^d$ and $M \subseteq \mathbb{R}^d$ with $n_1 = card(L)$ and $n_2 = card(M)$, if the sum of squared errors (sse) of L and M are as follow:

$$q_l = \sum_{i=1}^{n_1} ||L_i - \bar{L}||^2$$

$$q_m = \sum_{i=1}^{n_2} ||M_i - \bar{M}||^2$$

then the sum of squared errors of $B = L \cup M$ is:

$$q_b = q_u + q_m + n_1||\bar{L} - \bar{B}||^2 + n_2 * ||\bar{M} - \bar{B}||^2 \tag{III.15}$$

where $\bar{B}$ is computed as the average of the weighted averages

$$\frac{n_1\bar{L} + n_2\bar{M}}{n_1 + n_2} \tag{III.16}$$

The formula III.15 allows calculating the sse of B from the sse of L and M [2]. It represents a considerable gain because there is no need to iterate on the points of B.

Proof. We are trying to calculate the sum of two sse:

$$q_b = \underbrace{\sum_{i=1}^{n_1} ||L_i - \bar{B}||^2}_{q_{lb}} + \underbrace{\sum_{i=1}^{n_2} ||M_i - \bar{B}||^2}_{q_{mb}}$$

Let focus on the first term of this equation:

$$q_{lb} = \sum_{i=1}^{n_1} ||L_i - \bar{B}||^2 = \sum_{i=1}^{n_1} ||(L_i - \bar{L}) + (\bar{L} - \bar{B})||^2$$
$$= \sum_{i=1}^{n_1} ||L_i - \bar{L}||^2 + 2\left(\sum_{i=1}^{n_1}(L_i - \bar{L})\right)(\bar{L} - \bar{B}) + n_1||\bar{L} - \bar{B}||^2$$

But:

$$\sum_{i=1}^{n_1}(L_i - \bar{L}) = \sum_{i=1}^{n_1} L_i - \sum_{i=1}^{n_1} \bar{L} = n_1\bar{L} - n_1\bar{L} = 0$$

So $q_{lb} = q_l + n_1||\bar{L} - \bar{B}||^2$. If we repeat the same calculation for the second term q_{mb} then we obtain equation III.15.

[2] We consider the sse of L and M respectively q_l and q_m already calculated, which is the case because the sse are calculated entirely only in the leaves, but in the internal nodes the formula III.15 is used.

III.3.6.2 Mergeability concept

We focus on the detection of clusters that k-means could detect (i.e., clusters close to strict sphericity) and less spherical clusters present in real data. So our goal is, first, to capture natural clusters as defined in section III.1, i.e., clusters with spherical, elliptical or more complex shapes, e.g., curved elliptical shapes. Secondly, to identify these clusters even if they overlap with each other. To achieve these two objectives, it is necessary to define the properties that lead to consider that two clusters can be merged. We have set two properties to allow us to evaluate the mergeability between any two clusters, i.e., to determine the certainty level of the hypothesis that two clusters belong to the same cluster is true. The first one is connectivity. It corresponds to the level of proximity between the masses of points of two clusters. A mass of points is the smallest sub-space of a cluster concentrating the majority of the points practically. The second property corresponds to the spatial proximity between the two clusters without necessarily their masses being close. The stronger the connectivity and proximity between two clusters, the stronger the mergeability. Proximity alone is not enough to state that two clusters are mergeable because two clusters may be contiguous or overlapping without their two masses being close to each other. In this case, connectivity is necessary.

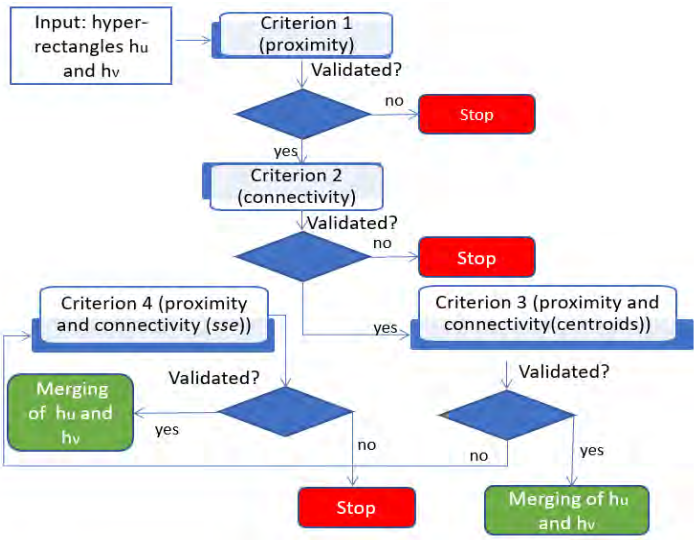


Figure III.15: criteria-based merging process between two hyper-rectangles x and y. This process is called merger test.

III.3.6.3 Merger test process (MTP)

We define a set of four criteria (represented as inequalities) to evaluate how much two clusters correspond to the same cluster and therefore, to decide whether two clusters can be merged following the properties that we have mentioned. So, these criteria involve measures of proximity and connectivity between clusters. Note that

the verification of the hypothesis that two clusters must be merged is done via their hyper-rectangles h_u and h_v. As shown in figure III.15, the sequence of these criteria forms the merge test process. This test verifies if the clusters of h_u and h_v belong to the same natural cluster. The test follows a precise order of the criteria. First of all, the first criterion must be validated before moving on to the second criterion; otherwise, the process stops. Similarly, the second criterion must be validated to activate the third criterion. If the latter is not checked, then the fourth criterion is triggered. It is therefore not necessary that the third criterion is validated to call the last one.

Note that merging several clusters means merging their associated hyper-rectangles. This also involves calculating the weighted average of the already calculated averages (of the point values) of the hyper-rectangles; concatenating the sets of indices; calculating the *sse* according to the equation III.15; and calculating the *mins* and *maxes* vectors according to the equation III.12.

Criterion 1. Let *mins* and *maxes* the limits of a defined data space $\in \mathbb{R}^d$ and ζ the hyper-rectangles set belonging to this space then two clusters represented by the hyper-rectangles $h_u \in \zeta$ and $h_v \in \zeta$ are possibly mergeable if:

$$\Delta G < \frac{\tau}{\epsilon} \tag{III.17}$$

$$\text{where } \tau = apd(\zeta),\ \Delta G = ||x_G - y_G|| \text{ and } \epsilon \in \mathbb{N}^*.$$

Criterion 1 is based on the proximity measure. It checks whether two hyper-rectangles are close enough to be possibly considered mergeable. It is based on the average of the pairwise distances between centroids (*apd*) to approximate the average distance between clusters of a node without calculating the distances between all point pairs of all clusters. The pairwise distances reduce the bias brought by the very distant centroids from the majority of other centroids. The value of ϵ controls the boundary between the qualifiers "distant" and "close" when referring to the distance between clusters of h_u and h_v. If this criterion is verified, then other criteria must be verified to confirm the fusion between h_u and h_v. Note that the higher the value of ϵ is, the fewer hyper-rectangles validate the criterion. Moreover, if $\epsilon = 1$, then a significant amount of hyper-rectangles will validate the criterion. This could lead to further unnecessary tests and probably to a non-merger result because if two clusters are very far apart, then there is no merger. The ϵ value is to be adjusted according to the strictness level of the "enough close" notion required by the criterion.

Criterion 2. Let h_u and h_v two hyper-rectangles, ΔC the distance between the centers of h_u and h_v and ΔG the distance between the centroids of the clusters C_u and C_v. If $\Delta G < \Delta C$ then h_u and h_v are possibly mergeable.

Criterion 2 refines the set of hyper-rectangles from the first criterion. It estimates how strong the connectivity between the two clusters is. We know that a hyper-

rectangle is the smallest rectangular envelope covering all the data points in a cluster. We could then decide that the center of the hyper-rectangle would correspond approximately to the centroid of the cluster if the cluster points are uniformly distributed with some sphericity. So the cluster with these characteristics is probably a natural cluster apart. As a result, the closer the distance between centroids (ΔG) is to the distance between centers (ΔC), the smaller the probability of merging h_u and h_v. This probability is considered null when $\Delta G \geq \Delta C$. Figure III.16 illustrates an example that validates this criterion.

The first two criteria make it possible to identify relatively close clusters. However, they also consider that two clusters are distant if their centroids are also distant, even if the two clusters are contiguous. The latter could occur if the majority of points surround the centroid while a minority is far from the centroid. However, these criteria are based only on the distances applied to centroids and centers. Two clusters could be considered as close, but one or more clusters could be located between them. In this case, they cannot be merged directly. They could only be if at least one cluster between them merges with them. The following two criteria require stricter distances and more robust connectivity to avoid directly merging two nearby clusters separated by other clusters.

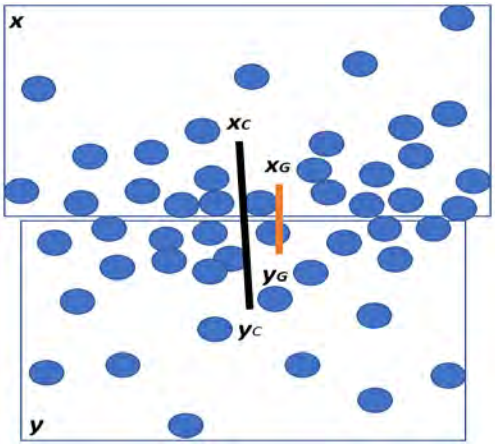


Figure III.16: Example of the situation that validates the criterion 2. Blue circles are data points. The orange and black lines represent respectively, the distance between centroids and the distance between centers of the hyper-rectangles x and y.

Criterion 3. Let be h_u and h_v two hyper-rectangles. We merge h_u and h_v when Criteria 1 and 2 are verified and if $minH(h_u, h_v) = 0$ and $\Delta G < \Delta C * \lambda$ with $\lambda \in]0; 1]$

Criterion 3 only deals with contiguous or overlapping hyper-rectangles. Indeed $minH(x, y)$ returns 0 if the two hyper-rectangles overlap or if their distance is zero. Also, a connectivity constraint is added. It results in a constraint on the proximity between centroids: for two hyper-rectangles h_u and h_v, the distance between centroids must be less than $\Delta C * \lambda$. The smaller λ, the more the notion of "close enough" between clusters to match the same cluster is strict.

Criterion 4. Let h_u and h_v two hyper-rectangles and $\epsilon \in \mathbb{N}^*$. We merge h_u and h_v when:

- Criteria 1 and 2 are verified,
- $minH(x, y) \leq \frac{\tau}{\epsilon}$,
- $evo \leq th\_evo \in \mathbb{R}^+$ where:

$$evo = \frac{|sse_{u\cup v} - (sse_u + sse_v)|}{sse_{u\cup v}} \tag{III.18}$$

with sse is the sum of the squared errors and $sse_{u\cup v}$ the sse of the C_u and C_v data points union

Unlike Criterion 3, to validate Criterion 4 hyper-rectangles are not necessary to be contiguous but a maximum distance is required. Two clusters may not be contiguous or maybe nearly contiguous. A threshold on the distance between them is necessary to prevent a cluster from interposing between them. This results in the following inequity: the distance $minH(x, y)$ must be less than $\frac{\tau}{\epsilon}$. The criterion also adds a requirement on the compactness of C_u and C_v clusters. To do this, it uses the homogeneity measure called sum of square error (sse). In this criterion, we choose to limit the quantity of sse for possible mergers. Indeed, two clusters are merged only if their evo value is less than the limit is th_evo. The latter is entered by the user. However, th_evo is adapted according to the value of Δk provided Equation III.19. The latter evaluates the difference between the value of k_{max} and the total sum of the values of k specific to each leaf ($\sum_{v\in\vartheta} k_v$) with ϑ the index list of all Kd-tree leaves. The purpose of this adaptation is to approach the estimated k value on the entire dataset at k_{max} and avoid an overestimation of k compared to k_{max}. If $\delta k \geq 1$ it means $\sum_{v\in\vartheta} k_v$ is at least equal to k_{max} and therefore th_evo will be unchanged. Otherwise, if $k_{max} > \sum_{v\in\vartheta} k_v$ then th_evo is recalculated according Equation III.20.

$$\delta k = |1 - \frac{k_{max}}{\sum_{v\in\vartheta} k_v}| \tag{III.19}$$

$$th\_evo = \begin{cases} th\_evo, & \text{if } \delta k \geq 1 \\ th\_evo + \delta k, & \text{otherwise} \end{cases} \tag{III.20}$$

III.3.6.4 Local merger algorithm

.

Algorithm 3 (l͡ocal merger) allows us to merge clusters whose representative hyper-rectangles are included in the ζ set. ζ hyper-rectangles are associated with a list of

Algorithm 3 Local merger

Input: ζ, τ, th_evo

Output: ζ

$merged \leftarrow True$
$it \leftarrow 0$
$old\_card\_lh \leftarrow card(\zeta)$
while $merged = True$ **do**
  $h_u \leftarrow \zeta_{it}$
  $T \leftarrow \{it\}$
  $tmp\_indexes \leftarrow \{0, ..., old\_card\_lh\}$
  $tmp\_indexes \leftarrow tmp\_indexes \backslash \{it\}$
  for $i \, in \, tmp\_indexes$ **do**
    $h_v \leftarrow \zeta_i$
    $\Delta G \leftarrow ||G_u - G_v||$
    {Criteria 1}
    if $\Delta G < \frac{\tau}{\epsilon}$ **then**
      $\Delta C \leftarrow ||\frac{h_u.mins + h_u.maxes}{2} - \frac{h_v.mins + h_v.maxes}{2}||$
      $e \leftarrow False$
      {Criteria 2}
      if $\Delta G < \Delta C$ **then**
        $minDistanceXY \leftarrow minH(h_u, h_v)$
        {Criteria 3}
        if $minDistanceXY = 0$ and $\Delta G < \Delta C * \lambda$ **then**
          $e \leftarrow True$
        end if
        {Criteria 4}
        if $e = True$ and $evo \leq th\_evo$ and $minDistanceXY \leq \frac{\tau}{\epsilon}$ **then**
          $e \leftarrow True$
        end if‘
        if $e = True$ **then**
          $T \leftarrow T \cup i$
        end if
      end if
    end if
    if $card(T) > 1$ **then**
      $\zeta \leftarrow merge(T)$
      {The new hyper-rectangles are placed at the end of the list}
    end if
    $new\_card\_lh \leftarrow card(\zeta)$
    if $old\_card\_lh < new\_card\_lh$ **then**
      $old\_card\_lh \leftarrow new\_card\_lh$
      $it \leftarrow 0$
    else
      if $it + 1 < new\_card\_lh$ **then**
        $it \leftarrow it + 1$
      else
        $merged \leftarrow False$
      end if
    end if
  end for
end while

ascending ordered numerical indices. Thus the indices of the first and last elements of ζ are respectively 0 and $card(\zeta) - 1$. We consider h_u to be a temporary variable of hyper-rectangle type that runs through the ζ elements. The algorithm process is as follows. First, we assign the first element of ζ to h_u. This one is now a hyper-rectangle representing a cluster. We then check if all the other clusters represented by hyper-rectangles h_v are mergeable with the cluster represented by h_u. If h_u is not mergeable to any of them, then the next hyper-rectangle in ζ is assigned to h_u. On the other hand, if it is mergeable to at least one hyper-rectangle, then the concerned hyper-rectangles merge, resulting in a new hyper-rectangle (merging operated by the *merge* function). In this case, the first item of freshly reconstituted ζ is reassigned to h_u. This procedure is repeated until h_u is the second last element of ζ, and there are no more mergers. At the level of the fusion test process, the first two criteria only select the hyper-rectangles candidates h_v for the merger. At the same time, they reduce the number of hyper-rectangles to be processed in the other two criteria. While the last two decide whether to merge h_u and h_v or not.

III.3.6.5 Global merger

Algorithm 4 Global merger

Input: *internal*,*th_evo*
Output: ζ

1: $\zeta \leftarrow internal.less.\zeta \bigcup internal.greater.\zeta$
2: $mins, maxes \leftarrow \theta(\zeta)$
3: **if** card(internal.greater.ζ)> 1 **then**
4: $\quad \tau \leftarrow apd(internal.greater.\zeta)$
5: $\quad internal.greater.\zeta \leftarrow localMerger(internal.greater.\zeta, \tau, th\_evo)$
6: **end if**
7: **if** card(internal.less.ζ)> 1 **then**
8: $\quad \tau \leftarrow apd(internal.less.\zeta)$
9: $\quad internal.less.\zeta \leftarrow localMerger(internal.less.\zeta, \tau, th\_evo)$
10: **end if**
11: $\zeta \leftarrow internal.less.\zeta \bigcup internal.greater.\zeta$
12: $\tau \leftarrow apd(\zeta)$
13: $\zeta \leftarrow localMerger(\zeta, \tau, th\_evo)$

The *global merger* fusion algorithm can be seen as a layer that envelops the algorithm *local merger*. Intuitively, for a given internal and parent node, a good part of the clusters of one of its children is closer to each other than to the other child's clusters. So we process the clusters of a child node locally but in the data space of the parent node. Hence the proposal of the *global merger* algorithm. It performs the merging of clusters node by node using the *local merger* algorithm. It starts first with the children of the "*internal*" node and then concatenates the hyper-rectangles lists of both children to form the list of hyper-rectangles of the *internal* node. Finally, *local merger*

is applied to *internal*. On the other hand, the value of *apd* changes from one node to another because it depends only on the concerned node's clusters. Consequently, if two clusters have not been merged into one of the two children nodes, they could be merged into the parent node.

III.3.7 Cluster regularization

This subsection discusses the strategy we have defined to identify clusters that should not be as such but rather be part of another cluster. The previous step algorithms could produce small (in a number of points) but compact clusters. A small cluster is located in three cases: either it is a natural cluster different from the others, or it is part of an agglomeration of small clusters that represent a single cluster, or it is part of a large cluster. The definition of a small cluster has no objective purpose. We consider the definition of the small cluster resulting from the work (Ailon et al., 2013) as a cluster with a number of points less than $\sqrt{n}$ with n the size of the cluster.

We have developed an algorithm that makes it possible to match a small cluster to one of the three cases. The goal of the algorithm is to get as many natural clusters as possible. Knowing that our solution's final result is in the root of the tree, then this algorithm is unrolled only in this one. The algorithm consists of two parts that are executed consecutively once. The first identifies the small clusters and then merges those that are close to each other. The second re-identifies small clusters from the new list of small clusters from the first part and affects those close to large clusters. If a small cluster has not undergone a merger operation in both parts, it is considered a separate natural cluster.

One of the reasons for the presence of small clusters is due to Criteria 3 and 4 where the limit of the minimum distance between hyper-rectangles is very strict. Indeed, if the minimum distance between two hyper-rectangles, at least one of which is a small cluster, is greater than this limit, they cannot be merged. This limit is respectively equal to 0 in Criterion 3 and $\frac{\tau}{\epsilon}$ in Criterion 4. The $\frac{\tau}{\epsilon}$ limit is more flexible than the first one. It was used in both parts of the algorithm to define the boundary between "close" and "distant" w.r.t the distance between hyper-rectangles.

III.3.8 Experimental assessments

III.3.8.1 Introduction

We compare Kd-means to known methods of estimating k from literature (X-means (Pelleg and Moore, 2000) and G-means (Foina et al., 2011, Hamerly and Elkan, 2003)). We show that our solution is faster than current methods while ensuring better clustering quality on massive data. The datasets used are various (synthetic and real). We also show its efficiency through a use case on a hyperspectral image. Moreover, we give recommendations on the values that the parameters of our solution could take (the minimum size of an internal node ψ and limit th_evo of Equation III.20).

III.3.8.2 Experimental protocol

Our solution and the compared algorithms were implemented in python 3.6. Moreover, k-means++ (Arthur and Vassilvitskii, 2007) is used for initializing the centroids before applying Elkan algorithm.

Different values are assigned to parameters k,d, n, th_evo and ψ to study the sensitivity of our algorithm to these parameters. All combinations of the values of these parameters have been tested:

- $k \in [10, 32, 80]$
- $d \in [4, 6, 8]$
- $n \in [1 \times 10^6, 2 \times 10^6, 4 \times 10^6]$
- $th\_evo \in [0.05, 0.10, 0.15, 0.20, 0.25, 0.30]$
- $\psi \in [4, 8, 10, 12]$

The parameter k represents the actual number of clusters in the dataset. The above-mentioned values of k, d and n are valid only for synthetic datasets. These three parameters are used to generate synthetic datasets. For the real data sets, the values of these parameters are in table III.8.

An experiment is structured as follows:

- first of all, then all combinations of the values of the five parameters mentioned just above are defined;
- then, either the experiment si to be performed on a synthetic dataset, so this one is generated according to the above-mentioned parameters' values. If it is on a real dataset, so this one is loaded;
- finally, the estimation of k is performed by the three algorithms.

For each combination of these parameters, the experiment is conducted five times. We set the values of ϵ and λ in the different criteria as follows:

- ϵ has been defined as 2 and 10 respectively in Criteria 1 and 4. The value 2 is the least strict while 10 is the most strict;
- in the cluster regularization algorithm, $\epsilon = 8$ in the first part of the algorithm. If there are still small clusters left, then the second part is executed. In this case $\epsilon = 4$;
- in Criterion 3, $\lambda = 0.75$. As a result, ΔG must be less than 75% of ΔC.

We allow our algorithm and X-means to search up to $k_{max} = 5k$ centroids.

III.3.8.2.1 Synthetic data We generated synthetic data so as to have overlapping and Gaussian clusters. These synthetic data have the following properties:

- a cluster is a set of data that follows a normal distribution. For generating this set, a semi-positive covariance matrix and an average (a vector) are generated randomly. Indeed, the covariance matrix values allow to define a cluster shape that is not isotropic (strictly spherical), and that can have different variances separately on an arbitrary basis of directions and not necessarily on those of the dimensions;
- there are at least two overlapping clusters;
- if the centroid of one cluster is in the hyperectangle of another cluster, then we consider that the maximum degree of overlap has been exceeded. So as a result at least one of the clusters concerned is replaced by a new cluster;
- clusters do not have the same number of points. The cardinalities are chosen randomly so that their sum is equal to the total number of points n defined previously.

III.3.8.2.2 Real data The real data comes from three known sources: Openml, UCI, and Kaggle. They are all of a numerical type. These data are used in other research projects to perform benchmarking machine learning methods (van Rijn et al., 2014). The values of real k, n and d are given in Table III.8. The data sets used are diverse, and they can be organized into several groups:

- **black and white or grayscale images**; clustering is applied directly to the pixels in this case. *Emnist* and $Fashionmnist$ contain images respectfully on the first ten digits, clothing, and shoes. Each image was flattened to form a single vector. And each vector has been assigned its corresponding class (label);

dataset	k	d	n
vehicle	4	18	1000000
satimage	6	36	1000000
japaneseVowels	9	14	1000000
fourier	10	76	1000000
pendigits	10	16	1000000
fashionmnist	10	784	70000
ldpa	11	5	164860
walking	22	4	149332
letter	26	16	999999
zernike	47	47	1000000
emnist	62	784	697932

Table III.8: Real data sets characteristics.

- **4-band multispectral images**; they characterize different types of soil. The corresponding dataset is *satimage*;
- **sounds**; the $JapaneseVowels$ dataset is a set of digitized Japanese vowel sounds. Each sound is associated with a speaker;
- **historization of people's activities**; the *ldpa* dataset is a collection of the positions of sensors present on people to identify the movement they perform at a given time. The *walking* dataset is a set of people's activities designed to determine the authors;
- **images represented by a set of characteristics. These are related to the objects to be identified and are extracted from the images.**. For example, in the $Fourier$ dataset, each record is a set of coefficients of the character (between 0 and 9) shapes. Similarly, in $Zernike$, each instance is a rotation invariant Zernike moments of the digit image(between 0 and 9). In $Pendigits$ each individual is a sequence of positions characterizing a digit. In $Letter$, each instance of a letter of the English alphabet contains statistical moments and edge counts of the given letter. In $Vehicle$ dataset a data point is just a set of geometric characteristics of a given vehicle;

III.3.8.2.3 Metrics for evaluating experimental results For evaluating the results of the three algorithms, we used three metrics: first, the execution time (Δt) of the algorithm, the relative difference (Δk)(Bennett and Briggs, 2014) between the k actual value and the k estimated value, and finally, the distance between the actual partitioning and the partitioning proposed by the algorithm.

The relative difference is calculated as follows:

$$\Delta k = \frac{|k_{real} - k_{estimated}|}{k_{real}} \tag{III.21}$$

The relative difference compares two measurements relative to a reference measure-

ment (k_{real}). In our case, the relative difference is better than the absolute difference $|k_{real} - k_{estimated}|$ because the latter does not consider the measurement scale, e.g., the difference between 10 and 11 is not the same as between 1000 and 1001.

The more the relative difference Δk tends towards 0, the closer the estimated value k is to that of the ground truth.

The variation of the information (vi) (Meilă, 2003) was used as a distance between two partitionings. It measures the amount of information gained and lost for a dataset passing from a partition A to another partition B. It respects the three properties of triangular inequality. So the smaller the vi is, the closer the partitionings are to each other.

			Kd-means	G-means	X-means
k	d	n	Δt	Δt (× slower than Kd-means)	
10	4	1×10^6	39.7	344.3(8.7)	816.2(20.6)
		2×10^6	65.4	774.1(11.8)	1420.5(21.7)
		4×10^6	117.4	**1517.3(12.9)**	2203.6(18.8)
	6	1×10^6	41.9	396.4(9.5)	1101.5(26.3)
		2×10^6	72.9	926.9(12.7)	1869.9(25.7)
		4×10^6	133.7	1628.0(12.2)	3061.4(22.9)
	8	1×10^6	45.9	386.0(8.4)	789.4(17.2)
		2×10^6	81.2	823.1(10.1)	**1253.5(15.4)**
		4×10^6	153.6	1727.2(11.2)	2378.8(15.5)
32	4	1×10^6	77.6	535.6(6.9)	2817.2(36.3)
		2×10^6	132.5	1152.8(8.7)	4831.0(36.5)
		4×10^6	233.7	2468.4(10.6)	7628.1(32.6)
	6	1×10^6	79.2	586.7(7.4)	3689.9(46.6)
		2×10^6	146.9	1375.6(9.4)	7880.7(53.7)
		4×10^6	260.2	2485.1(9.5)	7901.8(30.4)
	8	1×10^6	82.2	665.7(8.1)	2491.6(30.3)
		2×10^6	154.9	1383.3(8.9)	3740.0(24.2)
		4×10^6	299.5	2883.7(9.6)	5037.7(16.8)
80	4	1×10^6	141.5	**781.9(5.5)**	7807.2(55.2)
		2×10^6	283.2	1673.6(5.9)	14147.0(50.0)
		4×10^6	570.8	3417.8(6.0)	21887.1(38.3)
	6	1×10^6	148.3	893.0(6.0)	10385.8(70.1)
		2×10^6	285.4	1892.4(6.6)	22403.6(78.5)
		4×10^6	578	4039.2(7.0)	37716.6(65.3)
	8	1×10^6	151.5	1016.0(6.7)	7834.6(51.7)
		2×10^6	308.2	**1708.6(5.5)**	21824.6(70.8)
		4×10^6	**626.4**	**3581.8(5.7)**	**47130.8(75.2)**

Table III.9: Comparison of the three algorithms executed on synthetic data. Note that Δt is expressed in seconds. In the Δt sub-column of the G-means and X-means columns, the number in brackets represents how many times our algorithm is faster than the compared algorithm.

III.3.8.3 Runtime analysis

If we consider the three algorithms' performance according to the metric Δt, Kd-means takes the least time to provide a clustering result. This is true in synthetic and real data and regardless of the value of k, d and n. Our algorithm is 5.5 to 12.9

	Kd-means	G-means	X-means
dataname	Δt	Δt (× slower than Kd-means)	
vehicle	38.6	**60530.8(1566.6)**	281.5(7.3)
satimage	54.5	34595.7(635.3)	1151.0(21.1)
japaneseVowels	75.6	55220.1(730.5)	534.8(7.1)
pendigits	31	40806.3(1316.9)	759.2(24.5)
fourier	91.6	38210.3(417.0)	2791.2(30.5)
fashionmnist	58.6	**954.1(16.3)**	697.0(11.9)
ldpa	12.5	5996.6(480.7)	132.3(10.6)
walking	14.7	3254.1(220.8)	**93.7(6.4)**
letter	57.8	22502.6(389.5)	**6272.4(108.6)**
zernike	118.1	35710.8(302.3)	5583.1(47.3)
emnist	**948.6**	46224.8(48.7)	**36029.7(38.0)**

Table III.10: Comparison of the three algorithms, on real data, with respect to execution time(Δt).

times faster than G-means and 15.4 to 75.2 times faster than X-means in synthetic data, respectively. The same trend occurs in real data, but from 16.3 to 1566.6 times compared to G-means and from 6.4 to 108.6 times compared to X-means. Maximum execution times were reached for all three algorithms at $d = 8$, $n =$ and $k = 80$ in the synthetic data. In this combination, the elapsed execution times are 10.43 minutes for Kd-means, 59.41 minutes for G-means, and 13h05 for X-means. In the real data, the maximums are reached in the configuration $d = 784$, $n = 697932$ and $k = 62$ for X-means and Kd-means. That is 15.48 minutes for Kd-means and 10 hours for X-means. In the case of G-means, the maximum is even higher comparing to the other algorithms, and it is reached at 16.48 hours ($k = 4$, $d = 18$, $n = 1 \times 10^6$).

III.3.8.4 Cluster quality analysis

Kd-means is better than other algorithms in estimating k with good clustering quality in synthetic and real data.

In the synthetic data, Kd-means does not exceed 0.47 in Δk and 1.2 in vi. X-means even reaches $\Delta k = 3.72$ and $vi = 3.4$. The same for G-means with $\Delta k = 8.1$ and $vi = 2.8$. The decrease in Δk G-means when k increased in the synthetic data is due to the presence of many Gaussian clusters that are well separated from each other. Kd-means has a better estimate of good quality than the others. In terms of vi, it is, on average, 2.46 and 3.05 better than G-means and X-means. Same performance in Δk, it is better on average by 2.83 (G-means) and 4.29 (X-means). It could be pointed out that G-means is better than X-means in terms of quality (vi) in several combinations of (k, d, n).

In real data, the Kd-means maxima do not exceed $\Delta k = 3.81$ and $vi = 7.2$. They are higher for X-means and G-means, they reach respectively ($\Delta k = 4.73$, $vi = 8.6$) and ($\Delta k = 555.5$, $vi = 12.5$). The values of Δk and vi are higher in real data than in synthetic data. Indeed, the distributions of clusters in real data are more complex than

			Kd-means		G-means		X-means	
k	d	n	k(Δk)	vi	k(Δk)	vi	k(Δk)	vi
10	4	1×10^6	8.4(0.17)	0.4	74.0(6.4)	2.2	45.7(3.57)	2.5
		2×10^6	8.3(0.21)	0.5	**91.0(8.1)**	2.4	45.7(3.57)	2.5
		4×10^6	8.3(0.18)	0.5	87.0(7.7)	2.3	44.6(3.46)	2.4
	6	1×10^6	8.6(0.15)	0.4	69.2(5.92)	2.2	46.0(3.6)	2.3
		2×10^6	8.1(0.2)	0.5	89.0(7.9)	2.5	46.1(3.61)	2.3
		4×10^6	8.0(0.23)	0.5	79.1(6.91)	2.2	45.9(3.59)	2.3
	8	1×10^6	7.4(0.39)	0.8	63.2(5.32)	2.6	45.9(3.59)	1.5
		2×10^6	6.9(0.37)	0.8	71.3(6.13)	**2.8**	45.5(3.55)	1.6
		4×10^6	6.0(0.41)	0.9	78.9(6.89)	**2.8**	46.2(3.62)	1.6
32	4	1×10^6	35.4(0.38)	0.9	99.3(2.1)	1.9	148.4(3.64)	2.9
		2×10^6	31.4(0.35)	0.8	114.1(2.57)	2.0	147.6(3.61)	2.9
		4×10^6	27.3(0.36)	1.0	132.1(3.13)	2.1	145.2(3.54)	2.9
	6	1×10^6	**40.2(0.47)**	0.9	101.0(2.16)	1.8	150.2(3.69)	2.7
		2×10^6	34.6(0.44)	0.9	122.5(2.83)	2.1	149.1(3.66)	2.7
		4×10^6	29.1(0.44)	1.0	107.9(2.37)	1.7	135.9(3.25)	2.7
	8	1×10^6	35.7(0.36)	1.0	115.2(2.6)	2.2	149.0(3.66)	1.9
		2×10^6	37.2(0.42)	1.0	126.8(2.96)	2.3	148.1(3.63)	1.8
		4×10^6	36.4(0.46)	0.9	136.5(3.27)	2.6	133.8(3.18)	1.8
80	4	1×10^6	90.5(0.24)	0.9	144.1(0.8)	1.5	368.2(3.6)	3.3
		2×10^6	88.3(0.29)	1.0	170.4(1.13)	1.8	373.1(3.66)	3.3
		4×10^6	83.0(0.3)	**1.2**	186.9(1.34)	2.0	366.6(3.58)	**3.4**
	6	1×10^6	96.2(0.37)	0.9	167.0(1.09)	1.6	376.1(3.7)	3.2
		2×10^6	97.4(0.39)	0.9	186.2(1.33)	1.8	375.0(3.69)	3.2
		4×10^6	92.8(0.4)	1.1	200.4(1.51)	1.7	363.3(3.54)	2.9
	8	1×10^6	57.5(0.28)	**1.2**	207.8(1.6)	2.0	377.2(3.71)	2.2
		2×10^6	98.6(0.43)	0.9	150.2(0.88)	1.4	373.6(3.67)	3.2
		4×10^6	**99.2(0.47)**	1.0	166.9(1.09)	1.5	**377.7(3.72)**	3.2

Table III.11: Comparison of the three algorithms, executed on synthetic data, with respect to the quality of clustering(vi) and the k estimation.

	Kd-means		G-means		X-means	
dataname	k(Δk)	vi	k(Δk)	vi	k(Δk)	vi
vehicle	3.0(0.28)	2.9	**2226(555.5)**	11.8	16.0(3.0)	5.3
satimage	**28.9(3.81)**	2.7	1177(195.17)	8.8	28.7(3.78)	3.8
japaneseVowels	9.4(0.83)	4.5	3377(374.22)	**12.5**	32.0(2.56)	7.7
pendigits	29.1(1.91)	3.6	2574(256.4)	9.1	32.0(2.2)	4
fourier	39.2(2.92)	3	1309(129.9)	8	**57.3(4.73)**	3.9
fashionmnist	10.9(0.11)	3.3	533(52.3)	7.1	32.0(2.2)	4
ldpa	32.5(1.96)	6.3	1766(159.55)	10.8	47.2(3.3)	7.1
walking	39.6(1.35)	6.7	1257(56.14)	10	100.7(3.58)	8.4
letter	59.7(1.3)	**7.2**	1581(59.81)	11.1	117.0(3.5)	**8.6**
zernike	86.9(1.08)	5.3	1157(23.62)	9	128.0(1.72)	6.4
emnist	45.6(0.26)	6	3073(48.56)	8.6	256.0(3.13)	6.7

Table III.12: Comparison of the three algorithms, on real data, with respect to the quality of clustering(vi) and the k estimation.

synthetic clusters (Gaussian). As a result, these complex representations of clusters are difficult to capture completely point by point. This increase is much higher for G-means because it tends to identify Gaussian clusters in particular. However, clusters in real data are not necessarily Gaussian because they are not constituted by Gaussian distribution generators. As a result, G-means makes an excessive overestimation of k

(therefore Δk high) compared to Kd-means. Our algorithm is the least affected by the gaussianity aspect of the clusters. Because, in its operation, gaussianity is not explicitly sought.

III.3.8.5 Clustering complexity analysis of real data

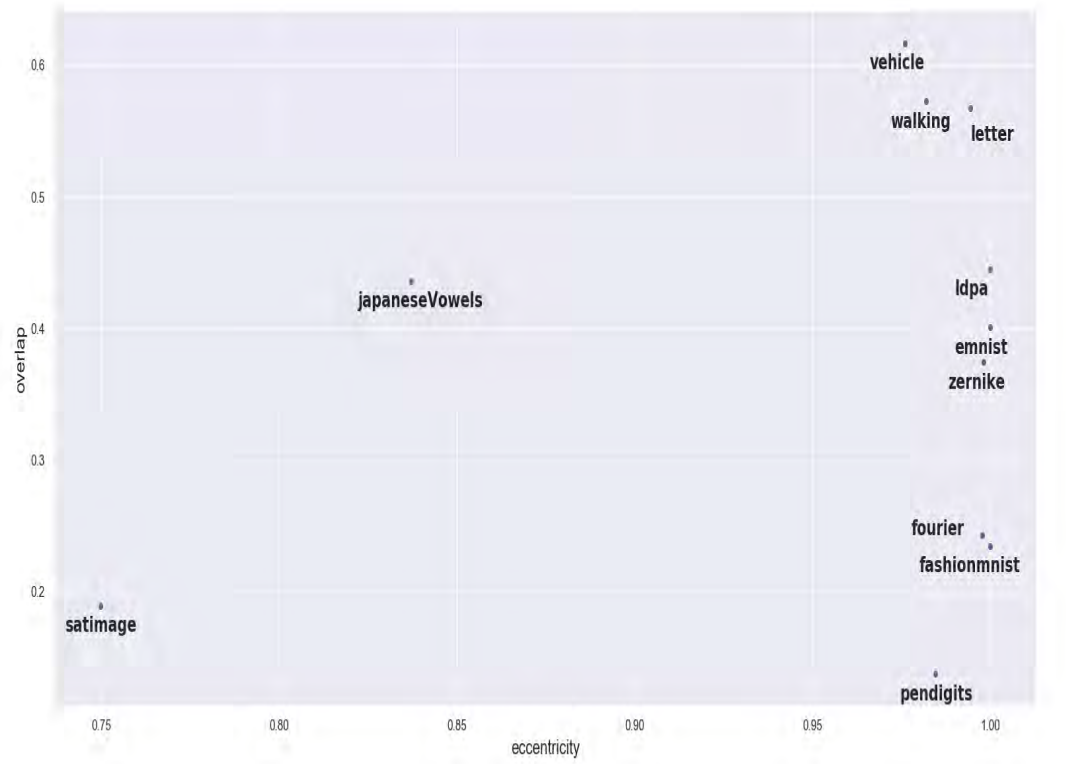


Figure III.17: Real datasets characterized by overlap and eccentricity measurements.

In this sub-section, we analyze the intrinsic complexity of the real datasets. For each dataset, we evaluate its difficulty to be clustered. For this, we used two metrics: eccentricity and overlap. These make it possible to characterize the problems to which Kd-means responds. Eccentricity is a positive real number that characterizes the shape of a set of points (Chekanov and Proudfoot, 2010). If it is equal to 0, the shape is strictly spherical, and when it approaches 1, it tends towards a very elongated elliptical shape. When it is equal to 1, it corresponds to another more complex shape. It is calculated in the following way for two-dimensional datasets:

$$ecc = \sqrt{1 - \frac{b^2}{a^2}} \tag{III.22}$$

Where b is the smaller half axis (i.e., half the length of a dimension) and a is the larger

half axis.

For datasets of more than two dimensions, the eccentricity is estimated as follows. First of all, all pairs of dimensions of a dataset are listed. Then the eccentricity is calculated for this dataset at the level of each pair. Then the average of all eccentricities is calculated. This is done for each class independently. Then the maximum of the eccentricities of the classes is calculated. Note that a class is a set of points belonging, in the ground truth, to the same group or phenomenon. The choice of the maximum is justified by the fact that if there is a class with a complex form, the algorithms for estimating k have more difficulties, even if other classes with simpler forms are in the same dataset. This difficulty is reflected in the clustering quality produced as well as in the overestimation of k with respect to reality.

The measure of overlap is a positive value between 0 and 1, which indicates at the global level of the dataset the level of overlap between classes of points. The value 0 indicates that there is no overlap, and 1 indicates that the classes are practically confused. It is calculated as follows. First of all, for each data point, we calculate its distance to p (set to 11) nearest neighbors, then we count the number of points belonging to the same class as the current point in its neighborhood. Intuitively, if there are many points not belonging to the same class as the current point in its neighborhood, it means that it is close to one or more other classes. So there is overlapping. A threshold of 25% is set at which overlapping is considered to exist. That is to say that at least 25% of points in the neighborhood of the current point must be different from it to consider the point as contributing to overlapping. Finally, a ratio is calculated by counting the points generating overlapping over the total number of points.

In figure III.17, we notice that all datasets have a strong ellipticity concerning the class shapes (eccentricity > 0.69 for all real datasets). Almost 72% of the datasets have an eccentricity higher than 0.95. This means that classes at this level have more difficult shapes than the elliptical one. Almost all of them correspond either directly to images or datasets related to images. The most difficult datasets could be those with high eccentricity and a high degree of overlap. Taking tables III.10 and III.12 into account, they correspond to walking, letter, ldpa, emnist, and zernike. In the case of Kd-means, it succeeds in estimating the number of clusters closest to real k while maintaining good quality and a short execution time compared to competitors. This is the case when it is used on the most complex datasets either with a high overlapping rate (e.g., vehicle overlapping rate > 0.6) or with elliptic classes or with a complex shape (e.g., zernike eccentricity > 0.9981 or fashionmnist eccentricity equal to 1.0).

	Synthetic data			Real data		
th_evo	Δk	vi	Δt	Δk	vi	Δt
0.05	0.3	0.8	213.2	1.7	4.9	132.5
0.10	0.3	0.8	203.1	1.6	4.8	144.5
0.15	0.3	0.8	198.2	1.5	4.7	134.0
0.20	0.3	0.8	193.5	1.4	4.7	141.1
0.25	0.3	0.9	188.6	1.2	4.5	136.5
0.30	0.4	1.0	183.8	1.3	4.6	130.5

Table III.13: Performance of our algorithm as a function of the value of *th_evo*

	Synthetic data			Real data		
ψ	Δk	vi	Δt	Δk	vi	Δt
4	0.5	1.1	195.0	1.3	4.2	131.1
8	0.3	0.8	201.5	1.2	4.4	141.3
10	0.3	0.7	200.0	1.4	4.8	136.1
12	0.3	0.8	190.5	1.9	5.3	137.5

Table III.14: Performance of our algorithm as a function of the value of ψ

III.3.8.6 Parameter sensitivity analysis

We analyze the three metrics (Δt, *vi* and Δk) according to *th_evo* and ψ. This is done in both data contexts (real and synthetic).

In Tables III.13 and III.14, *th_evo* and ψ do not have a particular influence on the execution time of Kd-means. Indeed, the range (the difference between the maximum and minimum values) of Δt does not exceed 10.2 seconds concerning ψ. It is about 30 seconds for *th_evo*. If we take into account the Δt magnitude of the three algorithms, these differences are negligible.

In the synthetic data, concerning ψ, the values of Δk are identical except for $\psi = 4$ where the value is slightly higher than the others by 0.2. The values of *vi* have a maximum difference of 0.4. This difference is 0.1 when $\psi \in [8, 10, 12]$ for which *vi* is minimal. At $\psi = 4$ Δk is slightly higher by 0.3 compared to the rest. In real data, the range is 1.1 for *vi* but at 0.2 if $\psi \in [4, 8]$. This range is 0.7 for Δk but drops to 0.2 for $\psi \in [4, 8, 10]$.

Concerning *th_evo* in the synthetic data, the values of Δk and *vi* have a range of 0.2. In real data, the respective ranges of Δk and *vi* are 0.5 and 0.4. They drop to 0.2 when $\psi \in [0.20, 0.25, 0.30]$ for Δk and $\psi \in [0.15, 0.20, 0.25, 0.30]$ for *vi*.

The observed ranges of the two metrics *vi* and Δk are lower than those of the same metrics of the competitor's algorithms in the real and synthetic data. However, if we are stricter by just accepting ranges less than or equal to 0.2, then the optimal values, when the data are Gaussian, are $\psi \in [8, 10, 12]$ and *th_evo* taking all the values tested. When the data is real (not necessarily Gaussian clusters), the optimal values are $\psi \in [4.8]$ and *th_evo* $\in [0.20, 0.25, 0.30]$.

III.3.8.7 Hyperspectral image use case

To see more concretely the Kd-means use, we propose to study a use case consisting in segmentation of a hyperspectral image named Pavia (figure III.18.a). Also, we compare the results of Kd-means and the Gaussian Mixture Model(GMM) (Titterington et al., 1985) method. Because the latter can detect spherical and elliptical clusters, nevertheless, it does not correspond to an estimator of k. It is then necessary to provide to GMM the k value. We provide as an input to GMM the exact number of clusters present in the ground truth. On the other hand, we let Kd-means estimate this value automatically.

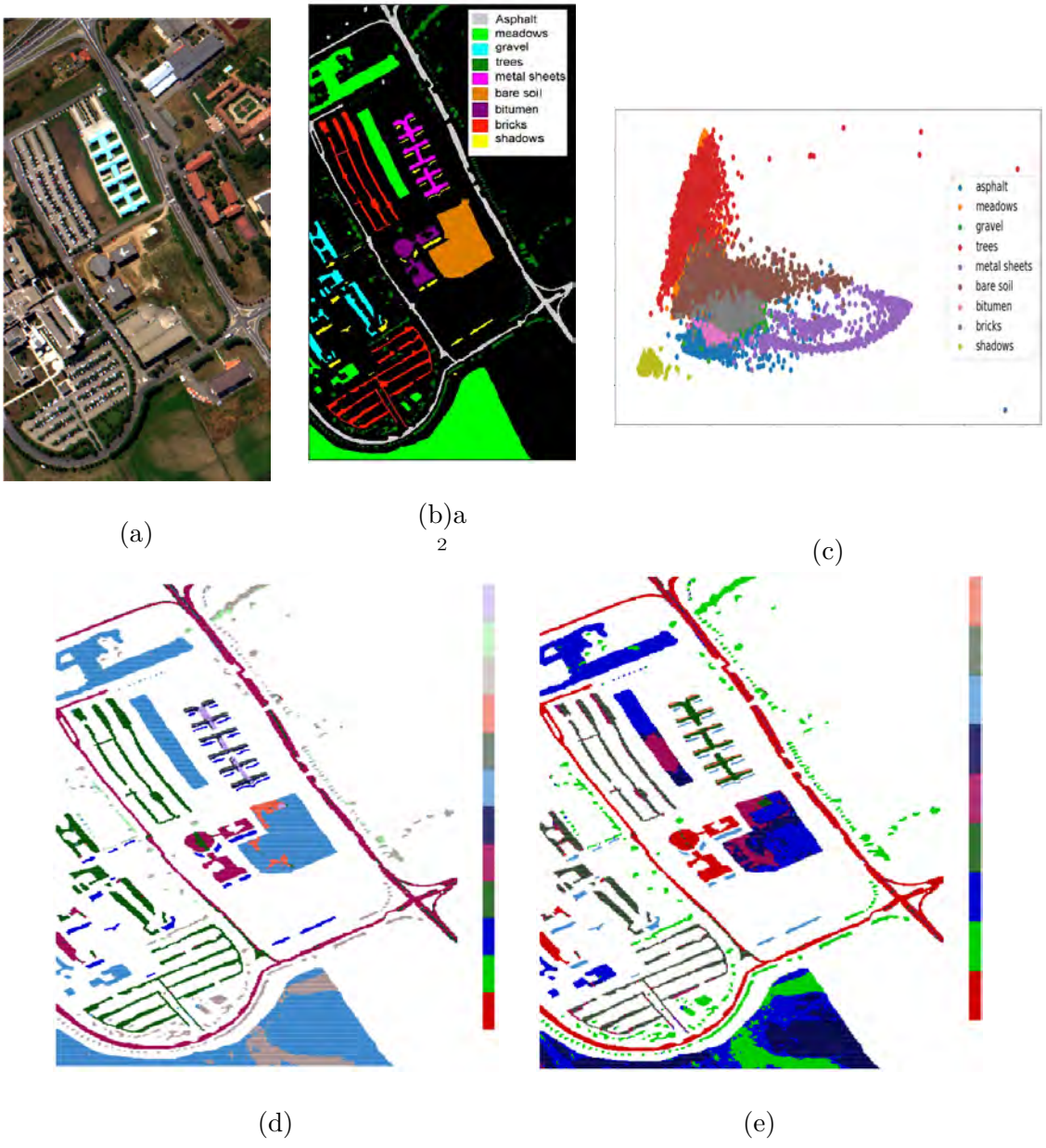


Figure III.18: a) Original RGB color image, b) Ground truth classes, c) Projection of the Pavia hyperspectral image on two of the most informative axes generated by PCA d) result produced by Kd-means, e) result produced by GMM.

class	asphalt	meadows	gravel	trees	metal sheets
overlapping rate	0.225	0.172	0.907	0.325	0.033

class	bare soil	bitumen	bricks	shadows
overlapping rate	0.729	0.489	0.423	0.001

Table III.15: Overlapping rate of each of the nine classes of Pavia. The rate is calculated as detailed in subsection III.3.8.5 but adapted to the class level instead of the entire dataset.

Pavia has a size of 610*610 pixels with 103 bands. The image is an aerial view of the university taken with the ROSIS sensor (figure III.18.a). It consists of 9 classes (figure III.18.b). The table III.15 and the figure III.18.c (projection of the image on two axes (or new dimensions) produced by PCA[3] to have a global idea on the spatial arrangement of the classes) inform about the clustering difficulty of Pavia. In the table, each class is associated with an overlapping rate. All classes are all overlapping. The least overlapping one is the *shadows* class, as we can see in the figure fig:pavia.c, it is separated from the others. Otherwise, the overlap of the other classes is clearly established in the figure. Moreover, the eccentricity is evaluated at 0.98, so there are classes with a shape at least elliptical if not more complex or irregular.

The task related to this image is to segment it into different parts, each belonging to a given class. Usually, clustering is used as part of the pre-processing step before using an algorithm specific to the type of hyperspectral image to produce a final result. In our case, we cluster directly on the raw image to maintain the clustering difficulty mentioned above. Therefore, the results produced by Kd-means and GMM are considered as intermediate results in a more global hyperspectral image processing framework.

The results produced by Kd-means and GMM are in figures III.18.d and III.18.e, respectively. To analyze these results, we rely on the figures III.18.a and III.18.b corresponding to the original image of Pavia [4]. The desired goal is to have an object whose color is as plain or homogeneous as possible because, in this way, the object's geometrical structure is identified. Based on this criterion, we can see that the objects' overall structures are relatively better preserved in the Kd-means results, which results in a better overall homogeneity of the objects compared to the GMM results. However, overall, the results of both methods are close and reasonable for a pre-processing step. This observation is reflected in more detail in figures III.19 and III.20. The first figure relates to each class its level of impurity. The fewer clusters that represent all class points, the less impurity the class has. To measure this, we use the Gini Impurity

[3]PCA(Principal component analysis) is a method that reduces a dataset defined in q-dimensional data space to another p-dimensional data space with $q > p$ while keeping as much relevant information as possible. In our case, this method allows us to visualize the approximate arrangement between the different classes of the dataset.

[4]A color present in different figures do not identify the same object(one instance of a class, e.g., meadows is a class, and an object is a meadow located in a zone of the image), i.e., a color of an object in one of the three figures b,d and e (figure III.18) could correspond to another color in the other two remaining figures.

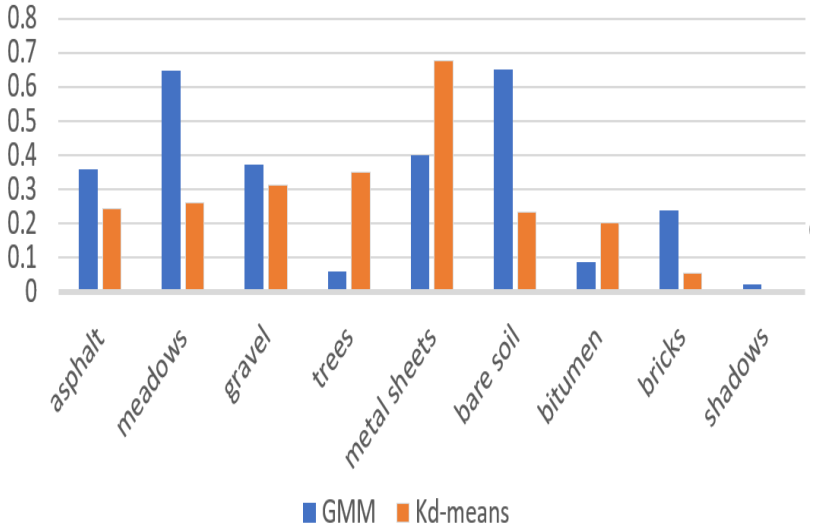


Figure III.19: Gini impurity index for each class. The index is framed between 0 and 1; the more it tends towards 0, the purer the class is.

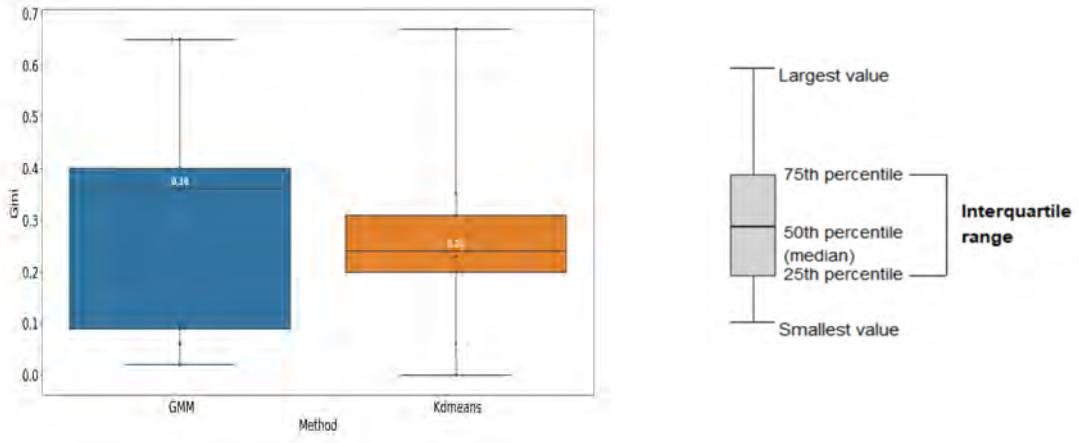


Figure III.20: On the right, the comparison of GMM and Kdmeans by boxplots. Solid circles are the Gini results. On the left, the boxplot legend.

Index. In six classes, Kd-means has less impurity than GMM. Among these six classes are the overlapping gravel class (overlapping rate = 0.907) and the less overlapping shadows class (overlapping rate = 0.001). A difference in impurity between Kd-means and GMM of nearly 6% is found to the advantage of Kd-means. In the second figure III.20, the results' distribution is presented through boxplots. In the latter, the Kd-means median (0.24) is lower than that of GMM (0.36), i.e., Kd-means, globally, produces purer classes. Moreover, the results' dispersion, measured by the interquartile range, is smaller in Kd-means than in GMM, which shows that Kd-means has greater stability in Gini purity than GMM.

In estimating the final clusters, Kd-means found 12, including two clusters with a cardinality of 1 and 4. Without taking into account these last two clusters, we could say that it produced 10 clusters.

To sum up, Kd-means produced a quality result globally better than GMM. Indeed, the values of vi of Kd-means and GMM are 1.29 and 1.63, respectively. This shows that Kd-means globally managed to capture more information from the field truth than GMM. Taking into account all the comparative results, Kd-means represents a good alternative to GMM.

III.3.9 Conclusion

In this part, the main objective is to estimate the actual number of clusters in a dataset and, at the same time, to build these clusters.

In the literature, several solutions have been proposed to meet this objective. The X-means and G-means methods are among the best known and most efficient. They are based on information theory and statistical tools. However, these methods deviate excessively in the estimation of k when the clusters are not strictly spherical or elliptical or when the clusters overlap. Consequently, the clustering quality is negatively impacted, i.e., the actual clusters have not been identified correctly. Moreover, X-means and G-means, iterative methods, use k-means on all points at each iteration. This makes them expensive for large datasets.

In response to these limitations, we have developed Kd-means, a solution based on the Kd-tree binary hierarchical data structure. It identifies clusters with spherical, elliptical, and even more complex shapes in large datasets. And this, even if the clusters overlap.

Kd-means consists of four consecutive steps:

- structuring and organization of data in Kd-tree. Kd-tree allows to pre-cluster the data, i.e., to put in the same node similar points. In this structure, we have integrated the hyper-rectangle geometric tool (a rectangle generalized to multidimensional data spaces) to frame a group of points while allowing to approximate the group's spatial distribution. Moreover, it allows quick calculations between two groups of points because there is no more need to visit these groups' points. This tool is used in the following steps;

- leaf initialization. once kd-tree is built, we find groups of points, called clusters in the leaves. So each leaf has its own clusters that are independent of those of the other leaves;

- Merging of clusters. Clusters in the leaves may not correspond to the real clusters, i.e., they correspond to the final clusters' subclusters and should be merged. For this reason, we have proposed a merge test process (MTP) to evaluate whether two clusters should be merged or not. Using this process, clusters are merged recursively in a hierarchical way from the leaves to the root;

- Regularization. At the end of the previous step, small clusters (in number of points) can be isolated from larger clusters. This step makes it possible to assign small clusters to the larger ones, agglomerate small clusters to form a single one or consider them separate clusters.

Experimentally, Kd-means has been compared to X-means and G-means on various data sets (synthetic and real). It has been shown that Kd-means could be better in runtime up to more than 1500 times. Moreover, in the estimation of real k and clustering quality Kd-means is much better than its competitors. It was also compared to GMM (Gaussian Mixture Models) [5] on a hyper-spectral image where GMM was given the real number of clusters k to identify, and Kd-means had to estimate this number automatically. Overall, in this application framework, the results were encouraging for Kd-means on the clustering quality and the estimation of k.

III.4 Conclusion

In this chapter, we have developed two solutions for different problems. The first, Sk-means, is a strategy that reduces the number of unnecessary calculations in k-means and its associated versions based on geometric reasoning. So its goal is to reduce execution time in a large dataset environment. The second solution, kd-means, automatically estimates the number of clusters while forming them. Kd-means also addresses its competitors' limitations, such as the strong constraint on cluster shape, cluster overlapping, and long execution time in a large dataset environment.

So far, we have contributed to the paradigm of partition-based methods (El Malki et al., 2019b)(El Malki et al., 2019a)(El Malki et al., 2020b). The next chapter focuses on the density-based methods paradigm (adopting a different cluster definition than the previous paradigm)(El Malki et al., 2020a). In the following, we do not focus on the execution time but the clustering quality and the data's high dimensionality.

[5] GMM is a probabilistic approach that produces both spherical and elliptical clusters. Nevertheless, it requires the number of clusters to be provided a priori.

Chapter IV

DECWA: Density-Based Clustering using Wasserstein Distance

Contents

IV.1 Introduction

In this chapter, we focus on the density-based methods paradigm. It takes a different clustering approach from the paradigm based on partitioning methods, and more specifically, on the cluster's definition. As a result, the paradigms, in practice, may have different applications. In this paradigm, a cluster is a dense region of points separated from other clusters by less dense regions (Kriegel et al., 2011). This definition has allowed these methods to detect arbitrary-shaped clusters. Nevertheless, density-based clustering has difficulties in detecting clusters having low densities regarding high-density clusters. Low-density clusters are either considered as outliers or included in another cluster. Moreover, near clusters of similar densities are often grouped in one cluster. Finally, density-based clustering does not manage well in high-dimensional data (Kriegel et al., 2009).

Our proposals consist of a new way to divide datasets into homogeneous groups in terms of density, a novel agglomerative method based on the Wasserstein metric, and a new clustering algorithm, called Decwa, based on spatial density and probabilistic approach.

First, we present the global approach of our solution (section IV.3), then we detail each of its steps. Finally, we experimentally evaluate the robustness of the solution on various data sets (section IV.9).

IV.2 Notations

In addition to table IV.1, we give more information about the cluster, sub-cluster, and distance probability density function.

Cluster. we consider a cluster as a contiguous region of points where density follows its own law of probability. Formally, we represent a cluster C_i by a set of pairwise distances between the points contained in the cluster C_i. This set is annotated D_i and follows any law of probability. D_i is computed via any distance defined as $X \times X \rightarrow \mathbb{R}^+$ that has to be at least symmetric and reflexive.

Sub-cluster. It corresponds to a subset of a cluster. The latter can be made up of several sub-clusters. In the solution intermediate steps, the sub-clusters are produced and used so that in the last step, a cluster set is returned.

Distance probability density function (distance $p.d.f$**)**. It characterizes the probability law of the distances between the points of a group (cluster or sub-cluster). In other words, it informs about the group spatial density as well as the source that generated the points.

Symbols	Meaning
X	$X = \{x_1, \ldots, x_n\}, n = \|X\|$ is a dataset
C	$C = \{C_1, \ldots, C_c\}$, $c = \|C\|$ is a set of clusters,
C_i, C_j	$C_i \in C$, $C_j \in C$ are clusters
D	$D = \{D_1, \ldots, D_{\|D\|}\}$ is a set of distance distributions
D_i, D_j	$D_i \in D$, $D_j \in D$ are distance distributions
D'	the set of distances
G	$G = (V, E)$ is an undirected weighted graph
V	$V = \{v_1, \ldots, v_{\|V\|}\}$ is a set of vertices
v_x, v_y	$v_x \in V$, $v_y \in V$ are vertices
E	$\{e_1, \ldots, e_{\|E\|}\}$ is a set of edges
e_q	$e_q \in E$ is an edge
(v_x, v_y, w_q)	$(v_x, v_y, w_q) = e_q$ is an edge, $(v_x, v_y) \in V^2, w_q \in \mathbb{N}$
w_q	w_q is the weight associated to the edge e_q
G^{min}	$G^{min} = (V^{min}, E^{min})$ is a Minimum Spanning Tree
sg_i, sg_j	$sg_i \subset G$, $sg_j \subset G$ are sub-graphs
$\mho$	$\{l_1, ..., l_t, ..., l_{\|\mho\|}\}$ is a set of mid-distances
$\aleph_t$, $\wp_t$	sets of connected sub-graphs at t^{th} iteration
$\|\|C_i - C_j\|\|$	a spatial distance between C_i and C_j
$ws(D_i, D_j)$	ws is the probabilistic distance between D_i and D_j
λ	λ is the spatial distance threshold
μ	μ is the probabilistic distance threshold

Table IV.1: Symbols

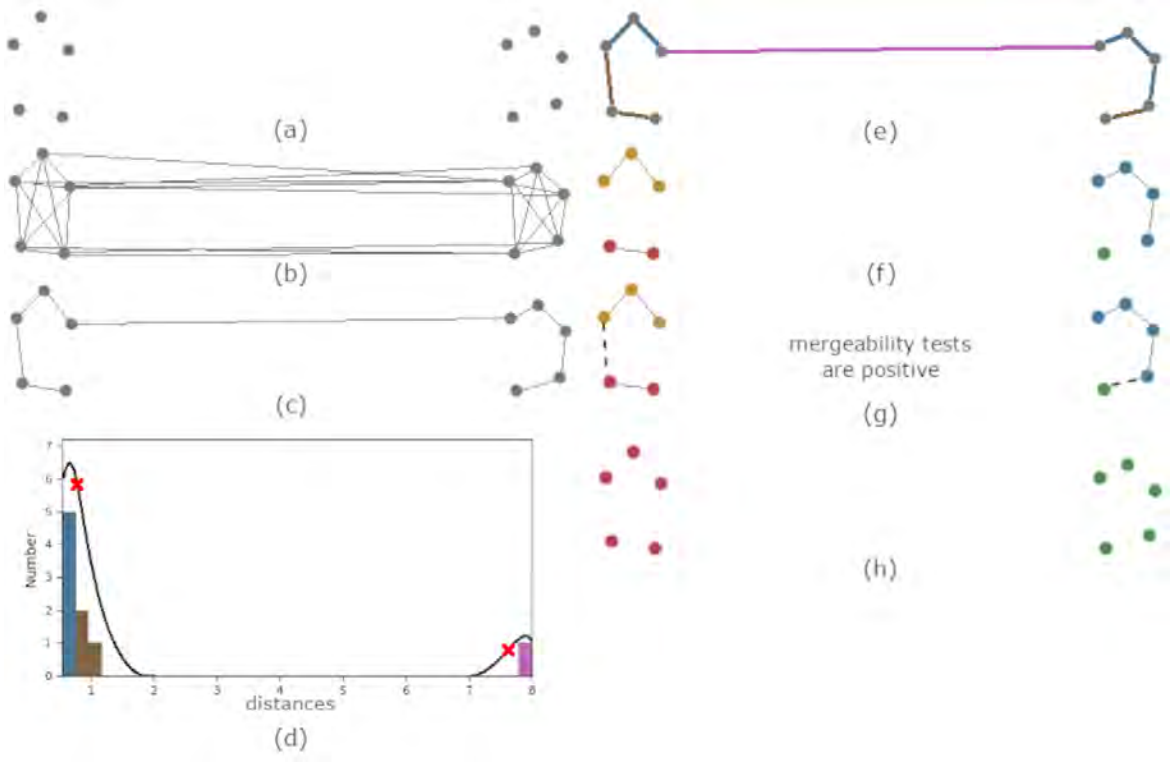
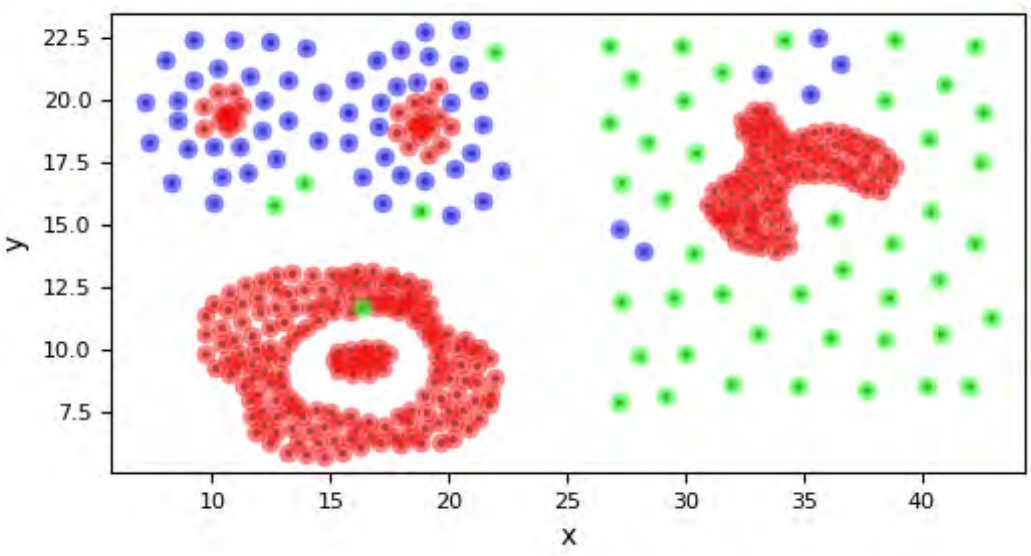
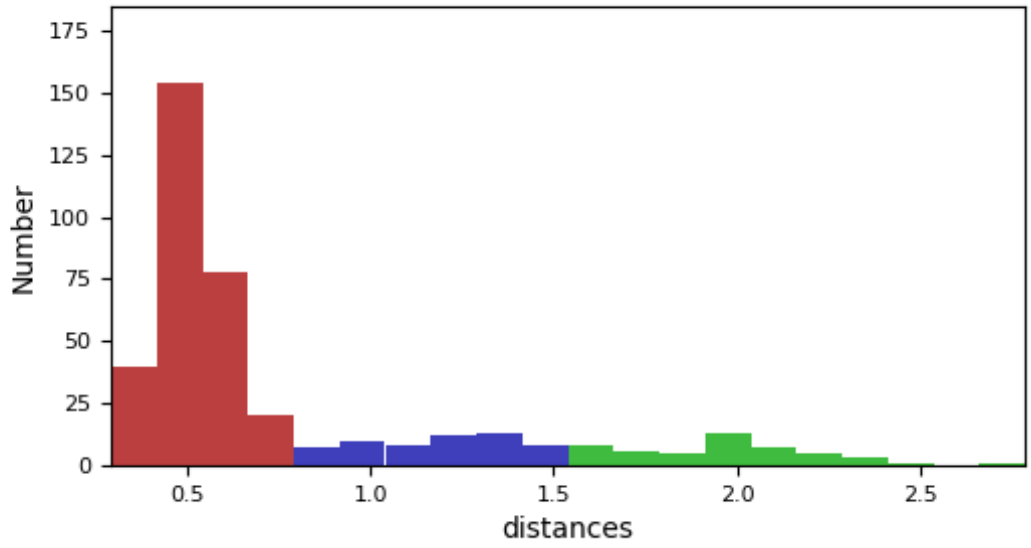


Figure IV.1: Overview



(a) Compound, a dataset with clusters of different densities



(b) Histogram representation of the distances in the MST for Compound dataset

Figure IV.2: Density detection

IV.3 DECWA overview

With the different steps described further, DECWA is able to detect small distance variations and thus to separate near clusters of same density by detecting the overlapping regions or the small interval between them. Moreover, it allows DECWA to solve the problem of low-density clusters by isolating its points. In addition, as DECWA uses distance $p.d.f$ to extract richer information from pairwise distances, it is more likely to benefit from the qualities of a distance suited for high dimensional data.

The proposed solution consists of four consecutive steps (In the figure IV.1, an example of the four steps of the solution through a dataset.):

1. graph-oriented modeling of the dataset. The first step transforms the dataset X (figure IV.1.a) into a k-nearest neighbor graph ($knng$) representation where nodes are points and edges are pairwise distances. This in order to perform clustering on the distance space (one-dimensional) instead of the multi-dimensional coordinate space. (IV.1.b,$k = 5$ i.e there are 5 edges per node in the example). The graph is then reduced to its Minimum Spanning Tree (MST) in order to keep significant distances (IV.1.c).

2. probability density estimation. The second step consists in calculating the $p.d.f$ from the significant distances of the MST. This $p.d.f$ is used to determine the extraction thresholds used to separate the different groups of spatial densities (IV.1.d,IV.1.e).

3. graph division. The third step consists in extracting sub-graphs from the MST according to extraction thresholds and identifying corresponding sub-clusters homogeneous in terms of spatial density (IV.1.f). To check whether sub-clusters should be merged because they might belong to the same cluster, the next step is performed.

4. agglomeration of sub-clusters. The fourth step is to agglomerate sub-clusters according to spatial and probabilistic distances (IV.1.g). Here, Wasserstein distance measures the similarity between probability distributions to regroup similar sub-clusters. Agglomeration is guided by mergeability tests.

IV.4 Graph-oriented modeling of the dataset

The first step consists in associating a new representation to the dataset X. From this representation in the following steps, the distance $p.d.f$ is calculated, and then sub-clusters are extracted. First, we discuss the representation and then how to generate it.

We have associated to X defined in a coordinate space, a distance space. The Decwa steps are carried out on inter-point distances instead of the original coordinates of the points. This allows to capture information on the different densities without referring directly to the points and performing probabilistic modeling on a uni-dimensional space instead of a multi-dimensional space. Indeed, a subset of distances represents several separated areas of similar densities in X. In figure IV.2, both representations are shown. Figure IV.2(a) is the coordinate space representation of Compound dataset as well as figure IV.2(b) is its corresponding representation using the frequency of pairwise distances where each color represents a level of density. In figure IV.2(a) the same density is shared by several separated groups of points in the coordinate space.

To generate the new distance space of X, we calculate distances between the points of the dataset. Our computation strategy of pairwise distances is based on the k-nearest neighbor method (Aggarwal and Reddy, 2013). The k-nearest neighbor graph highlights the local relationships between each point and its surrounding points in the data space. By extension, these relations make it possible to better capture the different spatial densities with local precision.

Therefore, the first step is the construction of the k-nearest neighbor graph of the dataset. From the dataset X, an undirected weighted graph annotated $G = (V, E)$ is constructed, where $V = \{v_1, v_2...\}$ is the set of vertices representing data points, and $E = \{e_q = (v_x, v_y, w_q) | v_x \in V, v_y \in V, w_q \in \mathbb{R}^+\}$ is the set of edges between data points. For each point, we determine k edges to k-nearest neighbor points. The weight w_q of each edge e_q is the distance between the two linked points v_x and v_y.

To gain efficiency and accuracy, it is possible to get rid of many unnecessary distances. We favor short distances, avoiding keeping too large distances in the k-nearest neighbors. To obtain dense (connected) sub-graphs, we generate a Minimum Spanning Tree (MST) from G. Constructing the MST of G consists in minimizing the number of edges while minimizing the total sum of the weights and keeping G connected. The interest of the MST in clustering is its capability to detect clusters of irregular boundaries (Zahn, 1971). As the MST, denoted G^{min}, is built thanks to the $knng$, therefore, k must be superior to a certain value so that the algorithm has enough edges to construct an optimal MST. Except for that matter, the value of k has no further influence as the MST will model the dataset.

IV.5 Probability density estimation

The overall objective of the $p.d.f$ estimation is to extract new information to delimit sub-clusters.

Classically, density-based methods explicitly or implicitly search for the densest

Algorithm 5 Elbow research

Input: $extrema$
Output: $elbow$
1: $elbow \leftarrow 0$
2: $reference \leftarrow 0$
3: **for** $h \in H$ **do**
4: {If there is an increase in the number of extremes and update authorization, the elbow value is modified}
5: **if** $extrema(h) > extrema(h-1)$ AND $elbow = reference$ **then**
6: $elbow = h$
7: **end if**
8: {If there is a decrease in the number of extremes, the update is authorized in line 5}
9: **if** $extrema(h) < extrema(h-1)$ **then**
10: $reference = elbow$
11: **end if**
12: **end for**

areas from a $p.d.f$ $f : X \rightarrow \mathbb{R}^+$. They correspond in f to the zones around the peaks (included) called level sets (Hartigan, 1975). Let $l > 0$ be a density threshold, a level set is a set of points defined as $\{x_i | f(x_i) > l\}$. This threshold l is on both sides of the bell (level set) centered around the peak. The sets of points $\{x_i | f(x) \leq l\}$ are either considered as outliers or assigned to denser clusters. This approach is not suitable for capturing clusters of different densities.

Another solution (Hartigan, 1975), based on the hierarchical approach, cuts f at different levels l to better capture clusters of different densities. In our case, f is defined as $f : D' \rightarrow \mathbb{R}^+$ with D' the set of distances. We consider in our case that a level set has two thresholds that are not necessarily equal. Areas of f below these thresholds are not considered as outliers. They are the less frequent distances that correspond to phenomenons with fewer points such as overlapping areas or small clusters.

For estimating the $p.d.f$, we use the Kernel Density Estimation (KDE) (Davis et al., 2011). Its interest lies in the fact that it makes no a priori hypothesis on the probability law of distances. The density function is defined as the sum of the kernel functions K on every distance. The commonly used kernels are Gaussian, uniform, triangular, etc. The KDE equation is below:

$$\widehat{f}_h(z_t) = \frac{1}{(n-1)h} \sum_{i=1}^{n-1} K\Big(\frac{z_t - z_i}{h}\Big) \tag{IV.1}$$

In our case, z_t and z_i correspond to the distances contained in G^{min}. We have $n = |X|$ (number of nodes in G^{min}), and $n-1$ is the number of distances.

The smoothing factor $h \in [0; +\infty]$ is called the bandwidth. It acts as a precision parameter, and in our case, it influences the number of peaks detected in the $p.d.f$ and therefore, the number of different densities. A commonly used method to estimate h

Algorithm 6 Graph division process

Input: $G^{min} = (V^{min}, E^{min}), \mho = \{..., l_t, l_{t+1}, ...\}$
Output: $\wp_1$

$\wp_{|\mho|} \leftarrow \emptyset$
$\aleph_{|\mho|} \leftarrow (V^{min}, E^{min})$
for $t \leftarrow |\mho| - 1$ **to** 1 **do**
  $E_t \leftarrow \{e_q \in E_{t+1} | w_q < l_t\}$
  $V_t \leftarrow \{v_q \in V_{t+1} | \exists (v_q, v_y, w_q) \in E_t \vee (v_x, v_q, w_q) \in E_t\}$
  $\aleph_t \leftarrow (V_t, E_t)$
  $\wp_t \leftarrow \wp_{t+1} \cup \{(V_{t+1} \setminus V_t, E_{t+1} \setminus E_t)\}$
end for
$\wp_1 \leftarrow \wp_1 \cup \aleph_1$

is Silverman's rule of thumb (Silverman, 1986). Most existing algorithms estimating h focus on having a bias-variance trade off and are not well suited for our algorithm. Intuitively, we want to have important phenomenons represented as peaks in the estimated $p.d.f$ and we want the best precision possible while avoiding noise to appear as peaks on the curve. Therefore, the solution we propose aims at finding when noise begins to generate peaks on the curve, it is based on the Elbow method (Satopaa et al., 2011). Indeed, the curve corresponding to the number of extrema associated with each value of h tends to have an Elbow shape. The proposed method finds *elbow*, the first point of the last peak in this curve, $extrema(h)$, with $h \in H$ and H the set of possible bandwidths. It is detailed in algorithm 5.

IV.6 Graph division

The purpose of the distance $p.d.f$ is to associate to each distance its density. The latter is a number that quantifies the intensity of the distance appearance frequency. The current step's goal is to extract the different density groups from the distance $p.d.f$ and the minimum spanning tree G^min. Each group represents a finite list of sub-clusters. The difficulty lies in how to extract these groups.

To separate different densities, the distance $p.d.f$ the decomposed into several parts via cuts. The significantly different densities are detected with the peaks of the distance $p.d.f$ curve. To separate highly represented distances to less represented distances, we consider an extraction threshold as the mid-distance between each maximum and its consecutive minimum, and conversely between each minimum and its consecutive maximum. Mid-distances allow capturing regions that are difficult to detect (low-density regions, or dense regions containing few points). Another interesting consequence concerns overlapping regions that have the same densities. These regions are often very difficult to capture properly, whilst our mid-distance approach makes it possible to detect these cases because overlapping between regions induces a density

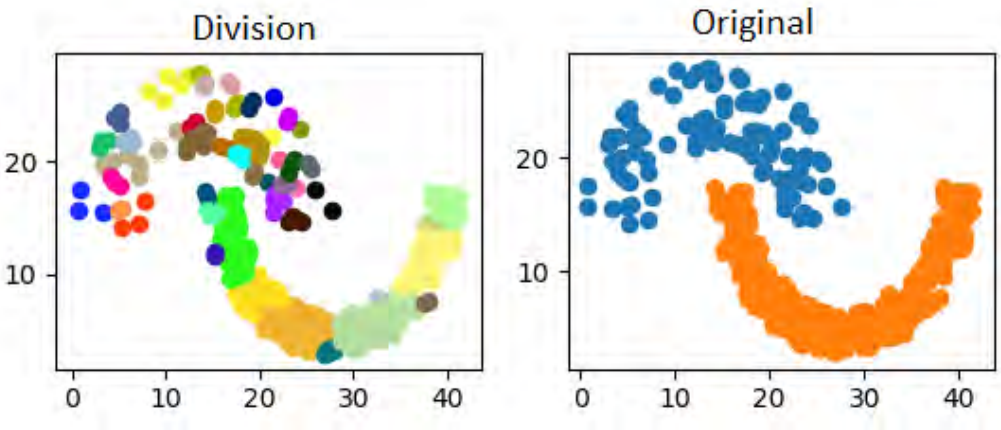


Figure IV.3: Comparison of the division result (on the left of the figure) with actual clusters for jain dataset (right)

variation. Once the mid-distances (ordered list called $\mho$) are identified on the $p.d.f.$ curve, we apply a process (algorithm 6) to treat successively the list of mid-distances in descending order. We generate sub-clusters, from the nodes of G^{min}, and sets of distances of sub-clusters, from the edges of G^{min}.

From G^{min}, via the process, we extract connected sub-graphs $\wp_t$ where all the nodes that are exclusively linked by edges greater or equal than the current mid-distance l_t. A node linked to an edge greater or equal to the l_t and another edge less than the l_t is not included. A sub-cluster C_i is composed of points that belong to one connected sub-graph. For each sub-cluster, we produce its associated set of distances D_i from edges between nodes of the sub-graph. A residual graph $\aleph_t$ is made up of edges whose distances are less than l_t, and all the nodes linked by these edges. This residual graph is used in the successive iterations. The residual graph can consist of several unconnected sub-graphs. At the last iteration (i.e. at the level of l_1), the residual graph $\aleph_1$ is also used to generate sub-clusters and associated sets of distances. After this step, some sub-graphs are close to each other while having a similar $p.d.f.$ According to our cluster characteristics, they should be merged. Figure IV.3 shows a comparison of the division result and the actual clusters for jain dataset. Figure IV.3 (a) is the result of the division. However, clusters are homogeneous it appears that many can be merged and still respect our cluster characteristics. Figure IV.3 (b) is the ground truth.

The connected sub-graphs of G^{min} have edge weights that are homogeneous overall. Indeed, the weight values of a connected sub-graph, annotated sg_i are framed between consecutive thresholds $l_t \in \mho, l_{t+1} \in \mho$, which guarantees its homogeneity in terms of spatial density.

Let us consider the list of mid-distances $\mho$, and a connected sub-graph sg_i with the weighted edges E_i. Suppose that $\mho$ is iterated consecutively in descending order as illustrated in algorithm 6. In a given iteration t, the current threshold is l_t. One

of the two following situations is performed: either $\forall e_q \in E_i, w_q < l_t$, in which case no action is taken on sg_i; or else if $\exists e_q \in E_i, w_q \geq l_t$ then from sg_i is extracted a set of connected sub-graphs called $\aleph_t$ with $w_q < l_t$ and another set of connected sub-graphs $\wp_t$ with $w_q \geq l_t$. At this point, for all edges into $\aleph_t$, $0 < w_q < l_t$. At the next iteration, $\aleph_t$ undergoes the same processing for the threshold l_{t-1}. If $\exists w_q \geq l_{t-1}$, the set of sub-graphs $\wp_{t-1}$ is extracted from $\aleph_t$. In this case, for all edges into $\wp_{t-1}$, $l_{t-1} \leq w_q < l_t$.

While homogeneous sub-clusters are constituted, some are close and have a similar density. They should be merged according to our definition of cluster. Therefore, the next step consists in identifying these sub-clusters to agglomerate them.

IV.7 Agglomeration of sub-clusters

This step aims at generating clusters from sub-cluster agglomeration. The main difficulty of this step is to determine which sub-clusters should be merged. When the distances between two sub-clusters are close, it is difficult to decide whether or not to merge them.

To arbitrate this decision, we use their density, represented as the distance $p.d.f$. Only sub-clusters that near spatially and have similar distance $p.d.f$ will be merged. As an example, at the top of the figure IV.4, two clusters Ci and Cj are indeed sub-clusters of the same cluster. Spatially, they are contiguous. In terms of probability law, at the bottom of the same figure, they are also close because their $p.d.f$ are similar, which in practice translates into similarity in spatial density.

To fulfill these objectives, we define the two following conditions. Given two sub-clusters C_i and C_j, the first condition is that these sub-clusters must be spatially close enough. To ensure this, $||C_i - C_j|| \leq \lambda$ must be respected, with $\lambda \in \mathbb{R}^+$. It is necessary to determine the distance between sub-clusters ($||C_i - C_j||$) to verify this condition. However, calculating every pairwise distance between two sub-clusters is a time-consuming operation. Therefore, we propose another solution based on G^{min}. We consider that the value of the edge linking sub-graphs corresponding to C_i and C_j as the distance between C_i and C_j. Because of the MST structure, we assume that this distance is nearly the minimum distance between the points of C_i and C_j.

The second condition relates to the similarity of the distance distributions D_i and D_j. The purpose of this condition is to ensure $ws(D_i, D_j) \leq \mu$, with $\mu \in \mathbb{R}^+$, and ws the Wasserstein distance. We have opted for the Wasserstein distance as a measure of similarity between probability distributions. Two advantages of this distance led us to its choice (Villani, 2009):

- it seamlessly takes into account the geometric shape of the $p.d.f$. curves;

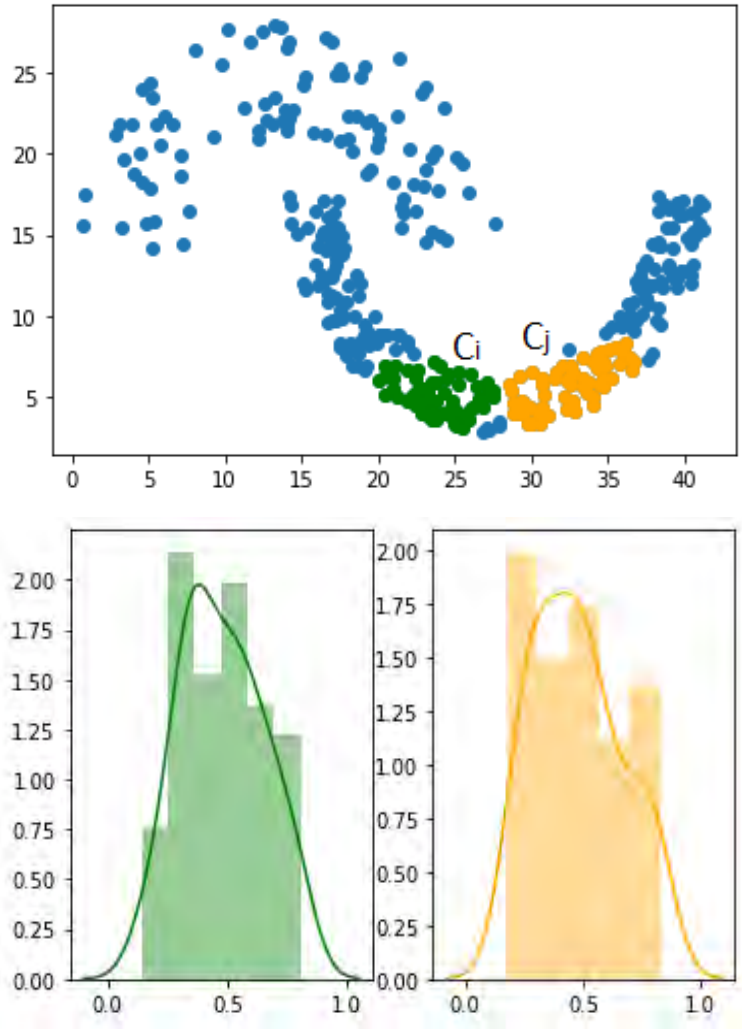


Figure IV.4: In the top sub-figure, two clusters C_i and C_j are shown. In the lower sub-figure, on the left the distance $p.d.f$ of C_i and on the right the distance $p.d.f$ of C_j

- if the two distributions are very close, then the small difference will be captured by Wasserstein, allowing a fine precision.

We introduce now an iterative process (algorithm 7) for merging sub-clusters. First of all, it identifies the edge set $E_g \subset E^{min}$ containing only nodes whose respective sub-graphs are different. Then it runs an iterative sub-process consisting in iterating on the edges of E_g. For each edge $e_q \in E_g$ we verify that $||C_i - C_j|| \leq \lambda$ and $ws(D_i, D_j) \leq \mu$. In this case, C_i and C_j are merged. This is operated by the union of D_i, D_j and w_q and considering the points of C_i and C_j as belonging to the same sub-graph. Note that if $min(|D_i|, |D_j|) \leq 1$ then only the first condition applies as ws can't be calculated with this cardinality. The iterative sub-process is repeated until there are no longer merges. The last iteration does not result in any merging and the points of the clusters whose cardinality is 1 are considered as outliers. As an example, the result obtained on the jain dataset corresponds exactly to the real clusters as on the right side of the

figure IV.3.

IV.8 Parameter estimation

In the merging phase, λ and μ are metric thresholds for d and ws. Therefore they are defined in $]0, +\infty[$. It is difficult for users to operate with this unbounded interval. In this sub-section, the objective is to map this interval to a bounded interval defined between 0 and 1. We argue that a bounded interval is more meaningful for users.

We consider $m \in \mathbb{R}^+, M \in \mathbb{R}^+$ respectively the lower bound and the upper bound of D' the set of distances and $\gamma, \tau \in]0, 1]$, so λ, μ are simplified as follows:

$$\lambda = \gamma * (M - m) + m \tag{IV.2}$$

$$\mu = \tau * (M - m) + m \tag{IV.3}$$

So when merging sub-clusters, the spatial distance between sub-clusters can never be greater than M and not smaller than m, hence the equation IV.2. The γ factor is a hyperparameter that indicates the level of severity on the merge. If it is equal to 1, the less strict value, it tends to allow all sub-clusters to be merged and inversely if it is near to 0. Similarly, the factor τ (equation IV.3) is also a hyperparameter and has the same effect on the probabilistic distance. Equation IV.3 is justified by the fact that the Wasserstein distance is bounded between 0 and $(M - m)$. The lower bound is 0 because Wasserstein is a metric. The following evidence proves that the upper bound is $(M - m)$.

Proof. Given $U : \Omega \to D'$ and $V : \Omega \to D'$ two random variables with $D' \subset \mathbb{R}^+$ and Ω the set of possible outcomes. D' is the set of distances in our case, U and V represents two sub-clusters and their inner distances. According to (Ramdas et al., 2015) the first Wasserstein distance between the distributions of U and V is:

$$ws(u, v) = \int_{-\infty}^{+\infty} |F_U(u) - F_V(v)| dudv \tag{IV.4}$$

F_U and F_V are respectively the Cumulative Distribution Function (CDF) of U and V. Let m be the lower bound of D' and M the upper bound of D', so $F_U(u) = F_V(v) = 0$ for $u, v \in I =]-\infty; m[$ and $F_U(u) = F_V(v) = 1$ for $u, v \in I' =]M; +\infty[$ according to CDF definition and because D' is bounded between m and M.

Then,

$$ws(u, v) = \int_I |F_U(u) - F_V(v)| dudv + \int_m^M |F_U(u) - F_V(v)| dudv + \int_{I'} |F_U(u) - F_V(v)| dudv$$

$$ws(u,v) = \int_I |F_U(u) - F_V(v)|dudv + \int_m^M |F_U(u) - F_V(v)|dudv + \int_{I'} |F_U(u) - F_V(v)|dudv$$

$$\iff ws(u,v) = \int_I |0-0|dudv + \int_m^M |F_U(u) - F_V(v)|dudv + \int_{I'} |1-1|dudv$$

$$\iff ws(u,v) = \int_m^M |F_U(u) - F_V(v)|dudv$$

By definition, $F_U : D' \to [0;1]$ and $F_V : D' \to [0;1]$, therefore $\int_m^M F_U(u)du \leq (M-m) \times 1$ and $\int_m^M F_V(v)dv \leq (M-m) \times 1$.

As F_U and F_V are positive functions :

$$ws(u,v) = \int_m^M |F_U(u) - F_V(v)|dudv \leq \int_m^M F_U(u)du \leq M - m$$

Figure IV.5 illustrates this proposition with an example. The two curves are F_U and F_V. Grey area between them corresponds to the area actually measured by Wasserstein and the rectangle area is the upper bound for $ws(u,v)$.

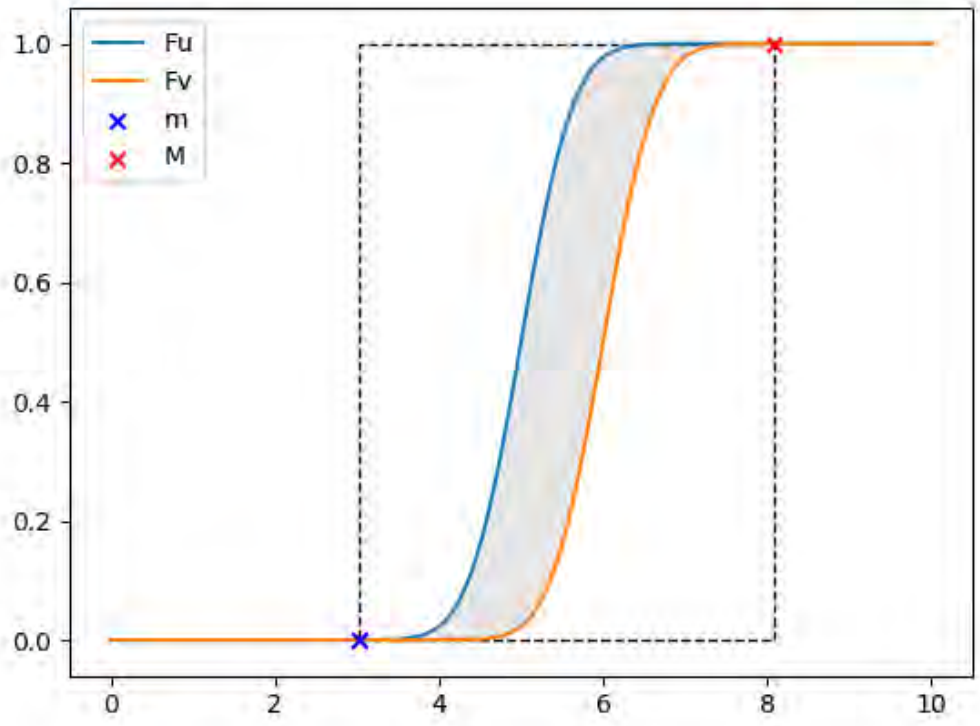


Figure IV.5: Maximum possible area between two CDF

Algorithm 7 Sub-cluster merger

Input: $D, \wp_1 = \{..., sg_i, ..., sg_j, ...\}, \lambda \geq 0, \mu \geq 0$
Output: $\wp_1$

$E_g \leftarrow \emptyset$
for all $e_q \in E^{min} | e_q = (v_x, v_y, w_q)$ **do**
  if $v_x \in sg_i \wedge v_y \in sg_{j \neq i}$ **then**
    $E_g \leftarrow E_g \cup \{e_q\}$
  end if
end for
repeat
  $\epsilon \leftarrow 0$
  for all $e_q \in E_g$ **do**
    if $w_q < \lambda$ **then**
      // D_i (respectively D_j) is associated to sg_i (sg_j)
      if $D_i \leq 1 \wedge D_j \leq 1$ **then**
        //the subgraph i and/or j as small, so it is considered close to the other w.r.t. wasserstein distance.
        $\delta \leftarrow -1$
      else
        // wasserstein distance ws between D_i and D_j is computed
        $\delta \leftarrow ws(D_i, D_j)$
      end if
      if $\delta < \mu$ **then**
        // agglomeration of sg_i and sg_j
        $D \leftarrow (D \setminus D_i) \setminus D_j$
        $D \leftarrow D \cup \{D_i \cup D_j \cup \{w_q\}\}$
        $\wp_1 \leftarrow (\wp_1 \setminus sg_i) \setminus sg_j$
        $\wp_1 \leftarrow \wp_1 \cup \{sg_i \cup sg_j \cup \{e_q\}\}$
        $\epsilon \leftarrow \epsilon + 1$
      end if
    end if
  end for
until $\epsilon = 0$

IV.9 Experimental assessments

IV.9.1 Introduction

We conducted an experimental study to show the effectiveness and the robustness of DECWA compared to state-of-the-art density-based methods. It was applied to a variety of synthetic and real datasets, using different distances depending on the data. First, an analysis comparing DECWA to other concurrent algorithms is performed. We showed that DECWA outperforms well known state-of-the-art density-based algorithms in clustering quality.

An other study followed concerns the influence of two internal parameters of DECWA: the bandwidth (h) and the k-nearest neighbors (k). In this study, we compare the different strategies for estimating these two parameters while maintaining a good clustering quality.

IV.9.2 Experiment protocol

IV.9.2.1 Algorithms

DECWA was compared to DBCLASD (Xu et al., 1998), DENCLUE (Hinneburg and Gabriel, 2007) and HDBSCAN (Campello et al., 2013). For the experiments conducted in this chapter, DECWA used the Gaussian kernel in the division phase. Algorithm parameter values are defined through the repetitive application of the random search approach (1000 iterations). This, in order to obtain the best score that the four algorithms can have. DBCLASD (parameter-free and incremental) was subject to the same approach but by randomizing the order of the data points because it is sensitive to it. The hierarchical structure proposed by HDBSCAN is exploited by the Excess Of Mass (EOM) method to extract clusters, as used by the authors in (Campello et al., 2013).

Clustering quality is measured by the commonly used metric named Adjusted Rand Index (ARI) (Hubert and Arabie, 1985). In addition, we also report the ratio of outliers produced by each algorithm for each dataset.

IV.9.2.2 DECWA environement

Several algorithms (e.g. Kruskal (Kruskal, 1956), Boruvka (Nešetřil et al., 2001), Prim (Prim, 1957)) exist to generate an MST. DECWA uses Kruskal algorithm to build the MST, it has a complexity of $O(|E|log(|E|))$ with $|E|$ the cardinal of edges. Kruskal is faster than Boruvka and Prim in sparse graphs (few edges) while Prim and Boruvka take less time than $Kruskal$ in dense graphs (Kershenbaum and Van Slyke, 1972). As G is a k-nearest neighbor graph, it is often sparse because of the small value of k which is studied in sub-section IV.9.5.

During the merging process, DECWA runs through G^{min} and therefore evaluates edges in an order which has a small influence over the result. Multiple sorting methods were tested and DECWA is not very sensitive to the order (ARI difference on average is less than 4% in the worst case). We obtained the best results by sorting edges by weight in ascending order, therefore, we suggest this sorting method for DECWA.

To cut $p.d.f$, different methods are tested and evaluated. Inspired by Hartigan's level set idea, the minima in the $p.d.f$ curve are tested to separate distance groups (44% ARI on average). The G-means algorithm, based on Gaussian Mixture Models, was tested to separate distance groups by considering them as sub-populations of a Gaussian mixture distribution (57% ARI on average). In the end, our mid-distance solution has better performances (67% ARI on average).

Dataset	**Dimensions**	**Size**	**LD**	**SD**
jain	2	373		
compound	2	399	x	x
pathbased	2	300		x
cluto-t7.10k	2	10000		
iris	4	150		x
cardiotocography	35	2126		
plant	65	1600	x	x
GCM	16064	190	x	x
new3	26833	1519		x
kidney_uterus	10936	384		x

Table IV.2: Characteristics of data sets.

IV.9.2.3 Synthetic datasets

The two-dimensional, jain, compound, pathbased (Fränti and Sieranoja, 2018), and cluto-t7.10k (Karypis et al., 1999) datasets are synthetic. They contain clusters of different shapes and spatial densities(i.e. clusters are not necessarily generated by the same law of probability). For these data, the Euclidean distance was used.

IV.9.2.4 Real datasets

The cardiotocography dataset (Ayres-de Campos et al., 2000) is a set of 2126 fetal cardiotocogram records, with 35 attributes providing vital information on the state of the fetus. The records are grouped into 10 classes, therefore 10 clusters are expected. Canberra distance was applied to it. Plant dataset (Mallah et al., 2013) is considered as a classification problem. It consists of 1600 leaves of 100 different classes. Each leaf is characterized by 64 shape measurements retrieved from its binary image. The Euclidean distance was used on Plant. Another known dataset, Iris (Dua and Graff, 2017) (150 iris flowers, 4 dimensions, 3 classes) was used with the Manhattan distance. A collection of two very large biological datasets were tested. The first dataset, GCM (Ramaswamy et al., 2001), is used to diagnose the type of cancer (14 classes). It

	DECWA		**DBCLASD**		**DENCLUE**		**HDBSCAN**	
Dataset	**ARI**	outliers(%)	**ARI**	outliers(%)	**ARI**	outliers(%)	**ARI**	outliers(%)
jain	**0.94**	**0.00**	0.90	0.03	0.45	0.06	**0.94**	0.01
compound	**0.95**	0.05	0.77	0.06	0.82	0.05	0.83	**0.04**
pathbased	**0.76**	**0.00**	0.47	0.03	0.56	**0.00**	0.42	0.02
cluto-t7.10k	**0.95**	0.03	0.79	0.06	0.34	**0.00**	**0.95**	0.10
iris	**0.77**	0.04	0.62	0.07	0.74	**0.00**	0.57	**0.00**
cardiotocography	**0.47**	0.04	0.24	0.17	**0.47**	**0.02**	0.08	0.28
plant	**0.18**	**0.03**	0.04	0.07	0.14	0.17	0.04	0.38
GCM	**0.46**	**0.05**	0.18	0.18	0.24	0.06	0.27	0.27
new3	**0.41**	**0.01**	0.09	0.45	0.08	0.13	0.13	0.27
kidney_uterus	**0.80**	**0.00**	0.53	0.08	0.56	0.09	0.53	0.26
Average	**0.67**	**0.02**	**0.46**	**0.12**	**0.44**	**0.06**	**0.48**	**0.16**

Table IV.3: Experimental results

consists of 190 tumor samples. Each one represented by the expression levels of 16063 genes. The second dataset (kidney_uterus) is proposed by expO (EXpression Project For Oncology) (Stiglic and Kokol, 2010). It consists of 384 tumor samples to be classified into two classes (kidney or uterus). Each sample is the expression level of 10937 genes. Given that the attributes all correspond to the same nature (gene expression) then the Bray-Curtis distance was applied to it. New3 (Han and Karypis, 2000), a very high-dimensional dataset, is a set of 1519 documents grouped into 6 classes. Each document is represented by the term-frequency (TF) vector of size 26833. Each element of this vector is the frequency of the i_{th} term in the document. The cosine distance was used to measure the similarity between documents.

IV.9.3 Results and discussion

The results are reported in the table IV.3 (best scores are in bold). LD column stands for low-density, it means that the datasets have at least one low-density cluster comparing to others. SD stands for near clusters of similar densities, it means that the marked datasets have at least two overlapping clusters with similar densities.

In many cases, DECWA outperforms competing algorithms by a large margin on average 23% (e.g. *jain* dataset, ARI margin is $0.94 - 0.45 = 0.49$). DECWA has the best results in datasets with low-density clusters (e.g. *compound* and *GCM*). In this case, there is an ARI margin of 20% on average in favor of DECWA. Though datasets with very high dimensions are problematic for the other algorithms, DECWA is able to give good results. Indeed, there is an ARI margin of 29% on average in favor of DECWA for the last three datasets. Near clusters of similar densities are also correctly detected by DECWA. Some datasets like *iris, kidney_uterus* and *pathbased* have overlapping clusters and yet DECWA separates them correctly. There is an ARI margin of 23% in favor of DECWA for datasets having this problematic.

The outlier ratio is not relevant in case of a bad ARI score because in this case, although the points are placed in clusters, clustering is meaningless. DECWA is the

one that returns the least outliers on average while having a better ARI score.

We conducted a statistical study, as recommended in (Demsar, 2006), to confirm the significant difference in performance between the algorithms and the robustness of DECWA comparing to the others. The overall concept of the study has two steps. 1) First, a statistical test (Iman-Davenport(Sheskin, 2007)) is performed to determine if there is a significant difference between the algorithms at the ARI level. 2) If so, a pairwise comparison of algorithms is performed, via a post-hoc test (Shaffer (Shaffer, 1986)), to identify the gain of one method over the others. Both tests return a p-value (in the case of the second test, it is returned for each comparison). The p-value allows us to decide on the rejection of the hypothesis of equivalence between algorithms. To reject the hypothesis, the p-value must be lower than a significance threshold α that we set at 0.01. The p-value by returned the first test is $5.38e^{-5}$. It is significantly small compared to α. This means that the algorithms are different in terms of ARI performance. Second, these differences are analyzed by the Shaffer Post-hoc test. It returns a p-value for each test on a pair of algorithms. Indeed, for the case of DECWA, the p-value is much lower than α when comparing DECWA to others ($7.5e^{-4}$ with DBCLASD, $4.6e^{-3}$ with DENCLUE and $3.0e^{-3}$ with HDBSCAN), which proves that DECWA is significantly different from the others. For the others, the p-value is equal to 1.0 in all the tests concerning them, which statistically shows that they are equivalent. The ranking of the algorithms according to the ARI was done by Friedman's aligned rank (Garía and Herrera, 2008). DECWA is ranked as the best. All in all, DECWA is significantly different from the others and is the best performing.

IV.9.4 h estimation

This experiment aims to find the best method to determine the value of h using Elbow approach. For these tests, we use Yellowbrick's implementation (Satopaa et al., 2011) of Elbow Method, Silverman's rule of thumb (Silverman, 1986), and the proposed method. Yellowbrick is able to give one point or to detect several points in the elbow, therefore we test two other methods by selecting a point among them. The first method is to take the median point among the set of proposed points. The intuition behind this method is that doing so h is less likely to induce noise than the last proposed point while being more precise than the first point. The second method is based on the observation that actual phenomenons, resulting in clusters, are likely to induce stable peaks on the $p.d.f$ curve. Therefore this method finds the point with the biggest plateau on the $extrema(h)$ curve. Fig. IV.6 shows the Elbow shaped curve obtained for jain, a synthetic dataset, and the points found by every solution. On this example, our method found $h = 0.17$ which induce 3 extrema in the estimated $p.d.f$. Table IV.4 shows the evaluation of each method on the presented datasets. This experiment is

	pm		yellowbrick		biggest plateau		median		silverman	
dataset	**ARI**	**ouliers**	**ARI**	**ouliers**	**ARI**	**ouliers**	**ARI**	**ouliers**	**ARI**	**ouliers**
jain	0.94	**0.00**	0.94	**0.00**	0.94	**0.00**	0.94	**0.00**	**1.00**	**0.00**
compound	**0.91**	0.03	0.91	**0.02**	0.89	0.10	0.89	0.10	**0.91**	**0.02**
pathbased	0.61	0.02	0.44	**0.00**	0.68	**0.00**	0.61	0.02	**0.68**	**0.00**
cluto-t7.10k	0.91	0.02	0.55	**0.00**	0.49	0.01	0.49	0.01	**0.93**	0.03
iris	**0.70**	0.09	0.60	**0.01**	0.70	0.09	0.72	0.09	0.58	0.02
cardiotocography	0.38	0.05	0.38	**0.03**	0.36	0.06	0.36	0.06	**0.47**	0.04
plant	0.18	0.03	0.11	**0.00**	0.12	0.01	0.18	0.02	**0.20**	0.03
GCM	0.43	0.09	**0.44**	**0.04**	0.43	0.18	0.43	0.18	**0.44**	**0.04**
new3	0.40	0.04	**0.42**	0.03	0.40	0.04	0.40	**0.01**	0.40	0.05
kidney_uterus	0.77	**0.00**	**0.80**	0.02	0.65	0.03	0.35	0.04	0.51	0.00
Average	**0.62**	0.04	0.56	**0.01**	0.57	0.05	0.54	0.06	0.61	0.02

Table IV.4: Comparison of methods for estimating h. Pm refers to our proposed method.

conducted using $k = log(|V|)$. Our proposed method achieves good scores and is even able to catch up Silverman's rule of thumb on average.

IV.9.5 k estimation

As explained in this paper (Gower and Ross, 1969), the MST can be used for agglomerative clustering, which is the case for DECWA. However, it is also pointed out that the exact distances may be lost during the process of creating the MST. A connected graph is recommended before applying an MST generation algorithm because it contains inter-sub-cluster relationships. These relations are useful to better ensure the DECWA merging part. The value of k must be large enough for the graph to be connected. The authors of (Brito et al., 1997) stipulate that for $knng$ to be connected k is chosen to the order $log(|V|)$.

In Table. IV.5, we compared the distances between sub-clusters retained by the MST and the distance between the two closest points each belonging to a different sub-cluster. On previously used datasets, for $k = 3$ a notable percentage of edges are different (2.3%), however it decreases as k gets bigger : 0.2% for $k = log(|V|)$ and 0% for $k = \sqrt{|V|}$. This results are shown in $diff\ edges$ column. Another experiment was carried out to study the influence of k on the DECWA final result and see if it is efficient in the case where k is of the order of $log(|V|)$. For each dataset, DECWA is called with $k = 3$, $k = log(|V|)$ and $k = \sqrt{(|V|)}$. The value of h is estimated by our proposed Elbow method. The values of γ and τ are set by using the random-search strategy. We have found (table IV.5) that the greater the k, the better the ARI. But between $k = log(|V|)$ and $k = \sqrt{(|V|)}$ the difference is only of 0.02. Regarding the outlier ratio, the difference between $k = log(|V|)$ and others is negligible. $ncsb$ stands for the number of connected subgraphs in case where G is not connected. On average, the smaller the $ncsb$, the larger the ARI. At the same time, $ncsb$ decreases as k grows. In the context of DECWA, the connectivity of the graph brings a better precision

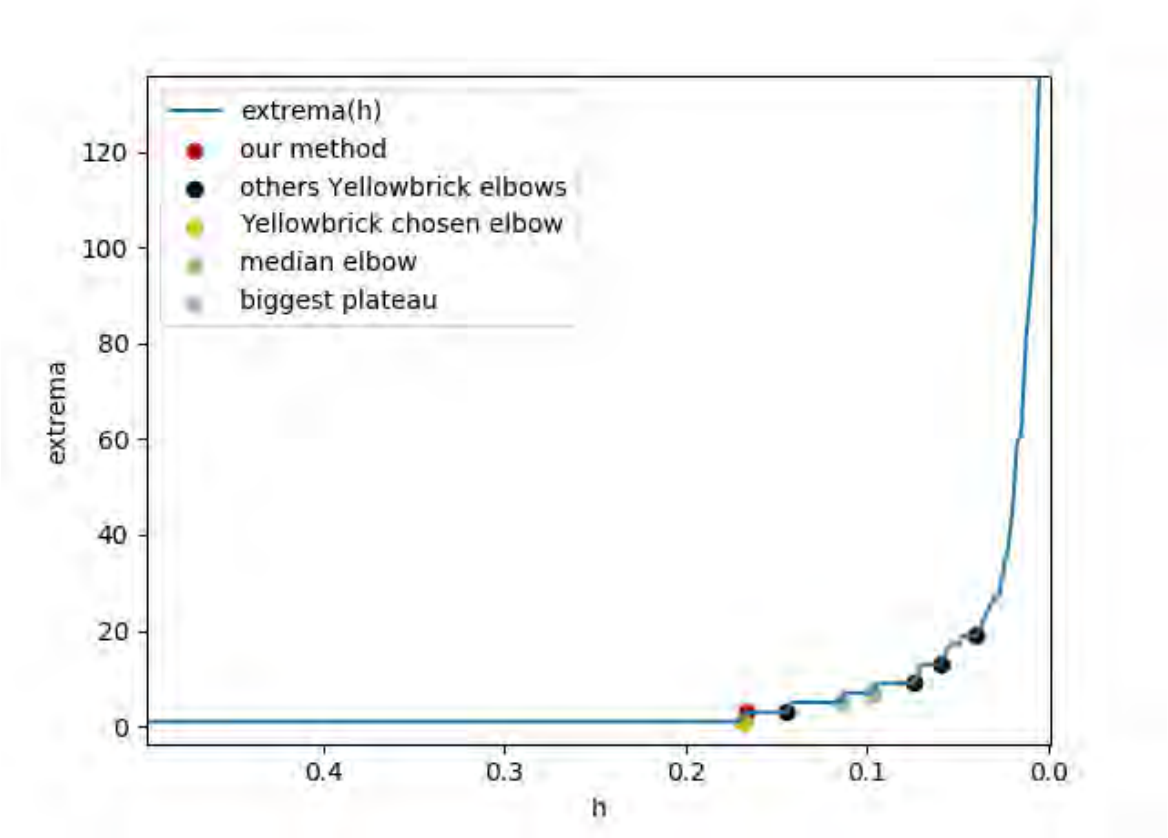


Figure IV.6: Elbow shaped curve of extrema in function of h for jain dataset

k	ARI	outliers(%)	ncsb	diff edges (%)
3	0.57	0.02	3.50	2.3
$log(\|V\|)$	0.65	0.03	1.50	0.2
$\sqrt{(\|V\|)}$	0.67	0.02	1.10	0

Table IV.5: Influence of k

in terms of clustering but it is not a prerequisite. If $k = log(|V|)$ or $k = \sqrt{(|V|)}$, even if the connectivity is not fully ensured DECWA remains efficient. Therefore, $k = log(|V|)$ might be a valid choice to consider rather than $k = \sqrt{|V|}$ if speed is required.

IV.10 Conclusion

Density-based methods can detect clusters with arbitrary shapes. However, they have difficulties operating in high-dimensional data and separating outliers from low-density clusters or close clusters of similar density.

To face these difficulties, we proposed Decwa, a solution based on a probabilistic approach and the spatial density concept. First of all, it structures the dataset in a minimum spanning tree (MST), where the nodes are the points and the edges are the distances between points, in order to obtain a unidimensional space of inter-

point distances. A density probability function ($p.d.f$) is applied to this new space to identify the level of representability of different distances. From this function and the MST, sub-clusters of points with different spatial densities are extracted. This is carried out by a strategy we have developed based on particular thresholds called mid-points to obtain the most homogeneous sub-clusters possible in density. Thus, in a high-dimensional data space, low-density clusters and outliers are separated as well as clusters that are densely and spatially close. Some sub-clusters are then merged (because belonging to the same real cluster) by a process based on the MST we have designed. This in order to bring the final result closer to the ground-truth. The inter-sub-cluster merging is carried out according to conditions based on the minimum spatial inter-cluster distance and the Wasserstein inter-cluster probabilistic distance.

Experimentally, DECWA (El Malki et al., 2020a) has been compared to density-based methods among the best known and performing methods in the literature on various data sets and different distances. DECWA is significantly better than competitors with a margin of 23% in clustering quality. More precisely, on high dimensional datasets, it outperforms competitors by a wide margin of 29%. DECWA is better by 20% and 23%, respectively, on datasets with low-density clusters and datasets with densely and spatially close clusters.

Chapter V

Conclusion & Perspectives

Clustering is a technique used to find automatically relationships in data that are either too slow or too difficult to analyze by humans (Aggarwal and Reddy, 2013). It has the advantage of being able to find relationships in more than three dimensions, which is a limitation for humans in terms of analysis. The main particularity of clustering is that it works on unlabeled data, i.e., without knowing a priori their nature (e.g., if the data is vehicles, nature can be the car, bicycle, plane, etc.). Concretely, clustering aims to divide a dataset into several groups of points (clusters) so that a cluster's points are as similar as possible.

Despite being widely used in practice, clustering has to face different challenges:

- in addition to the difficulty of storing large datasets, there is a more important problem, which is the automatic analysis of this data to derive knowledge.
- the greater the number of dimensions, the more difficult it is for clustering methods to work correctly;
- due to its unsupervised aspect, knowledge of the actual number of clusters existing in a dataset is difficult to access;
- another challenge lies in the internal structure of the data. Indeed, datasets could correspond to data distributions that are difficult to exploit.

Different clustering methods exist and are grouped into paradigms. In this manuscript, we are interested in two of them: partition-based methods and density-based methods. In the first paradigm, we focus on k-means (Llyod's version (Lloyd, 1982a)) because it is algorithmically practical and is one of the most widely used algorithms in data analysis.

In this manuscript, we have presented three contributions to address the problems mentioned above. For each paradigm, different problems are solved. For k-means, we made two contributions Sk-means and Kd-means. In the paradigm of density-based methods, we developed the Decwa solution.

Sk-means

The k-means standard version (Lloyd's version (Lloyd, 1982a)) divides a data set of size n into k clusters while minimizing an objective function (the quadratic error). It improves data partitioning from one iteration to the next until convergence. However, this version becomes more and more expensive as k and n increase, which is impractical for large sets of points. Moreover, it does not include strategies to avoid unnecessary and repetitive calculations. So to reduce the latter, versions of k-means based on geometric reasoning (Newling and Fleuret, 2016) have been developed. They are based on a property of distance metrics, called triangular inequality, attempting to avoid calculating unnecessary point-centroid distances. Nevertheless, this strategy does not avoid all unnecessary and repetitive calculations through several iterations.

To address this problem, we proposed an optimization strategy, called Sk-means, to reduce more unnecessary calculations in Lloyd's version and in versions based on geometric reasoning. Its main objective is to detect the points where the calculations performed on them do not improve the data partitioning. These points correspond to points that no longer change cluster from a given iteration to the end of the process of a k-means version. First of all, we introduced, for each cluster, two types of subsets. The first one is the kernel of the cluster. It contains only points called passive. They correspond to points considered as not improving clustering if they are subject to distance calculations. These calculations are then considered unnecessary. The second is the outer part containing points of the cluster that are likely to change cluster and thus contribute to clustering evolution. During iterations, the points move from the outer part to the kernel. We have designed an estimator E to estimate at which iteration the point of the outer part must pass into the kernel. The advantage of this strategy occurs as soon as the points switch into the kernel. At that moment, the points are no longer involved in the calculations until the k-means process end. So at each iteration, k distances are avoided for all points belonging to the kernel. In addition, in versions based on geometric reasoning, the variables associated with each point of the kernel are no longer updated and maintained. This is not the case for the points in versions not adopting Sk-means where the variables are treated at each iteration. So, the Sk-means advantages result in significant time-saving. This gain is all the more important as n, k and the number of iterations is large.

Two types of experiments were conducted. The first allowed the model of the E estimator to be inferred. 1750 synthetic data sets were generated, having different cardinalities, dimensions, number of clusters and data spatial distributions. A total of 3500 experiments were launched. The results were visually and statistically validated to infer E. In the second experiment, versions of k-means integrating Sk-means (Lloyd, Compare-means, Sort-means and Exponion) were compared against their original associated versions while using our E estimator. The datasets are various (synthetic

and real), including images. A total of 384 experiments have been launched wherein each one we compare a version with Sk-means with its associated original version (i.e., without Sk-means).

The comparison criteria were the execution time and the relative difference between the final values proposed by the objective functions (i.e., the minimum quadratic errors they found at the end of the k-means process). In execution time, versions with Sk-means are up to 191 times faster than Lloyd's version. They are up to more than five times faster than the non-Sk-means related versions. Apart from this time saving, the overall relative difference is negligible and does not exceed 1%. This shows that the data partitionings calculated by the versions with and without Sk-means are practically similar.

In terms of short-term perspectives, the passive nature of the points could be extended to clusters as well. Suppose the cluster has no output and input of points from a given iteration and during a certain number of consecutive iterations. In that case, it is considered as not contributing to the partitioning evolution. So this cluster is removed from calculations involving inter-center distances to save more execution time.

In the medium term, we could extend the previous perspective by considering that the cluster is passive if there are few entries and exits of points. These points are probably at the cluster boundary, and their influence on data partitioning is negligible. This new consideration increases the time saving because the cluster will be considered passive more quickly in a k-means version process compared to the previous definition of cluster passivity. Another improvement could be made on the E estimator. Currently, the E estimator produces a global estimate, i.e., it considers that all points stabilize at the same speed. The goal here is to individualize the stability estimate, i.e., to consider that each point has its own speed of stabilization. This could be done by integrating into the E model the distance between the point and its centroid. If it is very close to the centroid, then it should stabilize faster than the one far away from it. Another perspective could be to study the behavior of Sk-means, at least its adaptation to Lloyd's version, in shared-memory and shared-nothing architectures.

Kd-means

K-means requires from the user the number of k clusters before forming the clusters. If the user provides a wrong value of k, then the data partitioning may not make sense or be unusable. The estimation of k has been studied in the literature. The X-means and G-means methods are among the best known and most efficient. They correspond to iterative methods based on information theory and, more generally, on the probabilistic approach.

These methods have limitations on different aspects. First, they require iteratively executing k-means on the entire data set. Thus they are more and more expensive as

the number of points and the number of iterations increases. Second, they exaggerate on the k real value estimate when the shapes of the clusters are not Gaussian (i.e., G-means and X-means expect the real clusters to be respectively Gaussian (spherical or elliptical) and isotropic Gaussian (strictly spherical)) or the clusters overlap.

We have designed Kd-means to automatically estimate k while meeting the above limitations. It can detect clusters of more complex shapes (thus not necessarily Gaussian) even if they overlap. Kd-means is based on a hierarchical binary Kd-tree data structure and a hierarchical process. Kd-means consists of four consecutive steps. Firstly, the data are organized hierarchically through Kd-tree. This organization can be seen as a binary and recursive subdivision of the data. We have formulated conditions on the construction of Kd-tree so that it is not very deep and does not contain leaves with few points, to be suitable for large data sets. The second step forms clusters in each of the Kd-tree sheets. These clusters are indeed sub-clusters of the final clusters returned to the user. So after this step, a hierarchical process is launched on Kd-tree to merge clusters. The merge is performed recursively from the leaves to the root. In the root are then the final clusters. To carry out this process and meet the limits of overlapping and shapes of clusters, we have developed four criteria for inter-cluster merging. As soon as one seeks to verify the mergeability of two clusters, the criteria must be validated to merge them. The fourth step is optional and consists in improving the final data partitioning. It either assigns small clusters to larger clusters or groups the small clusters together to form a single cluster, or considers the small clusters as separate clusters because they are quite far apart from each other.

We compared the robustness of Kd-means against G-means and X-means. The types of data experienced are varied. 135 large synthetic data sets were generated with different cardinalities, dimensions and number of real clusters. Similarly, 11 large and real data sets were used, including spectral images, black/white or grayscale images, sound and logs. The comparison was performed on three different metrics: k estimation accuracy, partitioning quality and execution time. The experiments clearly showed that Kd-means is very competitive compared to G-means and X-means, even if the clusters overlap or have a complex shape. As a result, it is much better in quality and accuracy in estimating k. In addition, it was up to 1500 times faster than competitors.

In the short term, we could evaluate our solution to other tree data structures. For example, using Quadtree or Octree(Samet, 2005) instead of Kd-tree allows having for each node up to four child nodes or eight. This assumes that the data space is recursively divided into four or eight instead of two. So by increasing the number of child nodes, we would expect a better separation of the data.

In the medium term, another close alternative to Kd-tree could be Ball-tree. In the latter, the subspaces of data resulting from partitioning correspond to hyperspheres.

So, if we use Ball tree in our solution, the calculations of fusion tests are performed on hyper-spheres instead of hyper-rectangles. The spatial distribution of the cluster points could be more accurate with hyper-spheres than hyper-rectangles although they are more expensive to compute. Another interesting binary tree data structure to implement is Vantage point trees (Vp-tree)(Nielsen et al., 2009). The process of partitioning spaces does not depend on the dimension but the distance metric. For each node, its left child node is associated with a hyper-sphere. The other child is associated with everything external to this hyper-sphere. Another perspective consists in associating a weight vector to each data point where each ith weight corresponds to the degree of belonging of the point to cluster j. This weight is calculated according to its distance from each of the centroids or hyper-rectangles. This operation is performed only at the Kd-tree root.

Decwa

The third contribution deals with the density-based methods paradigm. It corresponds to a different clustering approach compared to partition-based methods, mainly on cluster definition. A cluster is a dense region of points separated from other clusters by less dense regions. This paradigm's methods can detect clusters of arbitrary shapes without having to inform the number of clusters a priori.

Nevertheless, they present difficulties when dealing with complex spatial data distributions. Among these difficulties, they may consider low-density clusters considered as groups of outliers or have them absorbed by a cluster of greater density while these low-density clusters represent a phenomenon in their own right. Moreover, they can lead clusters of similar density and very close in space to form a single cluster when they should not be. The third difficulty is related to the phenomenon of the curse of dimensionality. Indeed, when the data are of very high dimensionality, the methods may not work properly as for low dimensionality data.

We have designed, Decwa, a density-based clustering algorithm based on a probabilistic approach. Decwa proposes a new way of dividing into homogeneous clusters and a new way of aggregating clusters. In an environment of high dimensional data spaces, these two methods lead to the separation, on the one hand, between low-density clusters and outliers and, on the other hand, between clusters that are close in density and spatially. Decwa consists of four consecutive steps. First, the dataset is projected on a one-dimensional space via the minimum spanning tree (MST) where nodes are points and edges are inter-point distances. So the new space consists only of spatial inter-point distances. Then, from this space, a probability density function ($p.d.f$) is computed to obtain different distances' representation level. Then the process of dividing the set of points into clusters begins. In this process, we have proposed a new way of extracting clusters via the $p.d.f$ jointly exploited with the MST. It results on homogeneous clusters on the aspect of spatial density, i.e., each cluster follows its

own probability law. These clusters are only intermediate and may need to be merged to produce final clusters. For this, we have proposed an agglomeration method. Two clusters are merged if only they are relatively close spatially and in probability laws. The spatial proximity is based on the distance proposed by the user, while the probabilistic proximity is calculated by Wasserstein distance. Agglomeration is performed using the MST to avoid having to recalculate the inter-point distances.

Experimentally, we compared the robustness of Decwa against Hdbscan, Denclue and Dbclasd, the most well-known and efficient methods among density-based methods. The datasets used are quite varied in type, cardinalities and dimensions. The actual data include biological, medical, botanical and textual data. Different distances have been used, including Cosine, Manhattan, Braycurits and Euclidean distances. In terms of clustering quality, Decwa outperformed its competitors by a wide margin of 23%. Especially on very high dimensional data (containing up to 26833 dimensions), Decwa is, on average, 29% better than the others. Moreover, Decwa has the best results on datasets with low-density clusters with a margin of 20%. For datasets with similar density and spatial clusters, it is better with a margin of 23%.

In the short term, other kernels of the kernel density estimation method could be tested. For example, apart from the Gaussian kernel used in Decwa, it is interesting to study the impact of cosine, triangular, triweight and uniform kernels during cluster extraction. Moreover, Decwa could be experimented on more complex data such as multidimensional time series.

In the long term, Decwa could belong to the multi-view clustering methods approach (Fu et al., 2020). At this stage, it will be able to cluster data from heterogeneous data sources. Indeed, data could be represented under different views. A view corresponds to a data representation. In the presence of these different data sources, it is necessary to deploy strategies that extract the most useful knowledge from each source to produce a global clustering closer to the ground truth than using only one data source.

Appendices

In this part, we complete the content of the related work chapter as well as the Kd-means experimental part.

.1 Additional related work

We extend the the optimization strategies for k-means section II.4. Indeed in this section, we discussed the optimizations brought specifically to k-means. In the appendices, specifically in .1.1, we discuss optimization techniques for supervised and unsupervised classification methods. Then in .1.2, we complete the part on k-means versions based on geometric reasoning (II.4.1.5).

.1.1 General strategies

This sub-section extends the optimization strategies for k-means section II.4. We discuss strategies that improve algorithms' execution time (supervised, unsupervised or semi-supervised). The first three strategies focus on distances, i.e., functions that measure similarity between points. Indeed, distance is an essential function in many algorithms performing classification. So accelerating the distance calculation accelerates the algorithms' execution time. The last strategy is devoted to dimension processing. The distance functions and other tools necessary for classification are executed on data points and thus on their dimensions. In this sense, dimensions are also concerned with optimization strategies. These could lead to considerable execution time acceleration and improve classification quality if the reduced format data are more reliable for classification methods than the original data.

.1.1.1 Distance reformulation

The distance, a function measuring the similarity between two points, can be broken down into several independent parts. Some parts can be calculated and stored in advance of the process of an algorithm and remain unchanged. Only the other parts could be calculated during the process. In this way, the calculation of distances is accelerated in time. As an example, the squared Euclidean distance can be expressed from the inner products (equation 1). So in this equation, if we consider that x_i is a point in the dataset known in advance and c is another point that remains to be determined during the process, we could still calculate and store $< x_i, x_i >$ in advance (Hamerly and Drake, 2015a).

$$||x_i - c||^2 =< x_i, x_i > -2 < x_i, c > + < c, c > \quad (1)$$

.1.1.2 Reduced distance

To speed up the process execution time, the clustering algorithms replace the distances they usually use with faster alternatives called reduced distances (Hamerly, 2010). A reduced distance is an amount of dissimilarity that is more efficient to compute than the actual distance, but which preserves relative rankings. It is suitable when one is not looking for the value of the actual distance itself but just the relative orders of the points in relation to a given point as generated by the actual distance. In this case, it is more convenient to use an alternative distance than the actual distance.

For example, in k-means, the used distance is the Euclidean distance. The alternative distance could be the squared Euclidean distance; this avoids calculating the square root.

.1.1.3 Partial distance search (PDS)

PDS (Bei and Gray, 1985) is a way to speed up the search for the nearest points. It is used in algorithms where knowledge of exact distances is not necessary but only minimal distance. This is the case for k-means, when searching for the nearest centroid for a given point x. Suppose we already have m the distance between x_i and c_1 and want to know if c_2 is closer to x_i than c_1. The distance between x and c_2 can be computed according to the equation 2. The value of p is the dimension index such that $0 \leq p < d$ with d the number of dimensions. As p is incremented, the sum in the equation 2 increases. If it exceeds m, for a value of p, the distance calculation is then stopped because it cannot be close to x_i more than c_1 even if the distance calculation process continues to the last dimension.

$$dis(x_i, c_2) = \sum_{p=0}^{d-1} dis(x_{i,p}, c_{2,p}) \tag{2}$$

.1.1.4 Dimensionality reduction

The calculation of distances between points involves taking into account the dimensions of these points. Indeed, the greater the number of dimensions, the more expensive the distance is. Another consequence of having a large number of dimensions is that the data space volume is very large. As a result, in this space, data points often constitute a small unrepresentative dataset, making the classification task challenging. This problem is known as the curse of dimensionality.

The dimensionality reduction (Cunningham and Ghahramani, 2014) is a solution to reduce the both problems. It is an operation transforming a dataset from a large dimensional space to a smaller dimensional space. Although it is used for 2D or 3D visualization, it used to keep only the important information in the data and to reduce the algorithms' execution time (Boutsidis et al., 2010). Dimensionality reduction is

type	advantages	limitations
distance reformulation	speed up distance calculations	requires to store for each point a list of pre-calculated results
reduced distance		not suitable for methods requiring precise distances
pds	speeds up distance calculations on high-dimensional data	
dimensionality reduction	reduces the algorithmic complexity in time of the calculations performed on the points because the dimension is considerably reduced	classification quality affected if significant useful information is lost

Table 1: Advantages and drawbacks of general optimization strategies

usually applied upstream of the classification process. It is at the process beginning that the space resulting from the dimensionality reduction is exploited.

Different approaches to dimension reduction exist. Dimension selection (Chandrashekar and Sahin, 2014) consists in choosing as many as possible non-redundant and reliable dimensions according to pre-defined metrics. Another known approach, based on linear algebra (e.g., PCA, SVD) (Boutsidis et al., 2010, Tipping and Bishop, 1999), performs a projection from the high dimensional data space into a lower-dimensional space while trying to preserve essential properties of the original data. This projection leads to new dimensions. The reduction is also carried out by deep neural networks (autoencoders)(Makhzani et al., 2015). They consist in learning how to encode a dataset and then decode it to reproduce the original data. Indeed, the data in encoded format represents the dataset in lower dimensions that could be used by other algorithms.

We point out that the reduction of dimensionality could have a significant time cost. If several executions of the classification method are performed and the reduction of dimensionality is called once. In that case, there could be a significant time-saving.

.1.1.5 Conclusion

The first three above-mentioned strategies (table 1) are dedicated to accelerating the distance calculation. The distance reformulation decomposes the distance function into sub-functions, among which the results are obtained and stored in advance of the classification process. It requires nevertheless to store for each point a list of pre-calculated results. The reduced distance strategy associates to a distance another equivalent one less costly in time. During the process, the alternative is used instead of the actual distance. Even if both distances do not produce the same similarity values, they propose the same ordered list of points (e.g., in case of sorting task). The partial

distance search (PDS) approximates a distance value between two points using only a subset of dimensions to assert a given hypothesis is true. It is applicable for large spaces and also when actual distance values are not required. The reduced distance and PDS strategies are not suitable for classification methods requiring precise distances. However, they are suitable for nearer neighbor calculations, usually requiring only sorting the points. The fourth strategy, dimensionality reduction, does not implicitly accelerate the execution time but it could contribute to it indirectly. The strategy produces another subset of dimensions from the original dimensional space. This new space is then consumed by a classification method. Although this space reduction could lead to an acceleration of runtime, the classification quality could be poorer or even worse if there was too much loss of useful information during the reduction.

.1.2 Compare-means

$$2 \times dis(x_i, G_b) < dis(G_b, G_a) \tag{3}$$

We complete the section on versions of k-means based on geometric reasoning and particularly Compare-means (II.4.1.5.1).

Let $x_i, G_a, G_b \in R^d$, x_i a point, G_b the actual centroid of x_i and G_a another centroid. In the following, we prove that Compare-means assertion based the inequation 3 is true. Indeed, the assertion states that if this inequation is validated, then G_a cannot be closer to x_i than G_b. Therefore, there is no need to calculate the distance between x_i and G_a. To prove the veracity of the assertion , the following triangular inequality is used:

$$dis(G_a, G_b) \leq dis(G_a, x_i) + dis(x_i, G_b) \tag{4}$$

which implies

$$dis(G_a, G_b) - dis(x_i, G_b) \leq dis(G_a, x_i) \tag{5}$$

, by replacing in the left side $dis(G_a, G_b)$ by II.6 (inequality that compare-means considers true), we obtain :

$$2 \times dis(x_i, G_b) - dis(x_i, G_b) \leq dis(G_a, x_i) \tag{6}$$

So $dis(x_i, G_b) <= dis(G_a, x_i)$ which proves that if $2 \times dis(x_i, G_b) < dis(G_a, G_b)$ then necessarily G_a cannot be closer to x_i than G_b.

.2 Kd-means

.2.1 BIRCH

Although it has some similarity with Kd-means because it performs clustering via a tree structure called CF-Tree, it is nevertheless not comparable to Kd-means because it has not been specifically developed to estimate the number of clusters k. Moreover, Birch should be followed by another clustering algorithm (e.g., k-means) to refine its clustering result. Consequently, the problem of estimating k clusters reappears because k is necessary to produce the final result of clusters. However, we experimented Birch to study the clustering results produced by its hierarchical process.

Birch (Balanced Iterative Reducing and Clustering Hierarchies) (Zhang et al., 1996) is a clustering method proposing a hierarchical data structure called CF-tree in which it integrates the points. In CF-tree, each node can have no more than B subclusters whose radius R does not exceed a threshold T. The Birch process consists of sequentially integrating the points into CF-Tree. At each given point, the tree is traversed to the nearest leaf, and then the nearest subcluster is identified in the leaf. If the addition of this point in this subcluster involves $R > T$, then a new subcluster is added. If adding leads to the number of subclusters exceeding B, then the leaf is split in two, which are initialized with the furthest subclusters. Then the others are assigned to one of the two nearest leaves. This task of updating and adding nodes is repeated from child to father up to the root. If, during the process, the tree does not fit in memory, then the tree is rebuilt while aggregating the subclusters. One of its advantages is that it does not require the value of k. Nevertheless, it tends to produce a large number of clusters. A clustering algorithm is usually needed (k-means or hierarchical clustering algorithm) that clusters the final Birch results. This brings us back to the problem of how to estimate k. Moreover, the clusters it produces are spherical, so it has the same limitations as k-means in terms of the types of clusters to be found.

In this part we compare the Birch (Zhang et al., 1996) clustering algorithm with the three others presented in the section III.3.8 . Note that we consider the same three metrics used to compare the three algorithms. In the experiments we focus only on subclusters of the CF-tree leaves. The value of threshold T has been initialized as follows: $T = R_{root} * q$ where R_{root} the average distance between the points and the centroid of X and $q \in]0, 1]$. T is then updated during Birch execution when the CF-Tree is reconstructed as recommended by the Birch authors. Similarly, the number of subclusters per node B was also tuned. So, $B = \sqrt{card(X)} * b$ with X the dataset and $b \in]0.1]$. The results in the table correspond to the mean of the results.

.2.1.1 Analysis of Birch's results

Overall, in synthetic datasets, Birch overestimates the number of clusters compared to the actual k and even to the number of clusters produced by Kd-means and G-means. Birch overestimates with respect to X-means for $k = 10$ and $k = 32$. The sensitivity of Birch to the order of the point entries, the overlap and shape of the clusters (Birch tends to capture only spherical clusters), the maximum number of subclusters required for each node, and the dataset's size results in Birch results that are not very homogeneous. For a given real k and two datasets with the same data generation process (synthetic datasets), it could produce from the first dataset hundreds of subclusters and the other thousands. For example, for $k = 10$ and $d = 8$, Birch produces 12622 subclusters for $n = 1 \times 10^6$ and 170 subclusters for $n = 4 \times 10^6$. In case of real datasets, Birch overestimates k in a very exaggerated way compared to competitors and the real k. Except for the dataset japeneseVowels when Birch is compared to G-means, both overestimate k.

Compared to quality measurement vi, Kd-means is clearly better than Birch. It outperforms it on all datasets. Birch slightly outperforms X-means and/or G-means, e.g. for $(k = 10, d = 6, n \in \{2 \times 10^6, 4 \times 10^6\})$ and $(k = 10, d = 8, n = 4 \times 10^6)$ in synthetic datasets. In real datasets, Birch produces better quality in only three datasets compared to G-means but on average it has a vi of 11.9 compared to 9.7 for G-means. It exceeds X-means only on one dataset.

In terms of time execution, even though it is incremental, the construction of CF-tree is slow. For $k = 10$, the maximum time performed by the other three algorithms is 3061 seconds (X-means), while the minimum time for Birch is 5296 seconds on synthetic datasets. The same trend is observed for the other k values. The average time executions (in seconds) of the four algorithms on the actual data are respectively for Birch, G-means, X-means, and Kd-means of 54,731, 25,770 , 4,938 and 136. Birch is more than twice as slow as G-means and 402 times slower than Kd-means. Several factors explain the slowness of Birch. They include 1) Calculating the distance between the point that is about to be added and several nodes and subclusters at each depth of CF-tree. 2) Calculating the distance between several subclusters in order to merge or fragment the node. 3) Constantly update and eventually split parent nodes recursively when adding a point. 4) Reconstruction of the CF-tree when it no longer fits in memory. 5) Medium and large dimensional datasets are difficult to support. 6) sensitivity to the order of points.

In summary, according to the analyses made on the quality measure vi and the estimate of k, Birch should be coupled with another algorithm to refine and aggregate the clusters it has produced. Depending on the execution time, it is slow compared to competitors. Coupling it to another algorithm (e.g., k-means) takes even more time, and the user has to provide as an input to the added algorithm a value of k.

			Birch		
k	d	n	k(estimated)	vi	Δt
10	4	1×10^6	5025	5.82	5296
		2×10^6	637	2.76	31726
		4×10^6	3920	4.31	84728
	6	1×10^6	119	2.85	6040
		2×10^6	48	1.34	30705
		4×10^6	169	1.60	89192
	8	1×10^6	12622	3.24	10943
		2×10^6	1968	2.34	41103
		4×10^6	170	1.70	62366
32	4	1×10^6	8881	4.72	9222
		2×10^6	224	1.77	53734
		4×10^6	6885	4.41	105730
	6	1×10^6	2080	1.79	11120
		2×10^6	403	2.71	39967
		4×10^6	553	1.95	98346
	8	1×10^6	1618	1.91	11528
		2×10^6	231	1.92	68853
		4×10^6	218	1.58	134361
80	4	1×10^6	377	1.99	9757
		2×10^6	460	1.50	54712
		4×10^6	377	1.78	214153
	6	1×10^6	1003	2.13	9047
		2×10^6	140	1.52	46668
		4×10^6	6642	1.81	169094
	8	1×10^6	357	1.78	12314
		2×10^6	371	1.26	32205
		4×10^6	341	1.34	165503

Table 2: Experimental results from Birch's execution of the synthetic data. Note that Δt is expressed in seconds.

dataset	k	d	n	k(estimated)	**vi**	Δt
vehicle	4	18	1×10^6	185367	14.9	19878
satimage	6	36	1×10^6	78816	8.8	15977
japaneseVowels	9	14	1×10^6	133	8.3	5586
fourier	10	76	1×10^6	121954	6.4	15513
pendigits	10	16	1×10^6	70560	15.9	39768
fashionmnist	10	784	70000	36655	12.8	217
ldpa	11	5	164860	87375	14.6	381
walking	22	4	149332	20076	7.1	159
letter	26	16	999999	10580	11.6	18430
zernike	47	47	1×10^6	81693	16.6	49564
emnist	62	784	697932	92767	14.1	55956

Table 3: Experimental results from Birch's execution of the real data. Note that Δt is expressed in seconds.

Bibliography

C. C. Aggarwal and C. K. Reddy. *Data Clustering: Algorithms and Applications*. Chapman Hall/CRC, 1st edition, 2013. ISBN 1466558210.

N. Ailon, Y. Chen, and H. Xu. Breaking the small cluster barrier of graph clustering. *CoRR*, abs/1302.4549, 2013.

E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010. ISBN 026201243X.

A. Amine, Z. Elberrichi, L. Bellatreche, M. Simonet, and M. Malki. Concept-based clustering of textual documents using SOM. In *The 6th ACS/IEEE International Conference on Computer Systems and Applications, AICCSA 2008, Doha, Qatar, March 31 - April 4, 2008*, pages 156–163. IEEE Computer Society, 2008. doi: 10.1109/AICCSA.2008.4493530. URL `https://doi.org/10.1109/AICCSA.2008.4493530`.

M. Ankerst, M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. volume 28, pages 49–60, 06 1999.

D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *ACM-SIAM symposium on Discrete algorithms*, pages 1027–1025, 2007. ISBN 9780898716245.

D. Ayres-de Campos, J. Bernardes, A. Garrido, J. Marques-de Sá, and L. Pereira-Leite. Sisporto 2.0: A program for automated analysis of cardiotocograms. *The Journal of Maternal-Fetal Medicine*, 9(5):311–318, 2000.

C.-D. Bei and R. Gray. An improvement of the minimum distortion encoding algorithm for vector quantization. *IEEE Transactions on Communications*, 33(10): 1132–1133, oct 1985. ISSN 0090-6778. doi: 10.1109/TCOM.1985.1096214. URL `http://ieeexplore.ieee.org/document/1096214/`.

J. Bennett and W. Briggs. *Using and Understanding Mathematics: A Quantitative Reasoning Approach*. Pearson Education, 2014. ISBN 9780321912343. URL `https://books.google.fr/books?id=_6xKBAAAQBAJ`.

F. Bentayeb and C. Favre. Rok: Roll-up with the k-means clustering method for recommending OLAP queries. In S. S. Bhowmick, J. Küng, and R. R. Wagner, editors, *Database and Expert Systems Applications, 20th International Conference, DEXA 2009, Linz, Austria, August 31 - September 4, 2009. Proceedings*, volume 5690 of *Lecture Notes in Computer Science*, pages 501–515. Springer, 2009. doi: 10.1007/978-3-642-03573-9_43. URL https://doi.org/10.1007/978-3-642-03573-9_43.

J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, September 1975. ISSN 0001-0782. doi: 10.1145/361002.361007.

P. Berkhin. *A Survey of Clustering Data Mining Techniques*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978.

C. Boutsidis, A. Zouzias, and P. Drineas. Random projections for k-means clustering. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 298–306. Curran Associates, Inc., 2010. URL http://papers.nips.cc/paper/3901-random-projections-for-k-means-clustering.pdf.

P. Bradley, O. Mangasarian, and N. Street. Clustering via concave minimization. pages 368–374, 01 1996.

P. S. Bradley and U. M. Fayyad. Refining initial points for k-means clustering. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, page 91–99, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1558605568.

M. Brito, E. Chávez, A. Quiroz, and J. Yukich. Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection. *Statistics Probability Letters*, 35:33–42, 08 1997. doi: 10.1016/S0167-7152(96)00213-1.

R. Campello, D. Moulavi, and J. Sander. Density-based clustering based on hierarchical density estimates. volume 7819, pages 160–172, 04 2013.

M. E. Celebi, H. A. Kingravi, and P. A. Vela. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications*, 40(1):200 – 210, 2013. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2012.07.021. URL http://www.sciencedirect.com/science/article/pii/S0957417412008767.

G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers Electrical Engineering*, 40(1):16 – 28, 2014. ISSN 0045-7906. doi: https://doi.org/10.

1016/j.compeleceng.2013.11.024. URL http://www.sciencedirect.com/science/article/pii/S0045790613003066. 40th-year commemorative issue.

S. V. Chekanov and J. Proudfoot. Searches for tev-scale particles at the lhc using jet shapes. *Phys. Rev. D*, 81:114038, Jun 2010. doi: 10.1103/PhysRevD.81.114038. URL https://link.aps.org/doi/10.1103/PhysRevD.81.114038.

A.-G. Chifu, F. Hristea, J. Mothe, and M. Popescu. Word sense discrimination in information retrieval: A spectral clustering-based approach. *Information Processing Management*, 51(2):16 – 31, 2015. ISSN 0306-4573. doi: https://doi.org/10.1016/j.ipm.2014.10.007. URL http://www.sciencedirect.com/science/article/pii/S0306457314001046.

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, Sept. 2006. ISSN 0018-9448. doi: 10.1109/TIT.1967.1053964.

J. P. Cunningham and Z. Ghahramani. Linear Dimensionality Reduction: Survey, Insights, and Generalizations. *Journal of Machine Learning Research*, 16: 2859–2900, 2014. URL http://www.jmlr.org/papers/volume16/cunningham15a/cunningham15a.pdfhttp://arxiv.org/abs/1406.0873.

R. A. Davis, K.-S. Lii, and D. N. Politis. Remarks on Some Nonparametric Estimates of a Density Function. In *Selected Works of Murray Rosenblatt*, pages 95–100. Springer New York, 2011.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 01 2006.

I. S. Dhillon and D. S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In M. J. Zaki and C.-T. Ho, editors, *Large-Scale Parallel Data Mining*, pages 245–260, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-46502-7.

D. Donoho. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture*, pages 1–32, 01 2000.

D. Dua and C. Graff. UCI machine learning repository, 2017.

N. El Malki, F. Ravat, and O. Teste. k-means improvement by dynamic pre-aggregates. In J. Filipe, M. Smialek, A. Brodsky, and S. Hammoudi, editors, *Proceedings of the 21st International Conference on Enterprise Information Systems,*

ICEIS 2019, Heraklion, Crete, Greece, May 3-5, 2019, Volume 1, pages 133–140. SciTePress, 2019a. doi: 10.5220/0007675201330140. URL https://doi.org/10.5220/0007675201330140.

N. El Malki, F. Ravat, and O. Teste. Accélération de k-means par pré-calcul dynamique d'agrégats. In M. Rousset and L. Boudjeloud-Assala, editors, *Extraction et Gestion des connaissances, EGC 2019, Metz, France, January 21-25, 2019*, volume E-35 of *RNTI*, pages 255–260. Éditions RNTI, 2019b. URL http://editions-rnti.fr/?inprocid=1002488.

N. El Malki, R. Cugny, O. Teste, and F. Ravat. Decwa: Density-based clustering using wasserstein distance. In *Proceedings of the 29th ACM International Conference on Information amp; Knowledge Management*, CIKM '20, page 2005–2008, New York, NY, USA, 2020a. Association for Computing Machinery. ISBN 9781450368599. doi: 10.1145/3340531.3412125. URL https://doi.org/10.1145/3340531.3412125.

N. El Malki, F. Ravat, and O. Teste. Kd-means: Clustering method for massive data based on kd-tree. In I. Song, K. Hose, and O. Romero, editors, *Proceedings of the 22nd International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data co-located with EDBT/ICDT 2020 Joint Conference, DOLAP@EDBT/ICDT 2020, Copenhagen, Denmark, March 30, 2020*, volume 2572 of *CEUR Workshop Proceedings*, pages 26–35. CEUR-WS.org, 2020b. URL http://ceur-ws.org/Vol-2572/paper7.pdf.

C. Elkan. Using the triangle inequality to accelerate k-means. *ICML-2003*, 2003. doi: 10.1016/0026-2714(92)90278-S.

M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. volume 96, pages 226–231, 01 1996.

A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Y. Zomaya, S. Foufou, and A. Bouras. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE Transactions on Emerging Topics in Computing*, 2(3):267–279, 2014.

A. G. Foina, J. Planas, R. M. Badia, and F. J. Ramirez-Fernandez. P-means, a parallel clustering algorithm for a heterogeneous multi-processor environment. In *Proceedings of the 2011 International Conference on High Performance Computing and Simulation, HPCS 2011*, pages 239–248. IEEE, jul 2011. ISBN 9781612843810.

E. Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 1965. ISSN 00201669.

P. Fränti and S. Sieranoja. K-means properties on six clustering benchmark datasets, 2018. URL `http://cs.uef.fi/sipu/datasets/`.

L. Fu, P. Lin, A. V. Vasilakos, and S. Wang. An overview of recent multi-view clustering. *Neurocomputing*, 402:148 – 161, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2020.02.104. URL `http://www.sciencedirect.com/science/article/pii/S0925231220303222`.

G. Gan, C. Ma, and J. Wu. *Data Clustering: Theory, Algorithms, and Applications (ASA-SIAM Series on Statistics and Applied Probability)*. Society for Industrial and Applied Mathematics, 2007. ISBN 0898716233.

S. Garía and F. Herrera. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research - JMLR*, 9, 12 2008.

J. C. Gower and G. J. S. Ross. Minimum spanning trees and single linkage cluster analysis. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 18 (1):54–64, 1969.

R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. 1994.

M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17, 10 2001. doi: 10.1023/A:1012801612483.

G. Hamerly. Making k -means even faster. *2010 SIAM international conference on data mining (SDM 2010)*, pages 130–140, 2010. doi: 10.1137/1.9781611972801.12. URL `http://www.siam.org/proceedings/datamining/2010/dm10{_}012{_}hamerlyg.pdf{%}5Cnhttp://cs.baylor.edu/{~}hamerly/papers/sdm{_}2010.pdf`.

G. Hamerly and J. Drake. Accelerating lloyd's algorithm for k–means clustering. In *Partitional Clustering Algorithms*, pages 41–78. Springer International Publishing, jan 2015a. ISBN 9783319092591. doi: 10.1007/978-3-319-09259-1_2.

G. Hamerly and J. Drake. Accelerating lloyd's algorithm for k-means clustering. pages 41–78, 10 2015b. doi: 10.1007/978-3-319-09259-1_2.

G. Hamerly and C. Elkan. Learning the k in k-means. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, pages 281–288. MIT Press, 2003.

E.-H. Han and G. Karypis. Centroid-based document classification: Analysis and experimental results. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, page 424–431. Springer-Verlag, 2000.

J. A. Hartigan. *Clustering Algorithms*. John Wiley Sons, Inc., USA, 99th edition, 1975. ISBN 047135645X.

T. Hastie, R. Tibshirani, and J. Friedman. *Introduction*, pages 1–8. Springer New York, New York, NY, 2009. ISBN 978-0-387-84858-7. doi: 10.1007/978-0-387-84858-7_1. URL `https://doi.org/10.1007/978-0-387-84858-7_1`.

A. Hinneburg and H.-H. Gabriel. Denclue 2.0: Fast clustering based on kernel density estimation. volume 4723, pages 70–80, 09 2007.

A. Hinneburg and D. Keim. An efficient approach to clustering in large multimedia databases with noise. *First publ. in: Proceedings of the 4th International Conference on Knowledge Discovery and Datamining (KDD'98), New York, NY, September, 1998, pp. 58-65*, 98,, 03 1999.

V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22:85–126, 10 2004. doi: 10.1023/B:AIRE.0000045502.10941.a9.

C. . Huang and R. W. Harris. A comparison of several vector quantization codebook generation approaches. *IEEE Transactions on Image Processing*, 2(1):108–112, 1993.

J. Huang. A fast clustering algorithm to cluster very large categorical data sets in data mining. In *DMKD*, 1997.

J.-J. Huang, G.-H. Tzeng, and C.-S. Ong. Marketing segmentation using support vector clustering. *Expert Systems with Applications*, 32(2):313 – 317, 2007. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2005.11.028. URL `http://www.sciencedirect.com/science/article/pii/S0957417405003404`.

L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

A. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31: 651–666, 06 2010a.

A. K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, jun 2010b. ISSN 01678655.

T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. An efficient k-means clustering algorithm analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:881–892, 07 2002. doi: 10.1109/TPAMI.2002.1017616.

G. Karypis, Eui-Hong Han, and V. Kumar. Chameleon: hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

L. Kaufmann and P. Rousseeuw. Clustering by means of medoids. *Data Analysis based on the L1-Norm and Related Methods*, pages 405–416, 01 1987.

A. Kershenbaum and R. Van Slyke. Computing minimum spanning trees efficiently. In *Proceedings of the ACM Annual Conference - Volume 1*, ACM '72, page 518–527, New York, NY, USA, 1972. Association for Computing Machinery. ISBN 9781450374910. doi: 10.1145/800193.569966. URL `https://doi.org/10.1145/800193.569966`.

V. Y. Kiselev, T. S. Andrews, and M. Hemberg. Challenges in unsupervised clustering of single-cell rna-seq data. *Nature reviews. Genetics*, 20(5):273—282, May 2019. ISSN 1471-0056. doi: 10.1038/s41576-018-0088-9. URL `https://doi.org/10.1038/s41576-018-0088-9`.

M. Kleindessner, P. Awasthi, and J. Morgenstern. Fair k-center clustering for data summarization. volume 97 of *Proceedings of Machine Learning Research*, pages 3448–3457, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL `http://proceedings.mlr.press/v97/kleindessner19a.html`.

H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *TKDD*, 3, 01 2009.

H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek. Density-based clustering. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 1:231–240, 05 2011.

J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 1956.

T. Kucukyilmaz. Parallel k-means algorithm for shared memory multiprocessors. *Journal of Computer and Communications*, 02:15–23, 01 2014. doi: 10.4236/jcc.2014.211002.

A. Likas, N. Vlassis, and J. J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition*, 36(2):451 – 461, 2003. ISSN 0031-3203. doi: https://doi.org/10.1016/S0031-3203(02)00060-2. URL `http://www.sciencedirect.com/science/article/pii/S0031320302000602`. Biometrics.

Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95, 1980.

C. Liu, T. Hu, Y. Ge, and H. Xiong. Which distance metric is right: An evolutionary k-means view. In *Proceedings of the 12th SIAM International Conference on Data Mining, SDM 2012*, 2012. ISBN 9781611972320.

S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982a.

S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982b.

J. B. MacQueen. Some methods for classification and analysis of multivariate observations. 1967.

A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial autoencoders, 2015.

C. Mallah, J. Cope, and J. Orwell. Plant leaf classification using probabilistic integration of shape, texture and margin features. *Pattern Recognit. Appl.*, 3842, 02 2013.

S. Maneewongvatana and D. M. Mount. It's okay to be skinny, if your friends are fat. In *Center for Geometric Computing 4th Annual Workshop on Computational Geometry*, 1999.

L. McInnes and J. Healy. Accelerated hierarchical density based clustering. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 33–42, 2017.

M. Meilă. Comparing clusterings by the variation of information. In B. Schölkopf and M. K. Warmuth, editors, *Learning Theory and Kernel Machines*, pages 173–187, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45167-9.

A. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *In Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 397–405. AAAI Press, 2000.

J. Nešetřil, E. Milková, and H. Nešetřilová. Otakar borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Math.*, 233(1–3):3–36, Apr. 2001. ISSN 0012-365X. doi: 10.1016/S0012-365X(00)00224-7. URL `https://doi.org/10.1016/S0012-365X(00)00224-7`.

J. Newling and F. Fleuret. Fast k-means with accurate bounds. volume 48 of *Proceedings of Machine Learning Research*, pages 936–944, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `http://proceedings.mlr.press/v48/newling16.html`.

F. Nielsen, P. Piro, and M. Barlaud. Bregman vantage point trees for efficient nearest neighbor queries. In *2009 IEEE International Conference on Multimedia and Expo*, pages 878–881, 2009. doi: 10.1109/ICME.2009.5202635.

D. Pelleg and A. Moore. Accelerating exact k -means algorithms with geometric reasoning . In *KDD*, pages 277–281. ACM, 1999. doi: 10.1145/312129.312248. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.6396{&}rep=rep1{&}type=pdf`.

D. Pelleg and A. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *In Proceedings of the 17th International Conf. on Machine Learning*, pages 727–734. Morgan Kaufmann, 2000.

S. J. Phillips. Acceleration of k-means and related clustering algorithms. In D. M. Mount and C. Stein, editors, *Algorithm Engineering and Experiments*, pages 166–177, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45643-8.

R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957.

S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C.-H. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. P. Mesirov, T. Poggio, W. Gerald, M. Loda, E. S. Lander, and T. R. Golub. Multiclass cancer diagnosis using tumor gene expression signatures. *PNAS*, 98:15149–15154, 2001. ISSN 0027-8424.

A. Ramdas, N. Garcia, and M. Cuturi. On wasserstein two sample testing and related families of nonparametric tests. *Entropy*, 19, 09 2015.

C. Reddy and B. Vinzamuri. *A Survey of Partitional and Hierarchical Clustering Algorithms*, pages 87–110. 09 2018. ISBN 9781315373515. doi: 10.1201/9781315373515-4.

H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 0123694469.

V. Satopaa, J. Albrecht, D. Irwin, and B. Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems Workshops*, ICDCSW '11, page 166–171, USA, 2011. IEEE Computer Society. ISBN 9780769543864. doi: 10.1109/ICDCSW.2011.20. URL `https://doi.org/10.1109/ICDCSW.2011.20`.

J. Shaffer. Modified sequentially rejective multiple test procedures. *Journal of The American Statistical Association*, 81:826–831, 09 1986.

D. J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman Hall/CRC, 4 edition, 2007. ISBN 1584888148.

B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London, 1986.

G. Stiglic and P. Kokol. Stability of ranked gene lists in large microarray analysis studies. *Journal of biomedicine biotechnology*, 2010:616358, 06 2010.

N. Thanh, M. Ali, and L. Son. A novel clustering algorithm in a neutrosophic recommender system for medical diagnosis. *Cognitive Computation*, 9:1–19, 04 2017. doi: 10.1007/s12559-017-9462-8.

M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 61(3):611–622, aug 1999. ISSN 13697412. doi: 10.1111/1467-9868.00196. URL `http://doi.wiley.com/10.1111/1467-9868.00196`.

D. Titterington, A. Smith, and U. Makov. *Statistical Analysis of Finite Mixture Distributions*, volume 82. 01 1985. doi: 10.2307/2531224.

S. Trivedi, Z. Pardos, and N. Heffernan. The utility of clustering in prediction tasks. *ArXiv*, abs/1509.06163, 2015.

J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Algorithm selection on data streams. In S. Džeroski, P. Panov, D. Kocev, and L. Todorovski, editors, *Discovery Science*, pages 325–336. Springer International Publishing, 2014. ISBN 978-3-319-11812-3.

C. Villani. *Optimal transport : old and new*. Springer, 2009. ISBN 9783540710493.

S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1):37 – 52, 1987. ISSN 0169-7439. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.

X. Xu, M. Ester, H.-P. Kriegel, and J. Sander. A distribution-based clustering algorithm for mining in large spatial databases. pages 324–331, 01 1998.

Z. Yang and G. Villarini. Examining the capability of reanalyses in capturing the temporal clustering of heavy precipitation across europe. *Climate Dynamics*, 53, 03 2019. doi: 10.1007/s00382-019-04742-z.

C. T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, C-20(1):68–86, 1971.

J. Zhang, G. Wu, X. Hu, S. Li, and S. Hao. A parallel clustering algorithm with mpi – mkmeans. *Journal of Computers*, 8(1):10–17, 2013.

T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *SIGMOD Record*, 25(2):103–114, jun 1996. ISSN 01635808.

Q. Zhao, Weizhong; Ma, Huifang; He. Parallel K-Means Clustering Based on MapReduce. *Cloud Computing. Springer Berlin Heidelberg*, 2009:674–679, 2009. ISSN 03029743. doi: 10.1007/978-3-642-10665-1_71. URL `https://www.cs.ucsb.edu/{~}veronika/MAE/parallelkmeansmapreduce{_}zhao.pdf`.

X. Zhu. Semi-supervised learning literature survey. *Comput Sci, University of Wisconsin-Madison*, 2, 07 2008.