eman ta zabal zazu

Universidad del País Vasco   Euskal Herriko Unibertsitatea

Konputazio Zientziak eta Adimen Artifizialaren Saila

Departamento de Ciencias de la Computación e Inteligencia Artificial

K-means for massive data

by

Marco Vinicio Capó Rangel

Supervised by Aritz Pérez and Jose A. Lozano

Dissertation submitted to the Department of Computer Science and Artificial Intelligence of the University of the Basque Country (UPV/EHU) as partial fulfilment of the requirements for the PhD degree in Computer Science

Donostia - San Sebastián, April, 2019

Konputazio Zientziak eta Adimen Artifizialaren Saila

Departamento de Ciencias de la Computación e Inteligencia Artificial

K-means for massive data

by

Marco Vinicio Capó Rangel

Supervised by Aritz Pérez and Jose A. Lozano

Dissertation submitted to the Department of Computer Science and Artificial Intelligence of the University of the Basque Country (UPV/EHU) as partial fulfilment of the requirements for the PhD degree in Computer Science

Donostia - San Sebastián, April, 2019

This research was carried out at the Basque Center for Applied Mathematics (BCAM) within the Data Science- Machine Learning group and was supported by the Spanish Ministry of Economics and Competitiveness MINECO through the BCAM Severo Ochoa excellence accreditations SVP-2014-068574, SEV-2013- 0323 and SEV-2017-0718 and the ELKARTEK program. The support of MINECO and BERC Grants is acknowledged for the three months visit at the Laboratory of Artificial Intelligence and Decision Support- INESCTEC, Porto, Portugal.

Acknowledgments

First of all, I want to express my deep gratitude to my mentors, Aritz Pérez and Jose A. Lozano, for all their dedication, keen interest, timely advice, friendly environment and, especially, for their scholarly instruction in a scientific area that was unknown to me.

I also owe a debt of gratitude to Joao Gama for allowing me to complete a stay as a visitor in his research group, LIAAD-INESC TEC, at Porto University.

I want to express deepest gratitude to my parents, Ana and Delfín, my sisters, Carmen and Ana María, my brother, Johan, and my niece, Mariana, for all the memories, for being my main source of motivation and for always supporting me despite the critical situation back at home, Venezuela. Hopefully, one day we will be all together again...

I also want to dedicate this work and express my deepest appreciation and respect to all those Venezuelans that have died in the last few decades fighting for our freedom, our duty is to make sure that your deaths have not been in vain...

Finally, I want to thank my colleagues in BCAM, Diana Marcela Pérez, Mikel Agirre, Miguel Benítez, Nicole Cusimano, Mauricio Rincón, Lore Zumeta, Dae-Jin Lee, Isabella Marinelli, Daniel García, Havva Yoldas, Sandeep Kumar and Michael Barton, for all the nice lunches, trips, coffee breaks, advice, you made me feel at home during my stay in BCAM, I will be forever greatful to all of you.

Contents

List of Figures

List of Tables

0

Preface

0.1 Abstract

The K-means algorithm is undoubtedly one of the most popular clustering analysis techniques, due to its easiness in the implementation, straightforward parallelizability and competitive computational complexity, when compared to more sophisticated clustering alternatives. However, the progressive growth of the amount of data that needs to be analyzed, in a wide variety of scientific fields, represents a significant challenge for the K-means algorithm, since its time complexity is dominated by the number of distance computations, which is linear with respect to both the number of instances, n, and dimensionality of the problem, d. This fact hinders its scalability on such massive data sets. Another major drawback of the K-means algorithm corresponds to its high dependence on the initial conditions, which not only may affect the quality of the obtained solution, but may also have a major impact on its computational load, as for instance, a poor initialization could lead to an exponential running time in the worst case scenario.

In this dissertation, we tackle all these difficulties. Initially, we propose an approximation to the K-means problem, the Recursive Partition-based K-means algorithm (RPKM). This approach consists of recursively applying a weighted version of K-means algorithm over a sequence of spatial-based partitions of the data set, for which each cell of the partition is represented by the center of mass of the points that lie on it. From one iteration to the next, a more refined partition is constructed and the process is repeated using the optimal set of centroids, obtained at the previous iteration, as initialization. From a practical standpoint, such a process reduces the computational load of K-means algorithm as the number of representatives, at each iteration, is commonly much smaller than the number of instances of the data set. On the other hand, both phases of the algorithm are embarrasingly parallel. From a theoretical standpoint, and in spite of the selected partition strategy, one can guarantee the non-repetition of the clusterings generated at each RPKM iteration, which ultimately implies the reduction of the total amount of K-means

algorithm iterations, as well as leading, in most of the cases, to a monotone decrease of the overall error function. Afterwards, we present a RPKM-type approach, the Boundary Weighted K-means algorithm (BWKM). For this technique, the data set partition is based on an adaptative mesh, that adjusts the size of each grid cell to maximize the chances of each cell having only instances of the same cluster. The goal is to focus most of the computational resources on those regions where it is harder to determine the correct cluster assignment of the original instances (which is the main source of error for our approximation). For such a construction, it can be proved that if all the cells of a spatial partition are well assigned (have instances of the same cluster) at the end of a BWKM step, then the obtained clustering is actually a fixed point of the K-means algorithm over the entire data set. Furthermore, if, for a certain step of BWKM, this property can be verified at consecutive weighted Lloyds iterations, then the error of our approximation also decreases monotonically. From a practical stand point, BWKM was compared to the state-of-the-art: K-means++, Forgy K-means, Markov Chain Monte Carlo K-means and Minibatch K-means. The obtained results show that BWKM commonly converged to solutions, with a relative error of under 1% with respect to the considered methods, while using a much smaller amount of distance computations (up to 7 orders of magnitude lower).

Even when the computational cost of BWKM is linear with respect to the dimensionality, its error quality guarantees are mainly related to the diagonal length of the grid cells, meaning that, as we increase the dimensionality of the problem, it will be harder for BWKM to have such a competitive performance. Taking this into consideration, we developed a fully-parellelizable feature selection technique intended for the K-means algorithm, the Univariate K-means relevance for feature selection algorithm (KMR). This approach consists of applying any heuristic for the K-means problem over multiple subsets of dimensions (each of which is bounded by a predefined constant, $m \ll d$) and using the obtained clusterings to upper-bound the increase in the K-means error when deleting a given feature. We then select the features with the m largest error increases. Not only can each step of KMR be simply parallelized, but its computational cost is dominated by that of the selected heuristic (on m dimensions), which makes it a suitable dimensionality reduction alternative for BWKM on large data sets. Besides providing a theoretical bound for the obtained solution, via KMR, with respect the optimal K-means clustering, we analyze its performance in comparison to well-known feature selection and feature extraction techniques. Such an analysis shows KMR to consistently obtain results with lower K-means error than all the considered feature selection techniques: Laplacian scores, maximum variance and random selection, while also requiring similar or lower computational times than these approaches. On the other hand, KMR, when compared to feature extraction techniques, such as Random Projections, also shows a noticeable improvement in both error and computational time.

As a response to the high dependency of K-means algorithm to its initialization, we finally introduce a cheap and yet effective Split-Merge step that can be used to re-start the K-means algorithm after reaching a fixed point, Split-Merge K-means (SMKM). Under some settings, one can show that this approach reduces the error of the given fixed point without requiring any further iteration of the K-means algorithm. Moreover, experimental results show that this strategy is able to generate approximations with an associated error that is hard to reach for different multi-start methods, such as multi-start Forgy K-means, K-means++ and Hartigan K-means. In particular, SMKM consistently generated the local minima with the lowest K-means error, reducing, on average, over 1 and 2 orders of magnitude of relative error with respect to K-means++ and Hartigan K-means and Forgy K-means, respectively. We have observed that SMKM improves the previously commented methods in terms of the number of distances computed and the error of the obtained solutions.

0.2 Overview of the Dissertation

This document is divided into three separate parts: i) a quick introduction to the K-means clustering problem on massive data domains and the definition of the objectives, hypothesis and methodology of the dissertation, ii) a detailed analysis of the contributions and iii) conclusions and future work.

Part I introduces the main contributions of this PhD work to the machine learning field and is composed of three chapters: Chapter 1 contains a brief description of the K-means algorithm and the state-of-the-art in terms of the quality of the obtained solutions and of the reduction of computational requirements for the K-means problem. Then, Chapter 2 sums up the main contributions of this dissertation, their motivations, results and conclusions. These are as follows:

1. Design of two accurate approximations to the K-means problem for tall data applications.
2. Development of a dimensionality reduction technique (feature selection) designed for any heuristic that approximates the solution of the K-means problem.
3. Design of a cheap and effective re-start strategy for the K-means algorithm.

Part II is composed of the previously mentioned main contributions. This part is divided into Chapter 3, Chapter 4, Chapter 5, and Chapter 6, and they correspond to the contributions 1-3, respectively. Finally, in Part III some general conclusions and remarks on potential future work are drawn.

Part I

Introduction to the K-means Algorithm on Massive Data

1

A Brief Overview of the *K*-means Algorithm on Massive Data

Data generation has seen remarkable growth in the last few years. The emergence of a wide variety of areas, such as e-commerce, sensor networks, genomics, massive search engines and social media has transformed the way the storage, retrieval and analysis of such a massive amount of information is approached [1]. In a survey carried out by IBM, it is estimated that the amount of data generated surges to $2.5 \cdot 10^{18}$ bytes every day, while, by 2020, it is estimated that 1.7MB of data will be created every second for every person on earth [2, 3, 4]. In particular, it is reported that Google, Yahoo!, Microsoft, and other Internet-based companies have data that is measured in exabytes (10^{18} bytes) [1], while Facebook reports about 6 billion new photos every month and 72 hours of video are uploaded to YouTube every minute [5]. Such an overwhelming flow of information just keeps increasing every day, in fact, over the last two years alone, 90% of the data in the world was generated [3]. Data sets, as the ones described above, and the challenges involved when analyzing them, is often subsumed in the term *Massive Data* [1, 6].

Given such a tremendous amount of data, the development of efficient and effective tools has become necessary to analyze and reveal valuable knowledge that is hidden within the data [7]. In this sense, cluster analysis is a popular approach in data mining and has been widely used in several areas [7, 8].

1.1 Cluster Analysis

Cluster analysis is a multivariate method which aims to divide objects into a number of groups, called clusters, in such a way that intra-cluster similarity is high and the inter-cluster similarity is low. In this sense, clustering can be regarded as a form of classification that derives the labels of the objects only from the data set (cluster) and, for this reason, cluster analysis is sometimes referred to as unsupervised classification [9, 10, 11].

As described in [10], clustering has been mainly used for the following three purposes: i) to gain insight into data (underlying structure), ii) to identify the degree of similarity between objects (natural classification) and iii) to organize and summarize the data set into cluster prototypes. For this reason, this technique is a very important task in different areas, such as artificial intelligence, image processing, web mining, bioinformatics and pattern recognition [12, 9, 10, 13].

There exist different types of clustering algorithms, from which we can mainly distinguish: hierarchical versus unnested, and exclusive versus overlapping versus fuzzy/probabilistic [11].

The most common distinction between different clustering algorithms is whether they are hierarchical or unnested. Hierarchical clustering algorithms recursively find nested clusters (clusters are allowed to have subclusters) either in agglomerative mode (starting with each data point in its own cluster and combining the most similar pair of clusters recursively to form a cluster hierarchy) or in divisive mode (starting with all the data points in one cluster and recursively dividing each cluster into smaller clusters) [12, 9, 10]. On the other hand, unnested clustering algorithms divide the data set objects into a predefined number of clusters with no hierarchical structure [10].

Clustering algorithms can also be classified into exclusive versus overlapping versus fuzzy/probabilistic. While exclusive clusterings assign each object to a single cluster, overlapping clusterings allow objects to simultaneously belong to more than one cluster. On the other hand, in fuzzy clustering, every object belongs to every cluster with a membership weight that oscillates between 0 and 1. Similarly, probabilistic clustering techniques compute the probability with which each point belongs to each cluster. These last two approaches tend to be used to avoid the arbitrariness of assigning an object to only one cluster when it could be close to different groups [9, 11].

In particular, this dissertation is devoted to a very popular partitional clustering technique, which is in the intersection between unnested and exclusive clustering, called the K-means algorithm.

1.1.1 Partitional Clustering

The goal of partitional clustering is to divide a data set into a predefined number of clusters, in such a way that a given partitioning criterion is optimized. Even when there exists a wide variety of partitional clustering methods, the K-means algorithm [10, 14] stands out as one of the most popular, due to its ease of implementation, simplicity, efficiency and empirical success [15, 10]. This method, which attempts to find a user-specified number of clusters (K) which are represented by their centroids, has been named one of the top 10 algorithms in data mining by the organizers of the IEEE International Conference on Data Mining (ICDM) in 2008 [16, 17].

Other well-known partitional clustering techniques, such as K-medoids, PAM and CLARANS [18, 19, 20], are computationally more expensive than the K-means algorithm, commonly having time complexities supralinear with respect to the number of instances of the data set and/or the predefined number of clusters, which makes them inviable for massive data applications. More importantly, the fact that the K-means algorithm can be easily parallelized [21], as well as the existance of different speed-ups and cheap approximations, such as [22, 23, 24, 25, 26, 27, 28, 29, 30], makes the K-means algorithm a plausible alternative for this kind of applications.

In the following sections, we will describe the K-means problem/algorithm, analyze its different characteristics and discuss the most relevant variants/ approximations developed to improve its computational performance on large data sets.

1.1.1.1 *K*-means Problem

Given an unlabeled set of n data points (instances), $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq \mathbb{R}^d$, and a predefined number of clusters, K, the K**-means problem** is to determine a set of K centroids $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\} \subseteq \mathbb{R}^d$, so as to minimize the following error function:

$$E_X(C) = \sum_{\mathbf{x} \in D} \|\mathbf{x} - \mathbf{c}_{\mathbf{x}}\|^2, \text{ where } \mathbf{c}_{\mathbf{x}} = \arg\min_{\mathbf{c} \in C} \|\mathbf{x} - \mathbf{c}\|^2 \tag{1.1}$$

In other words, this is an unsupervised learning problem in which the goal is to minimize the within-cluster sum of squares, which indirectly increases the inter-cluster distances. The problem of minimizing the K-means error, which is reported to be first considered by Steinhaus in 1956 [31], is one of the earliest and most intensively studied formulations of the clustering problem, both because of its mathematical elegance and because it bears closely on the statistical estimation of mixture models of K point sources under spherically symmetric Gaussian noise [32, 33].

In particular, the K-means problem can be viewed as a combinatorial optimization problem, since it is equivalent to finding the partition of the n instances in K groups, whose associated set of centers of mass minimizes Eq.1.1. The number of possible partitions is a Stirling number of the second kind, $S(n, K) = \frac{1}{K!} \sum_{j=0}^{K} (-1)^{K-j} \binom{K}{j} j^n$ [34]. Since finding the globally optimal partition is known to be NP-hard for $K > 1$ [35], even for instances in the plane [36], and exhaustive search methods are not useful under this setting, iterative refinement based algorithms are commonly used to approximate the solution of the K-means problem [37, 38, 39, 40]. These algorithms iteratively relocate the data points between clusters until a locally optimal partition is attained. Among these methods, the most popular is the K**-means algorithm** [10, 14].

1.1.1.2 *K*-means Algorithm

The K-means algorithm is an iterative refinement method that consists of two stages: Initialization, in which we set the starting set of K centroids, and an iterative stage, called **Lloyd's algorithm** [14]. In the first step of Lloyd's algorithm (see Algorithm 1), each instance is assigned to its closest centroid (`Assignment step`), then the set of centroids is updated as the centers of mass of the instances assigned to the same centroid in the previous step (`Update step`). In Fig.1.1, we show a graphical example of the previously described procedure (from left to right). The yellow dots represent the original set of centroids, subsequently, we show the cluster assignment for each instance (represented in different colors) and the dotted lines represent the Voronoi frontiers generated by the set of centroids. It must be pointed out that the figure in the middle of Fig.1.1 is zoomed in, in order to highlight the obtained Voronoi tesselations. Finally, the red dots are the updated set of centroids. Commonly, the process described above is repeated until the set of centroids remains invariant, i.e., when the red and yellow dots completely overlap, in which case such a set of centroids is a local minima to the K-means problem [41].

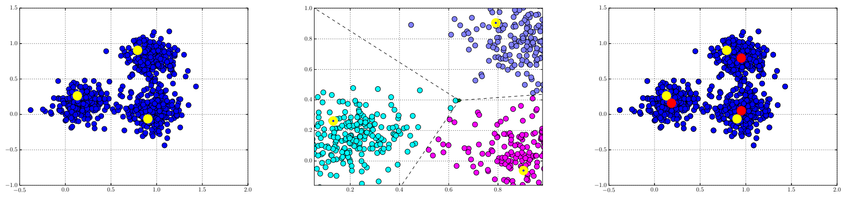

Fig. 1.1: Initial set of centroids (left), `Assignment step` (center), `Update step` (right).

Algorithm 1: Lloyd's algorithm

```
Input: Data set $X$, number of clusters $K$ and set of centroids
$C = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$.
Output: Local minima of Eq.1.1, $C^*$.
while Stopping Criterion do
    • Assignment step: Construct, for all $k \in \{1, \ldots, K\}$, the subsets
        $P_k = \{\mathbf{x} \in D : k = \underset{i \in \{1,\ldots,K\}}{\arg\min} \|\mathbf{x} - \mathbf{c}_i\|^2\}$
    • Update step: Take $\mathbf{c}_k = \overline{P_k}$ for all $k \in \{1, \ldots, K\}$.
end
Return $C^* = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$, $\mathcal{P} = \{P_1, \ldots, P_K\}$.
```

The computational time needed for `Assignment step` is $\mathcal{O}(n \cdot K \cdot d)$, which is due to the number of distances computed to determine the cluster affiliation of each instance, while updating the set of centroids requires $\mathcal{O}(n \cdot d)$ computations. As we previously mentioned, this process is commonly executed until the obtained set of centroids does not change in consecutive iterations.

However, a more general stopping criterion implies the computation of the error function (Eq.1.1): If the error does not change significantly with respect to the previous iteration, then the algorithm stops. In other words, if C and C' are the set of centroids obtained at consecutive Lloyd's iterations, then the algorithm stops when $|E_X(C) - E_X(C')| \leq \varepsilon$, for a fixed threshold $\varepsilon \ll 1$ [42].

To verify this condition $\mathcal{O}(n{\cdot}d)$ additional computations are needed. Hence, `Assignment step` is the most computationally demanding stage and, therefore, it is common practice to evaluate the number of distance computations to analyze the computational requirements of Lloyd's algorithm and similar heuristics [43, 26].

Among different advantages, such as the easiness of its implementation, Lloyd's algorithm reduces monotonically the K-means error, i.e., every step of the algorithm generates a more competitive solution to the K-means problem [10]. Furthermore, both steps of Lloyd's algorithm can be easily parallelized, which is a major key to meeting the scalability of the algorithm [17]. In the following sections, we comment on different features and variants to the K-means algorithm and their relation to massive data applications.

Importance of the initialization of Lloyd's algorithm

As widely reported in the literature, the performance of Lloyd's algorithm highly depends upon the initialization stage in terms of the quality of the solution obtained and the running time [44, 45, 46]. A poor initialization, for instance, could lead to an exponential running time in the worst case scenario: In [47], a simple construction in the plane, that leads to an exponential lower bound of $2^{\Omega(n)}$ running time, is presented.

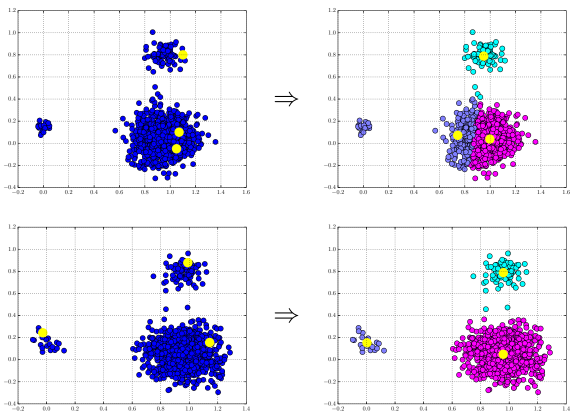

Fig. 1.2: An example of the K-means algorithm output for two different initializations. In the top figure, 11 Lloyd's algorithm iterations were executed, while, in the bottom figure, 2 iterations were needed.

In Fig.1.2, we show a simple example of the relevance of selecting a good initialization. In particular, we observe a 2D mixture of three Gaussians and the output of Lloyd's algorithm for two different sets of centroids (yellow dots). In the first case, we observe that two seeds are placed in the densest cluster, which hinders the convergence of Lloyd's algorithm to the expected solution and tends to increase the number of iterations needed to converge. In general, the selected seeding/initialization strategy should deal with different problems, such as outlier detection and cluster oversampling. A lot of research has been done on this topic: A detailed review of seeding strategies can be found in [45, 46].

The standard initialization procedure consists of performing several re-initializations via *Forgy's method* [48] and keeping the set of centroids with the smallest error [45, 46]. Forgy's technique defines the initial set of centroids as K instances selected uniformly at random from the dataset. The intuition behind this approach is that, by choosing the centroids uniformly at random, it is more likely to choose a point near an optimal cluster center, since such points tend to be where the highest density regions are located. Besides the fact that computing the error of each set of centroids is $\mathcal{O}(n \cdot K \cdot d)$ (due to the assignment step), the main disadvantage of this approach is that there is no guarantee that two, or more, of the selected seeds will not be near the center of the same cluster, especially when dealing with unbalanced clusters [45], see Fig.1.2, for instance.

More recently, different probabilistic seeding techniques have been developed and, due to their simplicity and strong theoretical guarantees, they have become quite popular. Among these, the most relevant is the *K-means++* algorithm proposed by Arthur and Vassilvitskii in [49]. K-means++ selects only the first centroid uniformly at random from the dataset. Each subsequent initial centroid is chosen with a probability proportional to the distance with respect to the previously selected set of centroids, see Fig.1.3.

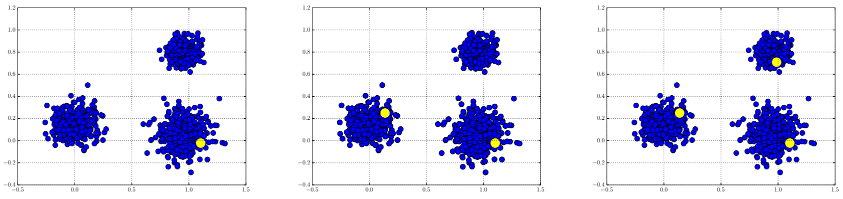

Fig. 1.3: K-means++ seeding example, for $K = 3$.

The key idea of this cluster initialization technique is to preserve the diversity of seeds while being robust to outliers. The K-means++ algorithm leads to a $\mathcal{O}(\log K)$ factor approximation[1] of the optimal error after the initialization [49]. The main drawbacks of this approach are its sequential nature, which hinders its parallelization, as well as the fact that it requires K full scans of the entire dataset, which leads to a complexity of $\mathcal{O}(n \cdot K \cdot d)$.

In order to alleviate such drawbacks, different variants of K-means++ have been proposed in the literature. In particular, in [50], a parallel K-means++ type algorithm is presented. This parallel variant achieves a constant factor approximation to the optimal solution after a logarithmic number of passes over the dataset. Furthermore, in [43], an approximation to K-means++ with a sublinear time complexity with respect to the number of data points, is proposed (AFKMC2). Such an approximation is obtained via a Markov Chain Monte Carlo sampling approximation of the K-means++ probability function. The proposed algorithm generates solutions of similar quality to those of K-means++, at a fraction of its cost [43], which makes it a competitive alternative for large data sets.

As we previously mentioned, since Lloyd's algorithm converges to a local minimum of the K-means problem, it is common practice to re-initialize Lloyd's algorithm multiple times, with one of the previously discussed seeding procedures, and keep the solution with the lowest error [45, 46]. Unfortunately, one of the biggest drawbacks of this approach is that no information about the data structure is transmitted from one re-start to another, which could be a key element to avoid converging to the same local minima several times, as well as for reducing the number of iterations executed when running Lloyd's algorithm. In the following section, we comment on a fairly popular heuristic that can be used to re-initialize Lloyd's algorithm taking into consideration information of the previously obtained local minimum.

Hartigan-Wong *K*-means algorithm

In this section, we comment on an alternative to the classic multi-start/ re-initialization heuristic for the K-means problem, the *Hartigan-Wong K-means algorithm* [51, 52]. Given an initialization, this method tries to determine a move of an instance, from its cluster to another, in a way that Eq.1.1 strictly decreases [53]. These single-point re-locations are repeated until convergence is reached.

In particular, for two clusters P_i, P_j and an instance, $\mathbf{x} \in P_i$, the change of the error function, Eq.1.1, obtained after moving $\mathbf{x}$ to P_j, is given by

$$\phi(\mathbf{x}, P_i, P_j) = \frac{|P_j|}{|P_j| + 1} \cdot \|\mathbf{x} - \mathbf{c}_j\|^2 - \frac{|P_i|}{|P_i| - 1} \cdot \|\mathbf{x} - \mathbf{c}_i\|^2 \qquad (1.2)$$

Hence, Hartigan's approach selects $\mathbf{x}, P_i, P_j$ that maximizes Eq.1.2 (Hartigan's heuristic) and iterates until $\phi(\mathbf{x}, P_i, P_j)$ is lower than zero [52]. It must

[1] Algorithm A is a λ factor approximation of the K-means problem, if $E_X(C') \leq \lambda \cdot \min_{C \subseteq \mathbb{R}^d, |C|=K} E_X(C)$, for any output C' of algorithm A.

be highlighted that, as Hartigan's fixed points are a subset of Lloyd's fixed points, this method has been suggested to replace Lloyd's algorithm [54]. Moreover, note that Hartigan's heuristic can also be used to exit the basin of attraction of a fixed point of Lloyd's algorithm, as it enforces the decrease of the error function, Eq.1.1 [53].

1.1.1.3 On improving the scalability of Lloyd's algorithm

Regardless of the initialization, a lot of research has been done in the recent years to make the K-means algorithm a more viable alternative for massive data applications. These articles can mainly be divided into two groups: i) those focused on reducing the number of instances and, ii) those referring to the use of dimensionality reduction techniques prior to running Lloyd's algorithm.

Reduction of the number of instances

As previously commented, one of the main drawbacks of Lloyd's algorithm is that, due to the number of distances that needs to be computed, its complexity is proportional to the size of the dataset, meaning that it may not scale well for massive data applications. One way of dealing with this is to apply the algorithm over a smaller set of points rather than over the entire dataset. Such smaller sets of points are commonly obtained in two different ways:

- *Via data set sampling*: In [55, 56, 57, 30], different statistical techniques are used with the same purpose of reducing the size of the dataset. Among these algorithms, we have the *Mini-batch K-means* proposed by Sculley in [30]. Mini-batch K-means is a very popular scalable variant of Lloyd's algorithm that proceeds as follows: Given an initial set of centroids obtained via Forgy's algorithm, at every iteration, a small fixed amount of samples is selected uniformly at random and assigned to their corresponding cluster. Afterwards, the cluster centroids are updated as the average of all samples ever assigned to them. This process continues until convergence. Empirical results, in a range of large web based applications, corroborate that a substantial saving of computational time can be obtained at the expense of some loss of cluster quality [30]. Moreover, very recently, in [58], an accelerated Mini-batch K-means algorithm, via the distance pruning approach of [26], was presented.
- *Via data set partition*: The reduction of the dataset can also be generated as sets of representatives induced by partitions of the dataset. In particular, there have been a number of recent papers that describe $(1+\varepsilon)$-factor approximation algorithms and/or (K,ε)-coresets[2] for the K-means

[2] A weighted set of points W is a (K,ε)-coreset if, for all set of centroids C, $|F_W(C) - E_X(C)| \leq \varepsilon \cdot E_X(C)$, where $F_W(C) = \sum_{\mathbf{y}\in W} w(\mathbf{y}) \cdot \|\mathbf{y} - \mathbf{c}_{\mathbf{y}}\|^2$ and $w(\mathbf{y})$ is the weight associated to a representative $\mathbf{y} \in W$.

problem [59, 60, 61]. Unfortunately, the constructions that propose these variants tend to be exponential in K and/or ε^{-1}, and so, they might not be viable in practice [49]. In the literature, other coreset constructions via sampling have been proposed as can be seen in [22, 23, 27, 29]. Moreover, in [62], a $(9+\varepsilon)$-approximated algorithm for the K-means problem is developed. Unfotunately, the cost of this process is still fairly prohibitive, $\mathcal{O}(n^3 \cdot \varepsilon^{-d})$.

In addition to the previous methods, the number of distances computed by Lloyd's algorithm can be further decreased by implementing pruning distace techniques, i.e., when it can be verified in advance that no cluster reassignment is possible for a certain instance in `Assignment step`. As presented in [24, 25, 26, 28], this can be done with the construction of different pairwise distance bounds between the set of points and centroids and additional information, such as the displacement of every centroid after a Lloyd's iteration. In particular, in [28], reductions of over 80% of the amount of distance computations are observed.

Dimensionality reduction

Dimensionality reduction consists of a set of algorithms that are used to embed the original features into a lower dimensional space of size $m \ll d$, which, ideally, contains the same information as the original dataset [63]. In particular, dimensionality reduction can be divided into two groups, feature selection and feature extraction:

- *Feature Selection:* In feature selection, actual dimensions of the dataset are selected to construct the lower dimensional space in which the dataset is embedded. This approach has been extensively used for the K-means problem [64, 65]. In particular, we can highlight procedures such as the Laplacian scores [66]. This method fundamentally uses a nearest neighbor graph to model the local geometric structure of the data and selects those features which are the smoothest on the graph [66]. Unfortunately, such a construction can be computationally costly as it requires, among other things, the computation of all pairwise distances between instances. Furthermore, no theoretical guarantees, related to the quality of the obtained clustering, are provided. Another popular unsupervised feature selection consists of selecting those features with the largest variance [65]. The intuition behind this approach is that low variance features may contain less clustering information and, therefore, tend to have a lower impact in the error function, Eq.1.1 [66].

  Other feature selection approaches consist of selecting features via random sampling, such as [67, 68] (uniformly), or [64], where a more elaborated Singular Value Decomposition (SVD)-based selection procedure is proposed. For this procedure, it can be proved that selecting

$m = \Theta(\frac{K \cdot \log \frac{K}{\varepsilon}}{\varepsilon^2})$ dimensions and running any λ-approximate K-means algorithm ($\lambda \geq 1$) leads to a $(1+(1+\varepsilon)\cdot\lambda)$-approximation of the optimal K-means error, with high probability [64]. Unfortunately, this technique may not be very practical, as it has a $\mathcal{O}(\min\{n \cdot d^2, n^2 \cdot d\})$ time cost.

- *Feature Extraction:* In feature extraction, the lower dimensional space is composed of new artificial features that are generated, for instance, via linear combinations of the original features. As can be seen in the literature, both SVD and PCA have been succesfully used to pre-procces the data prior to executing the K-means algorithm [64, 69, 24, 70]. In particular, among other results, it can be proved that, projecting the dataset into $m = \lceil \frac{K}{\varepsilon} \rceil$ dimensions via SVD, allows us to preserve the clustering error within a factor of $1+\varepsilon$ [69]. However, it must be pointed out that the time complexity of both procedures can be unpractical for massive data applications: $\mathcal{O}(\min\{n \cdot d^2, n^2 \cdot d\})$ for SVD and $\mathcal{O}(n \cdot d^2 + d^3)$ for PCA [71]. A fairly cheaper approach consists of applying Random Projections to the original dataset. This technique is just $\mathcal{O}(m \cdot d \cdot n)$ [72]. However, in this case, $m = \mathcal{O}(\frac{K}{\varepsilon^2})$ [69] dimensions are required to keep the $(1+\varepsilon)$-approximation factor.

Different variants of these methods exist, a detailed review and analysis of them can be found in [73, 69].

2

Objectives, hypothesis and methodology

2.1 Objectives

In this dissertation, we explore different alternatives for improving the scalability of the K-means algorithm on massive data applications. As we previously mentioned, the scalability of the K-means algorithm in such a scenario might be affected by, i) the large number of instances, ii) the dimensionality of the problem and, iii) the initialization phase of the K-means algorithm. In this document, we intend to provide computationally efficient and yet effective improvements in all these areas. In particular, we deal with the following targets:

- Development of accurate approximations to the K-means algorithm for *tall data*[1] applications, see Section 2.3.1, which briefly summarizes Chapter 3 and Chapter 4.
- Design of a dimensionality reduction technique for the K-means algorithm, see Section 2.3.2, which sums up in Chapter 5.
- Development of an efficient re-start strategy for the K-means algorithm, see Section 2.3.3 and Chapter 6.

The proposed methodology consists of analyzing both theoretically, by deducing multiple guarantees in terms of the quality of the obtained solutions and reduction of the computational requirements for the different proposals, and in practice, where we compare our work to the most popular approaches used to approximate the K-means problem. These results will be presented to the scientific community through publication in peer-reviewed journals, to validate the relevance and acceptance in the field.

[1] Data sets with an enormous number of instances and low number of dimensions.

2.2 Hypothesis

This thesis builds upon the idea that cluster analysis and, in particular, partitional clustering represents one of the most commonly used data analysis techniques in multiple scientific areas. These methods allow us to gain insight into data by discovering its underlying structure, as well as analyzing the degree of similarity between objects and organizing/summarizing the data set into cluster prototypes. Among them, the K-means algorithm emerges as one of the most popular.

On the other hand, the tremendous increase in the size of the data sets that are analyzed must be noted, as well as the fact that such an increase rate is expected to keep growing in the upcoming years. For this reason, the development of efficient and accurate approximations to the K-means problem on massive data is a topic of current relevance.

2.3 Methodology

2.3.1 Reducing the number of instances

The goal in this phase of the dissertation consists of developing competitive approximations to the K-means algorithm with an improved trade-off between the number of distances computed versus the quality of the obtained solution for the K-means problem. In this sense, in Chapter 3, we propose a general heuristic called the Recursive Partition Based K-means algorithm (**RPKM**). This approach consists of approximating the optimal solution to the K-means problem by applying Lloyd's algorithm over a reduced and weighted representation of the original data set, X. However, unlike the common coreset approach [22, 23, 27, 59, 60, 29, 61], RPKM does generate such an approximation over a sequence of spatial-based partitions of the data set, rather than over a predefined weighted set of points, which allows it to be more accurate as the number of iterations increases.

Definition 1 (Dataset partition induced by a spatial partition) *Given a data set X and a spatial partition $\mathcal{B}$ of its smallest bounding box, the partition of the dataset X induced by $\mathcal{B}$ is defined as $\mathcal{P} = \mathcal{B}(X)$, where $\mathcal{B}(X) = \{B(X)\}_{B \in \mathcal{B}}$ and $B(X) = \{\, \boldsymbol{x} \in X : \boldsymbol{x}$ lies on $B \in \mathcal{B}\}$*[2].

Applying the weighted version of K-means algorithm over the data set partition $\mathcal{P}$, consists of executing Lloyd's algorithm over the set of centers of mass (**representatives**) of $\mathcal{P}$, $\overline{P}$ for all $P \in \mathcal{P}$, considering their corresponding cardinality (**weight**), $|P|$, when updating the set of centroids. This means that we seek to minimize the weighted error function $E_{\mathcal{P}}(C) = \sum_{P \in \mathcal{P}} |P| \cdot \|\overline{P} - \mathbf{c}_{\overline{P}}\|^2$,

[2] From now on, we will refer to each $B \in \mathcal{B}$ as a **block** of the spatial partition $\mathcal{B}$.

where $\mathbf{c}_{\overline{P}} = \arg\min_{\mathbf{c}\in C} \|\overline{P} - \mathbf{c}\|$. Afterwards, the same process is repeated over a thinner partition $\mathcal{P}'$ of the dataset[3], using as initialization the set of centroids obtained for $\mathcal{P}$. In algorithm 2, we show a pseudocode of the described process.

Algorithm 2: RPKM algorithm pseudo-code

```
Input: Data set $X$ and number of clusters $K$.
Output: Set of centroids $C$.
Step 1: Construct an initial partition of $X$, $\mathcal{P}$, and define an initial set
  of $K$ centroids, $C$.
Step 2: $C$ = WeightedLloyd($\mathcal{P}, C, K$).
while not Stopping Criterion do
    Step 3: Construct a data set partition $\mathcal{P}'$, thinner than $\mathcal{P}$. Set
      $\mathcal{P} = \mathcal{P}'$.
    Step 4: $C$ = WeightedLloyd($\mathcal{P}, C, K$).
end
return $C$
```

In general, the RPKM algorithm can be divided into three tasks: The construction of an initial partition of the dataset and set of centroids (`Step 1`), the update of the corresponding set of centroids via weighted Lloyd's algorithm (`Step 2` and `Step 4`) and the construction of the sequence of thinner partitions (`Step 3`). The computational cost of `Step 2` and `Step 4` is $\mathcal{O}(|\mathcal{P}| \cdot K \cdot d)$, while the cost of `Step 2` and `Step 4` depends on the partition strategy used. In any case, independently of the partition strategy, the RPKM algorithm offers some interesting theoretical properties such as the no clustering repetition. That is, none of the obtained groupings of the n instances into K groups can be repeated at the current RPKM iteration or for any thinner partition than the current one (see Lemma 3). This is a useful property since it can be guaranteed that the algorithm discards many possible clusterings at each RPKM iteration using a much reduced set of points than the entire dataset. Furthermore, this fact enforces the decrease of the maximum number of Lloyd's iterations that we can have for a given partition (see Theorem 1). All these properties make such a heuristic a viable approach for the K-means problem on large data sets.

In the experimental section, we considered a data set partition based on grids, in a similar manner to [59], to which we referred to as the **grid based RPKM**. In particular, the initial spatial partition is defined by the grid obtained after dividing each side of the smallest bounding box of X by half, i.e., a grid with 2^d equally sized blocks. In the same fashion, at the i-th grid based RPKM iteration, the corresponding spatial partition is updated by dividing each of its blocks into 2^d new blocks, i.e., $\mathcal{P}$ can have up to $2^{i\cdot d}$ representa-

[3] A partition of the dataset $\mathcal{P}'$ is thinner than $\mathcal{P}$, if each subset of $\mathcal{P}$ can be written as the union of subsets of $\mathcal{P}'$.

tives, see Fig. 2.1. It can be shown that this approach produces a (K,ε)-coreset, with ε descending exponentially with respect to the number of iterations (see Theorem 9).

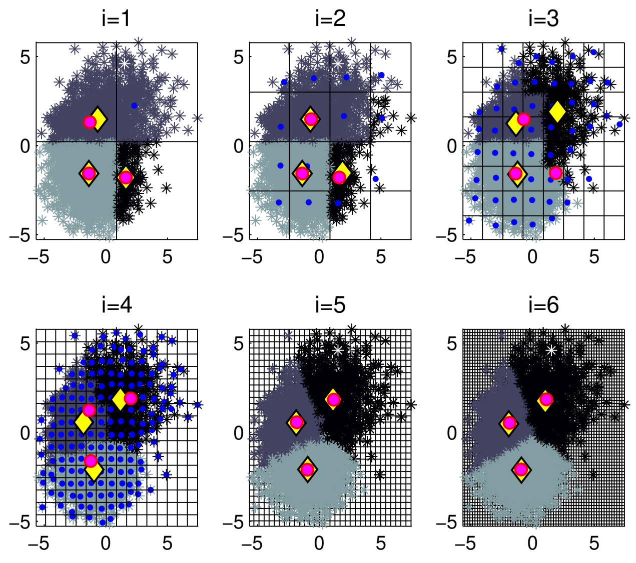

Fig. 2.1: 2D example for grid based RPKM.

In order to have a better idea of the trade-off quality of the approximations versus computational resources required, we compared the performance of the grid based RPKM to K-means++ algorithm (K-means++ seeding and Lloyd's algorithm), as well as to the Minibatch K-means algorithm (with different batch values), which are probably the most known heuristics for the K-means problem in terms of error guarantees and reduction of computational resources, respectively. The experiments show that the grid based RPKM algorithm tends to reduce several order of distance computations with respect to both alternatives (under some settings such a reduction could be of up to 4 and 6 orders of magnitude with respect to Minibatch K-means and K-means++, respectively), while consistently generating competitive approximations that are under 1% of relative error when compared to the best solution found by K-means++.

Even when the experimental results obtained via the grid based RPKM are fairly competitive with respect to well-known techniques, different difficulties may arise for such a partition strategy. In particular, it is clear that the grid based RPKM does not scale well on the dimensionality of the problem: Observe that, for a relatively low number of iterations, $i \simeq \log_2(n)/d$, and/or dimensionality $d \simeq \log_2(n)$, applying this RPKM version can be similar to applying Lloyd's algorithm over the entire dataset, i.e., no reduction of distance computations might be observed, as $|\mathcal{P}| \simeq n$. In fact, for the experimental section in Chapter 3, $d, i \leq 10$. On the other hand, the sequence of partitions of the grid based RPKM is induced by an equally sized spatial partition of the smallest bounding box containing X, meaning that a large amount of computational resources might be spent on regions whose misclassification does not add a significant error to our approximation. Moreover, the construction of every partition of the sequence has a $O(n \cdot d)$ cost, which is particularly expensive for massive data applications.

Taking these observations into consideration, in Chapter 4, we propose a RPKM-type approach called the Boundary Weighted K-means algorithm (**BW*K*M**). The idea behind this approach is to prioritize the use of resources on the cluster boundaries of our weighted approximation, which are constituted by those blocks that may not be well assigned, i.e., cells of the spatial partition that contain instances with different cluster affiliations (see Fig.2.2).

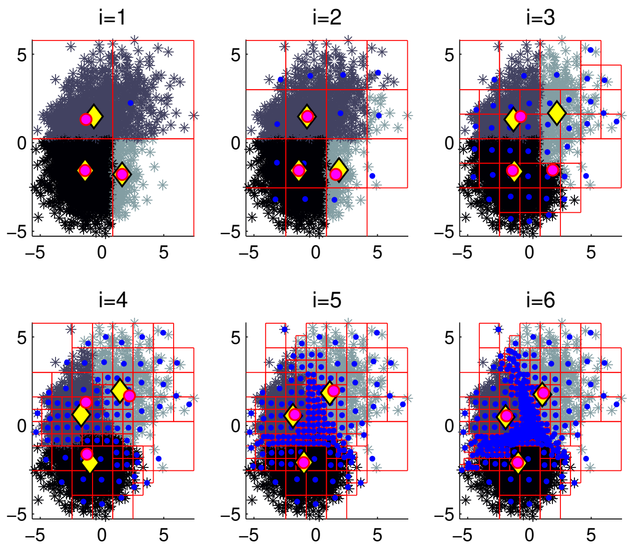

Fig. 2.2: 2D example of a spatial partition based on the cluster boundaries.

Definition 2 (Well assigned blocks) *Let C be a set of centroids and X be a given dataset. We say that a block B is well assigned, with respect to C and X, if every point $\boldsymbol{x} \in B(X)$ is assigned to the same centroid $c \in C$.*

It can be shown that our weighted error approximates the K-means error function more accurately, as we increase the number of well assigned blocks (see Theorem 3). Ultimately, if all the blocks of a spatial partition are well assigned at the end of a BWKM step, then the obtained clustering is actually a fixed point of the K-means algorithm (see Theorem 4). Note that the fixed point is generated using only a small number of representatives in comparison to the actual size of the data set. More importantly, if, for a certain step of BWKM, this property can be verified at consecutive weighted Lloyd's iterations, then the error of our approximation also decreases monotonically (see Theorem 10). In order to develop a partition strategy based on maximizing the possible number of well assigned blocks, we define the following measure for detecting them

Definition 3 *Given a set of K centroids, C, a set of points $X \subseteq \mathbb{R}^d$ and $P = B(X) \neq \emptyset$ the subset of points contained in a block B, we define the misassignment function for B, given C and X, as:*

$$\epsilon_{C,X}(B) = \max\{0, 2 \cdot l_B - \delta_P(C)\}, \tag{2.1}$$

where $\delta_P(C) = \min\limits_{\boldsymbol{c} \in C \setminus \boldsymbol{c}_{\overline{P}}} \|\overline{P} - \boldsymbol{c}\| - \|\overline{P} - \boldsymbol{c}_{\overline{P}}\|$ and l_B is the length of the diagonal of B. In the case $P = B(X) = \emptyset$, we set $\epsilon_{C,X}(B) = 0$.

It can be proved that, if a certain non-empty block B satisfies $\epsilon_{C,X}(B) = 0$, then all the instances in B share the same cluster affiliation. Moreover, their affiliation corresponds to the one of its centers of mass, i.e., B is well assigned with respect to C (see Theorem 2). Hence, the partition strategy is to divide a subset of blocks for which $\epsilon_{C,X}(B) > 0$, where C is the set of centroids obtained after executing the weighted version of Lloyd's algorithm (`Step 2`, `Step 4` of algorithm 2). Once a subset of blocks are selected, they are only divided on the dimension associated to its longest side, which minimizes the diagonal of the newly created blocks. Consequently, the error bound of the BWKM approximation is reduced (see Theorem 3) and the well assignment condition of the generated blocks (Definition 2.3.1) is more likely to be satisfied in the next iterations of the algorithm. In other words, such a contruction tends to maximize the number of well assigned blocks and implies a linear growth in the number of representatives, with respect to the dimensionality of the problem.

One of the major advantages of the proposed criterion to detect well assigned blocks is its low computational cost: it only uses information generated by the weighted K-means algorithm -the distances between the center of mass of each block and the set of centroids- and a feature of the corresponding spatial partition -the diagonal length of each block-. Therefore, computing $\epsilon_{C,X}(B)$, for all $B \in \mathcal{B}$, can be done in just $\mathcal{O}(|\mathcal{P}| \cdot K)$ time. In particular, as can be seen in Appendix .2.2, it is common to observe that, when the dimensionality of the problem is not large, after a few BWKM iterations, most of the blocks obtained generated are well assigned.

In Chapter 4, we also propose a simple heuristic for constructing the initial partition of the data set with a predefined number of blocks, which are mostly placed on the cluster boundaries. In particular, starting with the smallest bounding box of the data set, X, the proposed procedure iteratively divides subsets of blocks of the spatial partition with high probabilities of not being well assigned. To evaluate the well assignment criterion, multiple sets of centroids are selected via a weighted K-means++ run over the current set of representatives. The goal in this step is to bound the size of the initial partition of the data set and to determine a competitive initialization for BWKM.

In addition to all the theoretical guarantees that motivate and justify our algorithm, we empirically compare the performance of BWKM to K-means++, Minibatch K-means (with different batch values) and we additionally consider the also popular Forgy K-means and a scalable variant of K-means++, the Markov Chain Monte Carlo K-means++ (AFKMC2). The obtained results show that BWKM massively reduces the number of distances computed by the previously mentioned methods, while in 18 out of 35 configurations of the experimental setting, BWKM actually generated the most competitive solutions among all of them. Additionally, in 206 out of 210 cases, BWKM has converged to solutions with a relative error of under 1% with respect to the best considered method, while using a much smaller amount of distance computations (up to 7 orders of magnitude lower). Furthermore, in Appendix .2.2, we observe that the grid based RPKM failed to converge, within a time limit of 24 hours, for most of the data sets considered, with $d > 10$. This fact shows the improvement on the scalability of BWKM, in comparison to the grid based RPKM.

2.3.2 Dimensionality Reduction

Even when the results presented in Chapter 4 are very competitive and show the advantages of using BWKM for data sets with a large number of instances, the quality guarantees of BWKM are still dependant on the diagonal length of the cells in the spatial partition. Since the partition strategy introduced in BWKM divides the selected blocks along a single dimension, it will take more BWKM iterations to generate partitions with mostly well assigned blocks, as the dimensionality of the data set increases (see Appendix .2.2). This problem not only affects the quality of the approximation, but also increases the number of distance computations, since the number of representatives is also higher.

In this regard, there is a wide variety of dimensionality reduction techniques that are known for preserving the quality of the clusterings obtained via K-means algorithm, such as Laplacian Scores, Random Projections, Principal Component Analysis and Singular Value Decomposition [64, 73, 69, 24, 66, 70]. These techniques are not necessarily linear with respect to the dimensionality an/or number of instances of the data set and their parallelization is not straightforward, hence they are not viable for massive data applications.

In Chapter 5, we developed a low computational complexity and fully-parellelizable feature selection technique intended for the K-means algorithm, the univariate K-means relevance for feature selection algorithm (KMR). This approach consists of three steps: First, given the number of features that we wish to select, m, the set of dimensions $\{1, \dots, d\}$ is divided into the smallest number of chunks possible, $\{D_1, \dots, D_t\}$, so that each of them has a cardinality lower than a predefined dimensionality limit, m. The partition of the dataset is then given by $\{X_{D_1}, \dots, X_{D_t}\}$, where X_{D_i} stands for extracting the columns in D_i of X. Afterwards, a λ-approximate K-means algorithm, algorithm $\mathcal{A}$[4], is applied in parallel over each chunk X_{D_i}. The obtained clustering is used to score the importance of each feature in X_{D_i} by approximating the error increment that would happen if a given feature is eliminated. Finally, after submitting the score obtained for each variable, to a central server, the m variables that seem to affect the clustering quality the most are selected, and algorithm $\mathcal{A}$ is applied over them to obtain our approximation.

Evaluating the proposed variable importance score only has a $\mathcal{O}(K \cdot m)$ cost, on each machine. Therefore, the computational cost of our proposal is dominated by the cost of algorithm $\mathcal{A}$, which makes it suitable for large data sets

Besides the fact that the previosuly described process is fully parallelizable, it can be proved that KMR leads to a $\mathcal{O}(1+\varepsilon)$-approximation of the optimal solution of the K-means problem, where ε is the maximum normalized error increase after discarding the variables on one of the machines, see Theorem 6. Additionally, we compare its performance to well-known feature selection and feature extraction techniques. Such an analysis shows KMR to consistently obtain results with lower K-means error than all the considered feature selection techniques: Laplacian scores, maximum variance and random selection, while also requiring similar or lower computational times than these approaches. Even more interesting, KMR, when compared to feature extraction techniques, such as Random Projections, also shows a noticeable improvement in both error and computational time.

2.3.3 Re-start strategy for the *K*-means algorithm

As is well documented in the literature, one factor that must be considered when reducing the computational requirements of the K-means algorithm is the quality of its initialization [49, 44, 45, 46, 47]. A poor initialization not only may affect the quality of the obtained approximation, but may also increase tremendously the number of iterations required to convergence [74, 47]. In order to ease the convergence to competitive approximations of the K-means problem, the common practice is to re-start Lloyd's algorithm multiple times, with different seeds, and to keep the solution with the lowest error [45, 46]. Taking this into consideration, in Chapter 6, we propose a simple re-start process called Split-Merge K-means algorithm, or just SMKM, which is an alternative to the typical multi-start K-means algorithm. Given a local minima of the K-means problem, SMKM detects those regions that may have

[4] The results are presented for any heuristic used to approximate the solution of the K-means problem. Algorithm $\mathcal{A}$ could be for instance, K-means algorithm, BWKM, RPKM or any coreset-type approach in general.

an excessive (over-represented) and insufficient (under-represented) amount of centroids. SMKM follows a two-step re-initialization process: i) the split step divides the cluster from where the largest error reduction can be reached and generates two new centroids. ii) the merge step condenses the pair of clusters (and their corresponding centroids), whose fusion produces the smallest increment in the error. The split step reduces the K-means error of our approximation, while the merge step increases it, hence if the difference between both phases is negative, then the quality of the previous K-means local minima is already improved. In Theorem 8, we additionally provide a condition that can be easily verified using information of the current local minima to predict the error descent after such a re-initialization process.

In general, SMKM utilizes the information of previous Lloyd's algorithm runs to facilitate the convergence to a more competitive local minima, as can be seen in the experimental section of Chapter 6. The proposed procedure has a computational complexity of $\mathcal{O}(\max\{n, K^2\} \cdot d)$, which is cheaper than a single Lloyd's algorithm iteration.

Experimentally, SMKM was compared in both quality of the achieved local minima and number of distances computed with respect to competitive multi-start approaches (K-means++ and Forgy K-means, with a fixed number of restarts, and Hartigan K-means) on a wide variety of real-life data sets. In terms of the quality of the approximation, SMKM consistently generated the local minima with the lowest K-means error, reducing, on average, over 1 and 2 orders of magnitude of relative error with respect to K-means++ and Hartigan K-means and Forgy K-means, respectively. The quality of the solutions obtained by SMKM tends to be much lower than the previously commented methods and, in terms of computational resources, SMKM also required a much lower number of distance computations (about an order of magnitude less) to convergence.

Part II

Our Contributions

3

An efficient approximation to the K-means clustering for massive data

Due to the progressive growth of the amount of data available in a wide variety of scientific fields, it has become more difficult to manipulate and analyze such information. Even though data sets have grown in size, the K-means algorithm remains as one of the most popular clustering methods [15, 10, 17]. In order to deal with this problem, in this chapter, we propose a heuristic for the K-means problem based on a recursive data partitioning process that reduces the number of distance calculations and data scans, while generating competitive approximations. The algorithm considers a sequence of thinner partitions of the data set and applies a weighted version of Lloyd's agorithm over the centers of mass of each subset of the partition (set of representatives). Among other benefits, this approach reduces the number of distance computations over the whole data set, which is the most computationally demanding stage of the K-means algorithm.

This chapter is organized as follows: In Section 3.1, we describe the idea behind our algorithm and introduce notation that we use, in Section 3.2, to state some theoretical guarantees of our approach. The proofs of such statements can be found in Appendix .1. In Section 3.3, we present a set of experiments in which we analyze the effect of different factors, such as the size of the data set and the dimension of the instances over the performance of our algorithm. Additionally, we compare these results with the ones obtained by the K-means++ and the minibatch K-means methods.

3.1 Recursive partition based K-means

We propose a novel, iterative approximation strategy for the K-means problem that is based on a sequence of recursive partitions of the data set, being each partition thinner than the previous one. We call this approach **recursive partition based K-means** (**RPKM**). The idea behind the algorithm is to approximate the K-means problem for the full data set by recursively applying a weighted version of the K-means algorithm over a growing, yet small, number of subsets of the data set.

In the first step of the RPKM, the data set is partitioned into a number of subsets each of which is characterized by a representative (center of mass) and its corresponding weight (cardinality). Finally, a weighted version of Lloyd's algorithm (see Section 3.1.2 for further details) is applied over the set of representatives. From one iteration to the next, a more refined partition is constructed and the process is repeated using the optimal set of centroids obtained at the previous iteration as initialization. This iterative procedure is repeated until a certain stopping criterion is met.

In the next section, we describe in detail the recursive partition process and characterize some of its properties. The notation introduced in this section will be used later for a formal description of the RPKM algorithm.

3.1.1 Recursive Partitions

As previously mentioned, the recursive partition process is the first stage of the RPKM algorithm. This step consists of generating a thinner partition than the previous one at each iteration.

From now on we will use the following definition of partition of a data set.

Definition 4 (Partition of a data set X) *$\mathcal{P} = \{S_1, \ldots, S_t\}$ is a partition of the data set X if $\bigcup\limits_{i=1}^{t} S_i = X$ and if the subsets of $\mathcal{P}$ are (pairwise) disjoint and nonempty. Moreover, given a subset $S \in \mathcal{P}$, we define its weight as its cardinality, $|S|$, and its representative as its center of mass, $\overline{S} = \frac{\sum_{\mathbf{x} \in S} \mathbf{x}}{|S|}$.*

The partition of the data set allows us to describe it with a reduced number of representatives, which ultimately implies the reduction of the number of distance computations with respect to the Lloyd's algorithm for the full data set. In the RPKM algorithm, we will use the set of representatives and weights, rather than the partition itself.

Definition 5 (Partition thinner than $\mathcal{P}$) *Given two partitions of the data set X, $\mathcal{P}$ and $\mathcal{P}^{'}$, we say that $\mathcal{P}^{'}$ is a partition thinner than $\mathcal{P}$ ($\mathcal{P} \succ \mathcal{P}^{'}$) if, for all $S \in \mathcal{P}$, $S = \bigcup\limits_{R \in \mathcal{P}^{'}[S]} R$, where $\mathcal{P}^{'}[S] = \{R \in \mathcal{P}^{'} : R \subseteq S\}$.*

In other words, $\mathcal{P}^{'}$ is a partition thinner than $\mathcal{P}$ if every subset of $\mathcal{P}$ can be written as the union of subsets of $\mathcal{P}^{'}$.

The partition process generates a **sequence of thinner partitions** $\mathcal{P}_1, \ldots, \mathcal{P}_m$, such that $\mathcal{P}_{i-1} \succ \mathcal{P}_i$ for all $i \in \{2, \ldots, m\}$. Evidently, the number of representatives tends to increase as we generate a thinner partition. In the extreme case $\mathcal{P}_m = \{\{\mathbf{x}\} : \mathbf{x} \in X\}$, however, in practice, in order to reduce the computational complexity of the RPKM, we control the number of representatives so that $|\mathcal{P}_m| \ll n$.

Note that the weight and the representative of $S \in \mathcal{P}_i$ can be easily computed from $\mathcal{P}_{i+1}[S]$ as follows: $|S| = \sum\limits_{R \in \mathcal{P}_{i+1}[S]} |R|$, $\overline{S} = \frac{\sum\limits_{R \in \mathcal{P}_{i+1}[S]} |R| \cdot \overline{R}}{|S|}$. As we noted before, we are interested in the computation of the set of representatives and weights, thus, we will use $\mathcal{P}_m$ to generate the set of representatives and weights of the entire sequence of thinner partitions backward, from $\mathcal{P}_{m-1}$ to $\mathcal{P}_1$. Hence, the construction of $\mathcal{P}_i$ has a $\mathcal{O}(|\mathcal{P}_{i+1}| \cdot d)$ time cost for $i < m$. Moreover, if the assignment criterion of each instance of X into its corresponding subset in $\mathcal{P}_m$ is of order $\mathcal{O}(d)$, as it is in the case of the grid based RPKM (see Section 3.1.4), then the construction of $\mathcal{P}_m$ is $\mathcal{O}(n \cdot d)$ and, therefore, the cost of the entire partition process is $\mathcal{O}(d \cdot (n + \sum_{i=2}^{m} |\mathcal{P}_i|))$.

3.1.2 Weighted K-means problem

In this section, we introduce a generalization of the K-means problem defined over a set of weighted points, e.g. the set of representatives and their respective weights associated to a partition. As a first step, we define a clustering for a partition.

Definition 6 (Clustering for a partition $\mathcal{P}$) *We say that a partition of the data set X, $\mathcal{G}$, is a clustering of the data set for a partition $\mathcal{P}$, when $|\mathcal{G}| = K$ and $\mathcal{G} \succ \mathcal{P}$.*

In other words, a cluster for a partition is a set of K subsets of points of X, such that all the points of any $S \in \mathcal{P}$ are assigned to the same cluster.

We call $\mathcal{G} = \{G_1, .., G_K\}$ a **clustering induced by a set of centroids** $C = \{\mathbf{c}_1, ..., \mathbf{c}_k\}$, when $G_k = \bigcup\limits_{S \in M_k} S$ for $k = 1, ..., K$, where $M_k = \{S \in \mathcal{P} : k = \underset{j=1,...,K}{\arg\min} \|\overline{S} - \mathbf{c}_j\|^2\}$. In other words, a clustering induced by a set of centroids is a partition of the data set in which all the data points that have the same closest centroid from C are grouped in the same cluster. We denote that the clustering $\mathcal{G}$ is induced by a set of centroid C by $\mathcal{G} \leftarrow C$. Similarly, we call $C = \{\mathbf{c}_1, ..., \mathbf{c}_k\}$ a **centroids set induced by a clustering** $\mathcal{G} = \{G_1, ..., G_K\}$, when $c_i = \overline{G}_i$ for $i = 1, ..., K$. In other words, the set of centroids induced by a clustering $\mathcal{G}$ is the set of centers of mass associated to each cluster in $\mathcal{G}$. We denote that the set of centroids C is induced by a clustering $\mathcal{G}$ by $C \leftarrow \mathcal{G}$.

Given a partition of the data set X, $\mathcal{P}$, the **weighted K-means problem** seeks to determine a set of K centroids $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$ in $\mathbb{R}^d$, so as to minimize the **centroid error** associated to a partition $\mathcal{P}$, which is defined as follows:

$$E_{\mathcal{P}}(C) = \sum_{S \in \mathcal{P}} |S| \cdot \min_{k=1,\ldots,K} \|\overline{S} - \mathbf{c}_k\|^2 = \sum_{k=1}^{K} \sum_{S \in \mathcal{P} : S \subseteq G_k} |S| \cdot ||\overline{S} - \boldsymbol{c}_k||^2 \quad (3.1)$$

where the clustering $\mathcal{G}$ is induced by the set of centroids C. This error measures the weighted error between the representative of each subset with respect to its closest centroid.

Algorithm 3: Weighted Lloyd (WL)

```
Input: Set of representatives $\{\overline{S}\}_{S\in\mathcal{P}}$ and weights $\{|S|\}_{S\in\mathcal{P}}$, for
       the partition $\mathcal{P}$. Number of clusters $K$ and initial set of
       centroids $C_0$.
Output: Set of centroids $C_r$ and corresponding clustering pattern $\mathcal{G}_r$.
Step 0 (Initial Assignment):
    $\mathcal{G}_0 \longleftarrow C_0;\ \ r = 0$.
while not StoppingCriterion do
    $r = r + 1$.
    Step 1 (Update Step):                          Clustering error
        $C_r \longleftarrow \mathcal{G}_{r-1}$                          $E_{\mathcal{P}}(\mathcal{G}_{r-1})$ (Eq. 3.3)
    Step 2 (Assignment Step):                      Centroid error
        $\mathcal{G}_r \longleftarrow C_r$                              $E_{\mathcal{P}}(C_r)$ (Eq. 3.1)
end
Return $C_r$ and $\mathcal{G}_r$.
```

In order to approximate the solution of the weighted K-means problem, we use a generalization of Lloyd's algorithm called the **weighted Lloyd's algorithm** (**WL**, see Algorithm 3). In the assignment stage of WL (`Step 0` and `Step 2`), the clustering $\mathcal{G}_r$ is induced by the set of centroids C_r. Furthermore, in the update step (`Step 1`), the set of centroids C_r is induced by the clustering $\mathcal{G}_{r-1}$. Similarly to Lloyd's algorithm, an execution of WL with l iterations produces a sequence of sets of centroids and clusterings that can be represented as follows:

$$C_0 \to \mathcal{G}_0 \to, C_1 \to \mathcal{G}_1 \to ... \to C_{l-1} \to \mathcal{G}_{l-1} \to C_l \to \mathcal{G}_l$$

where C_0 is the set of centroids used for initialization and C_l is the returned set of centroids.

The assignment step requires $\mathcal{O}(|\mathcal{P}|\cdot K\cdot d)$ computations, since we just need to compute the distance between the set of centroids and the set of representatives, while for the update step of the set of centroids and the computation of its error (centroid error) $\mathcal{O}(|\mathcal{P}|\cdot d)$ computations are needed. Remember that the most common stopping criterion of the K-means algorithm consists of verifying that the difference of the set of centroids error, in two consecutive iterations, is smaller than a certain threshold. Moreover, observe that the set of weights is only used when updating the set of centroids. Since the number of representatives usually satisfies $|\mathcal{P}| \ll n$, when dealing with massive data problems, we can have a relevant reduction in the complexity with respect to the K-means algorithm for the full data set.

3.1.3 RPKM Algorithm

In this section, we formally present the RPKM algorithm. This algorithm mainly consists of constructing a sequence of thinner partitions $\mathcal{P}_1, \ldots, \mathcal{P}_m$ and then applying WL over the set of representatives of each partition in the sequence. From one iteration to the next, the preceding found solution is used as initialization. As we will show later, this initialization assignment allows us to reduce the maximum number of WL iterations at every RPKM run. The pseudo-code of the RPKM algorithm can be seen in Algorithm 4.

Algorithm 4: RPKM Algorithm

```
Input: Dataset $X$, number of clusters $K$, maximum number of
       iterations $m$.
Output: Set of centroids approximation $C_i$.
Step 1 Compute the set of weights and representatives of the
       sequence of thinner partitions, $\mathcal{P}_1, \ldots, \mathcal{P}_m$, backwards.
       Set $i = 1$.
while not Stopping Criterion do
    Step 2 Update the centroid's set approximation, $C_i = \{\mathbf{c}^i_j\}_{j=1}^K$:
           $C_i = WL(\{\overline{S}\}_{S\in\mathcal{P}_i}, \{|S|\}_{S\in\mathcal{P}_i}, K, C_{i-1})$
    $i = i + 1$
end
Return $C_i$
```

In the first step of the RPKM algorithm, we obtain backwards (see Section 3.1.1) the set of representatives and weights associated to the sequence of thinner partitions $\mathcal{P}_1, \ldots, \mathcal{P}_m$. Observe that we are assuming, without loss of generality, that $|\mathcal{P}_1| > K$. In `Step 2`, we update the centroids approximation by applying WL using the representatives and weights set determined at the previous step, we take as initialization the approximation for the previous iteration, C_{i-1}. In the first RPKM iteration, we set C_{i-1} as K random representatives of $\{\overline{S}\}_{S\in\mathcal{P}_i}$ (Forgy's type initialization). The algorithm iterates until $i = m$ or until a stopping criterion is met. We recommend the computation of a centroid's set displacement measure as stopping criterion: $\delta(C_{i-1}, C_i) = \max\limits_{j=1,\ldots,K} \|\mathbf{c}^i_j - \mathbf{c}^{i-1}_j\|^2$. If this value is smaller than a certain threshold, the algorithm stops, since the approximation did not improve significantly after the last RPKM iteration.

In relation to the complexity of Algorithm 4, we know, from Section 3.1.1, that `Step 1` has an $\mathcal{O}(d \cdot (n + \sum_{i=2}^m |\mathcal{P}_i|))$ time cost. Moreover, at the i-th RPKM iteration, the time required for WL (`Step 2`) is $\mathcal{O}(|\mathcal{P}_i| \cdot K \cdot d)$. Finally, the recommended stopping criterion just performs $O(K \cdot d)$ computations. Hence, the overall complexity of the RPKM algorithm, in the worst case, is $O(\max\{d \cdot (n + \sum_{i=2}^m |\mathcal{P}_i|), |\mathcal{P}_m|\} \cdot K \cdot d\})$.

3.1.4 RPKM implementation based on grid partitions

Later on, we will verify that the theoretical advantages of the RPKM algorithm hold independently of the geometry that we use to generate the partition. Nonetheless, one way to guarantee the generation of a sequence of thinner partitions of the data set consists of partitioning the space in a recursive manner. To do so, one possibility is to use a generalization of the *quadtrees* for higher dimensions [75]. The quadtree data structure has been used in several areas such as dimension reduction problems, spatial indexing, storing sparse data, computer graphics: computational fluid dynamics, etc [76].

The d-dimensional generalization of a quadtree is a tree data structure that generates partitions of the space into d-dimensional hypercubes and, subsequently, of the data set in subsets in the following way: each internal node of the tree is exactly divided in 2^d children, i.e., each subset of the i-th partition is divided into, at most, 2^d sets of the $(i+1)$-th partition (see Fig.3.1). This property allows us to generate, in a simple manner, a sequence of thinner partitions at each iteration satisfying $|\mathcal{P}_i| \leq \min\{n, 2^{i \cdot d}\}$.

In the following example, we consider a set of 10000 points generated from a mixture of three $2D$ Gaussians. We compute, as a reference, the solution for $K = 3$ using the K-means++ method. After ten runs, we obtained, on average, an error of 11393.45 with a standard deviation of 4.69. The number of distance computations was, on average, 642000. In Fig.3.1, we show the evolution of the RPKM algorithm, for $m = 6$, the red circles represent the initial set of centroids, the yellow diamonds the final set of centroids and the blue points the set of representatives for each iteration.

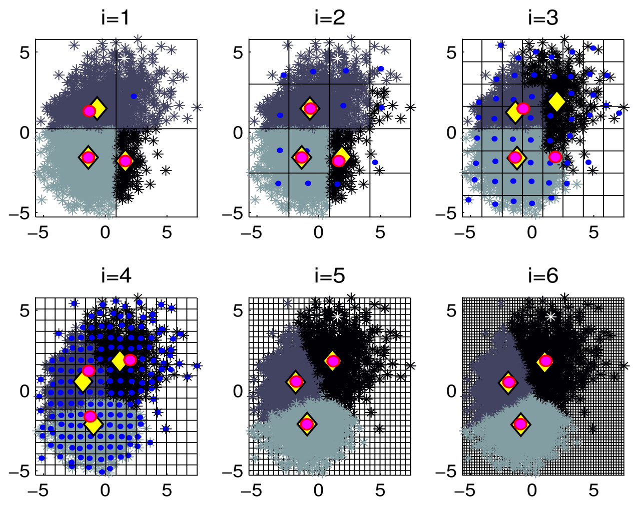

Fig. 3.1: Best clustering obtained at each RPKM iteration.

i	**Dis Com**	$\lvert\mathcal{P}_i\rvert$	$E(C_i)$	i	**Dis Com**	$\lvert\mathcal{P}_i\rvert$	$E(C_i)$
1	24	4	14050.06	4	5697	173	11424.24
2	114	15	14024.38	5	10449	528	11408.40
3	1545	53	12350.41	6	26781	1361	11389.54

Table 3.1: RPKM iteration results.

From Table 3.1, we can observe that, even at the fourth grid based RPKM iteration, which in this case implies 173 representatives (1.73% of the data set), we have a fairly good approximation of the average best solution found by the K-means++ algorithm for the entire 10000 points. On average, the RPKM algorithm computed 0.887% and 4.17% of the total number of distance computations of the K-means++ algorithm, at the fourth and final iteration respectively.

As we consider higher iterations of the RPKM, the associated cost function converges to the best solution obtained by the the K-means++. The intuition behind this method is to transform a random initial set of centroids into a competitive approximation by using small groups of representatives, instead of the entire data set. Next, we consider higher values of i to refine such an approximation.

3.2 Theoretical analysis of the RPKM algorithm

In this section, we perform a theoretical analysis of RPKM. In Section 3.2.1, we analyze the evolution of the clustering error at different steps of RPKM. Then, in Section 3.2.2, we investigate the repetitions of the clusterings obtained during the execution of RPKM and we bound the maximum number of WL iterations for different steps of RPKM.

Before starting with the theoretical results, we summarize an execution of RPKM with m steps given in terms of sequences of centroids and clusterings:

$$\begin{aligned}
\mathcal{P}_1 :&\quad C_0^1 \to \mathcal{G}_0^1 \to C_1^1 \to \mathcal{G}_1^1 \to ... \to \mathcal{G}_{l_1-1}^1 \to C_{l_1}^1 \to \mathcal{G}_{l_1}^1 \\
\mathcal{P}_2 :&\quad C_0^2 \to \mathcal{G}_0^2 \to C_1^2 \to \mathcal{G}_1^2 \to ... \to \mathcal{G}_{l_2-1}^2 \to C_{l_2}^2 \to \mathcal{G}_{l_2}^2 \\
...& \\
\mathcal{P}_i :&\quad C_0^i \to \mathcal{G}_0^i \to C_1^i \to \mathcal{G}_1^i \to ... \to \mathcal{G}_{l_i-1}^i \to C_{l_i}^i \to \mathcal{G}_{l_i}^i \\
...& \\
\mathcal{P}_m :&\quad C_0^m \to \mathcal{G}_0^m \to C_1^m \to \mathcal{G}_1^m \to ... \to \mathcal{G}_{l_m-1}^m \to C_{l_m}^m \to \mathcal{G}_{l_m}^m
\end{aligned} \tag{3.2}$$

where l_i corresponds to the number of iterations of WL at step i of RPKM, the set of centroids C_{r+1}^i is induced by the clustering $\mathcal{G}_r^i$, and $\mathcal{G}_r^i$ is induced by C_r^i for $r = 1, ..., l_i - 1$ and $i = 1, .., m$. Each line corresponds to an execution of WL for a given partition $\mathcal{P}_i$ for $i = 1, ..., m$. It should be noted that, in step i of RPKM, the set of centroids C_0^i corresponds to the set of centroids obtained at the end of its previous step, that is $C_0^i = C_{l_i-1}^{i-1}$ for $i = 1, ..., m$. However, the clustering induced by $C_0^i = C_{l_i-1}^{i-1}$ for partition $\mathcal{P}_i$ does not have to correspond to the clustering induced for the previous partition $\mathcal{P}_{i-1}$. This fact is one of the main difficulties in guaranteeing a monotone decrement of the error function (see Eq.1.1) during an execution of RPKM.

In order to analyze the behavior of RPKM, we define the **clustering error** associated to a partition $\mathcal{P}$ as follows:

$$E_{\mathcal{P}}(\mathcal{G}) = \sum_{k=1}^{K} \sum_{S \in \mathcal{P}: S \subseteq G_k} |S| \cdot ||\overline{S} - \boldsymbol{c}_k||^2 \tag{3.3}$$

where the set of centroids C is induced by the clustering $\mathcal{G}$. This function measures the weighted error between each representative of a partition $\mathcal{P}$ and the center of mass of its corresponding cluster. Note that the only difference between the centroid error and clustering error is that, the centroid error is given in terms of a set of centroids and its induced clustering, while the clustering error is given by a clustering and its induced set of centroids. The importance of the clustering error is that it represents an intermediate value between the centroid errors obtained at two consecutive iterations of the algorithm, that is

$$E_{\mathcal{P}_i}(C_r^i) \geq E_{\mathcal{P}_i}(\mathcal{G}_r^i) \geq E_{\mathcal{P}_i}(C_{r+1}^i) \tag{3.4}$$

for $r = 0, ..., l_i - 1$ (see Eq.3.2). In the following subsections we will analyze the relation between the centroid error for different partitions of the data based on the inequality provided in Eq.3.4.

3.2.1 Evolution of the centroids error

In this section we analyze the evolution of the centroid error for RPKM. The obtained results will be the basis for bounding the number of iterations of WL at each step of the RPKM. The next result will be used in order to analyze the relation between the clustering error given two partitions of the data set (one thinner than the other).

Lemma 1 *Given a set of points X in $\mathbb{R}^d$ and a partition of it, $\mathcal{P}$. Then the function $f(\boldsymbol{c}) = |X| \cdot \|\overline{X} - \boldsymbol{c}\|^2 - \sum_{R \in \mathcal{P}} |R| \cdot \|\overline{R} - \boldsymbol{c}\|^2$ is constant.*

This result implies that the difference of the set of representatives with respect to a centroid, for two partitions of the data set (one thinner than the other), is constant. The fact that such a difference is constant allows us to state, in the following lemma, the invariance of the clustering error for two different partitions of the data set. Observe that Lemma 1 allows us to change the clustering, for both partitions, without changing the difference of the error associated to them.

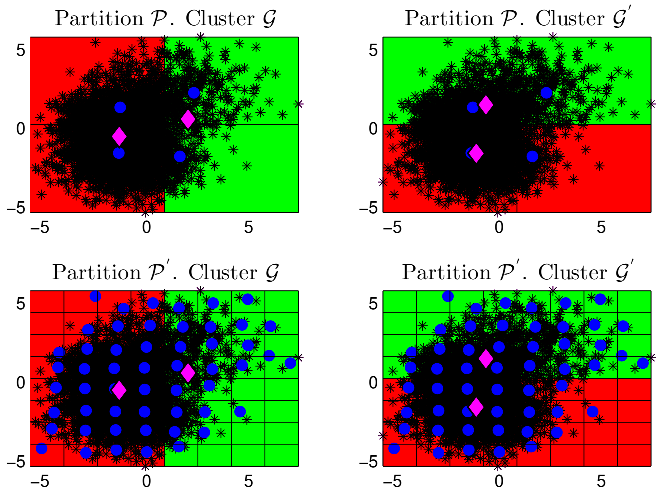

Fig. 3.2: Illustration of Lemma 2, for two clusterings $\mathcal{G}$ and $\mathcal{G}^{'}$ defined on a partition $\mathcal{P}$.

Lemma 2 *Let $\mathcal{P}$ and $\mathcal{P}'$ be two partitions of the data set X, with $\mathcal{P} \succ \mathcal{P}^{'}$, and let $\mathcal{G}$ and $\mathcal{G}'$ be two clusterings of $\mathcal{P}$. Then, the difference between both clustering errors is constant with respect to the partitions $\mathcal{P}$ and $\mathcal{P}'$:*

$$E_{\mathcal{P}}(\mathcal{G}) - E_{\mathcal{P}}(\mathcal{G}^{'}) = E_{\mathcal{P}'}(\mathcal{G}) - E_{\mathcal{P}'}(\mathcal{G}^{'})$$

In other words, the difference between two clustering errors is independent of the partition. For example, in Fig.3.2, clustering $\mathcal{G}$ restricts the subsets with center of mass to the left (right) of the middle point of the bounding box to belong to the same cluster. The pink diamonds represent the centers of mass of each group of $\mathcal{G}$, evidently such centers of mass are invariant with respect to the partition that we use to represent $\mathcal{G}$. Furthermore, Lemma 2 states that the difference of the clustering error difference between the upper figures is equivalent to the difference of the error associated to the lower figures in Fig.3.2.

In general, we can not guarantee a monotone descent of the error function given in Eq.1.1, which corresponds to the centroid error for the thinnest partition, i.e., $\mathcal{D} = \{\{x\} : x \in X\}$. However, in the next result, under mild conditions related to the difference of the centroid error, we prove a monotone descent of the clustering error for two partitions, one thinner than the other. In consequence, if the conditions stated for the difference of the clustering error hold for all the steps of RPKM, a monotone descent of the error function in Eq.1.1 is guaranteed.

Corollary 1 *Let C_i and C_{i-1} represent the set of centroids obtained at the i-th and (i − 1)-th RPKM step, respectively. Then $E_X(C_i) \leq E_X(C_{i-1})$, if and only if $E_X(\mathcal{G}^{i-1}_{l_{i-1}-1}) - E_X(C_{i-1}) \leq \xi_i + (E_X(\mathcal{G}^{i}_{l_i-1}) - E_X(C_i))$, where $\xi_i = E_{\mathcal{P}_i}(\mathcal{G}^{i-1}_{l_{i-1}-1}) - E_{\mathcal{P}_i}(\mathcal{G}^{i}_{l_i-1})$*

That is, if after the assignment step for both sets of centroids, C_i and C_{i-1}, with respect to the full data set, the condition in Theorem 1 is satisfied at every RPKM iteration, then we can guarantee the monotone descent of the error over the entire data set. In particular, if there are no reassignments for any of the two cases, with respect to their associated cluster membership, we can guarantee the monotone descent of the overall error. Clearly, as the difference of the local error of the initial and final cluster at the i-th RPKM step is larger, then it is more likely to satisfy such a condition.

Even when the monotone descent over the entire data set of the RPKM approximation, at every step, is not proved in general, we will see in the experiments summarized in Section 3.3 that, for real and artificial data sets, this property has been observed.

3.2.2 Bounding the iterations of the weighted Lloyd's algorithm

In this section, using the properties of the clustering error (Lemma 2), we can analyze the construction of the set of clusterings at different RPKM steps, for example we can verify the implications of repeating a clustering from a previous RPKM step.

In Lemma 3, we state that the unique clustering that can be repeated in a step of RPKM, is the previous clustering of the sequence of clusterings generated by the RPKM. If the repeated clustering is obtained at the first iteration of WL, then the previous clustering corresponds to the one obtained at the last iteration of WL (at the previous step of RPKM). On the other hand, if the repeated clustering is not obtained at the first iteration of WL, then the previous clustering corresponds to the one obtained at the previous iteration of WL (in the same RPKM step).

Lemma 3 *At the i-th step of the RPKM, if $\mathcal{G}^i_r = \mathcal{G}^j_s$, with $j < i$, for some $r \in \{1, \ldots, l_i - 1\}$ and $s \in \{1, \ldots, l_j - 1\}$, then $l_{j+1} = \ldots = l_i = 1$. Moreover, in that case, $s = l_j - 1$.*

In the following theorem, we use Lemma 3 to bound the number of WL iterations at each RPKM step. Lemma 3 indicates that the only cluster that can be repeated, from a previous RPKM step, is the last one generated by the corresponding WL execution. Therefore, we can eliminate, from the total number of possible clusterings, the ones that were generated at previous RPKM iterations (except the last one). In particular, if we have more than one Lloyd iteration at a certain RPKM step, then we automatically discard every single cluster that was generated at a previous RPKM step.

Theorem 1 *An upper bound to the number of Lloyd iterations at the i-th RPKM step is given by* $l_i \leq \left\{ {|\mathcal{P}_i| \atop K} \right\} - \sum_{j=1}^{i-1} (l_j - 1)$*, where* $\left\{ {|\mathcal{P}_i| \atop K} \right\}$ *is a Stirling number of the second kind.*

Following the same reasoning as in Theorem 1, observe that, if, at the $(i-1)$-th RPKM step, WL converges to the associated global optima, then $l_i \leq \left\{ {|\mathcal{P}_i| \atop K} \right\} - \left\{ {|\mathcal{P}_{i-1}| \atop K} \right\} + 1$. Moreover, observe that all the clusters with an error greater than $E_{\mathcal{P}_i}(\mathcal{G}^{i-1}_{l_{i-1}-1})$ will not be generated in the current or at any further RPKM iteration, however the amount of clusterings satisfying this condition can not be counted at the moment, one hypothesis is that the number of such clusterings is of order $O(\left\{ {|\mathcal{P}_i| \atop K} \right\})$.

For this reason, selecting the local initialization of WL in this manner may help reducing the number of Lloyd's iterations, while discarding, at each step, all the generated clusters (except one) and probably others of similar form. Not only that, but the discarding of such clusters occurs while analyzing a small number of representatives with regard to the full data set, which implies, as we will see in the experimental section, a drastic reduction in the number of distance computations.

3.3 Experimental section

In this section, we perform a set of experiments so as to analyze the relation between the number of distance computations and the quality of the approximation for the grid based RPKM algorithm proposed in Section 3.1. In addition, we analyze the effect on the algorithm performance of varying the different parameters of the clustering problem: size of the data set, n, dimension of the instances, d, and number of clusters, K. For the purposes of the experimental analysis, we compare the performance of the grid based RPKM algorithm against the **K-means++** (**KM++**) and the **minibatch K-means** (**MB**) on artificial and real data sets.

The grid based RPKM was implemented in Python, while we used the KM++ and MB implementations that are available in the open source machine learning library *scikit-learn* of Python. As stopping criterion for the RPKM, we just set a maximum number of iterations, m, since we want to analyze the behavior of the error function, at each step, as the number of representatives approaches the number of instances. For the analyzed data sets, we observe that, with $m \leq 6$, this occurs. Evidently, as we increase the dimension, this property will be seen inmediately, since the number of representatives increases exponentially with respect to this parameter.

In this section, we refer to the result obtained after the m-th step of the grid based RPKM by **RPKM** m, and to the solution obtained using MB with a batch size $b \in \{100, 500, 1000\}$ by **MB** b. In equivalent experimentations similar batch sizes were used [30].

3.3.1 Artificial data sets results

The artificial data sets are generated as a d-dimensional mixture of K Gaussians. In particular, we set $K \in \{3, 9\}$, $d \in \{2, 4, 8\}$ and $n \in \{1000, 10000, 100000, 1000000\}$. For each setting, we generate 50 replicates of the data set. Additionally, we consider a component overlapping lower than 5%.

3.3.1.1 Distance computations

In this section, we compare the behavior of RPKM, KM++ and MB in terms of the computed distances. As we commented in Section 1.1.1.2 and Section 3.1.2, the most time consuming phase of the Lloyd's algorithm, and its weighted version, refers to the computation of distances. Especially at the initial steps, RPKM considers a number of representatives which is a small fraction of the size of the data set. Thus, we would expect a greater reduction in the number of distance computations, with respect to the other methods, as we consider larger data sets.

In Fig.3.3, we present the relation between the number of distance computations and the data set size for the different settings.

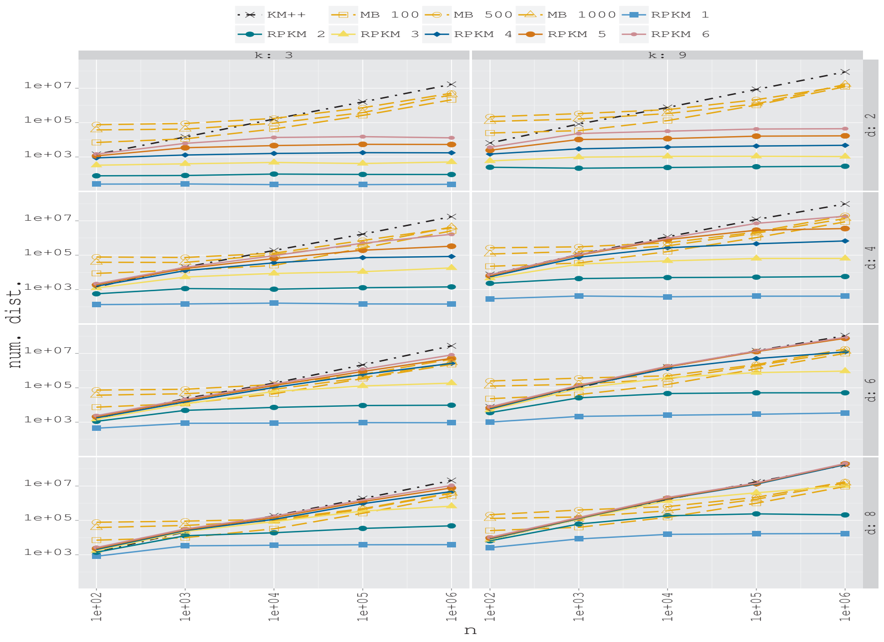

Fig. 3.3: This figure shows the number of distance computations with respect to the size of the data set (n), for different numbers of dimensions (d) and numbers of clusters (K).

At first glance, we observe that RPKM, in general, executes a much smaller number of distance computations than both KM++ and MB. Such a relation seems to change for the latter steps of RPKM when we consider larger dimensions. However, in that case, KM++ still requires a similar order of computations in comparison to the latter steps of the RPKM. Analogously, MB, for the different batches, is not able to match the number of distance computations of RPKM at its first steps, for any of the analyzed settings. In addition, we observe that, for some RPKM steps, the number of distance computations does not increase as we consider a higher number of instances, as happens with the other algorithms.

In particular, we notice that the number of distance computations, at the first steps of the RPKM, i.e., RPKM 1 and RPKM 2, does not necessarily increase with respect to the data set size. This is plausible since, in this case, the number of representatives is of the same order, independent of the number of instances (see Fig.3.4). Evidently, as we consider thinner partitions ($m \geq 3$), the number of representatives will increase and so will the number of distance computations.

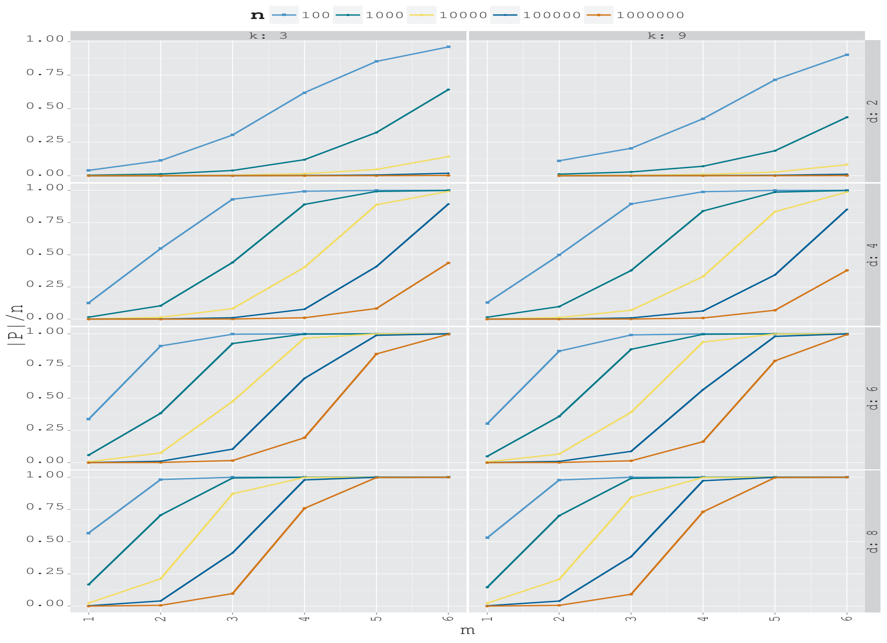

Fig. 3.4: This figure shows the ratio between the size of the partition P_m and the size of the data set n, for $m = 1, ..., 6$. The size of the partition $\mathcal{P}_m$ corresponds to the number of representatives used by RPKM at its m-th step.

Since, at the earlier stages of the RPKM, the number of representatives does not necessarily increase with respect to the data set size, then the ratio between the number of distance computations of RPKM and KM++ (or MB) decreases with respect to the size of the data set. In particular, for the larger number of instances, the number of distances computed by RPKM with respect to KM++ is 3 orders of magnitude lower for $K = 3$ and $d = 8$ and 6 orders of magnitude lower for $K = 3$ and $d = 2$. In comparison with MB, the number of distances computed by RPKM is 2 orders of magnitude lower for $K = 3$ and $d = 8$ and 5 orders of magnitude lower for $K = 3$ and $d = 2$. As we can see, the dimensionality of the problem, d, has a great impact on the number of distances computed by RPKM as m increases. The reason is that the number of representatives used by RPKM can increase exponentially with respect to both m and d. In addition, we can see that the number of distance computations, for all the algorithms, as expected, increases linearly with the number of clusters K.

3.3.1.2 Quality of the approximation

In the previous section, we observed that RPKM entails a drastic reduction in the amount of distance computations with respect to the other approaches (especially when we consider large data set sizes). However, in this section, we would like to analyze the quality of the approximations obtained by means of RPKM.

In Fig.3.5, we show the evolution of the standarized error (std.error) for the full data set for the set of centroids obtained at the end of the m-th step of the RPKM. The std.error is defined as $\rho(m) = \frac{E^*_m - E_m}{E^*_m}$, where E_m is the error for RPKM at the m-th step, and E^*_m is the error obtained by K-means algorithm over the full data set X, taking as initialization the centroids obtained by RPKM at the m-th step. Observe that $\rho(m) \leq 0$ and it measures the percentage of error with respect to the K-means over the entire data set.

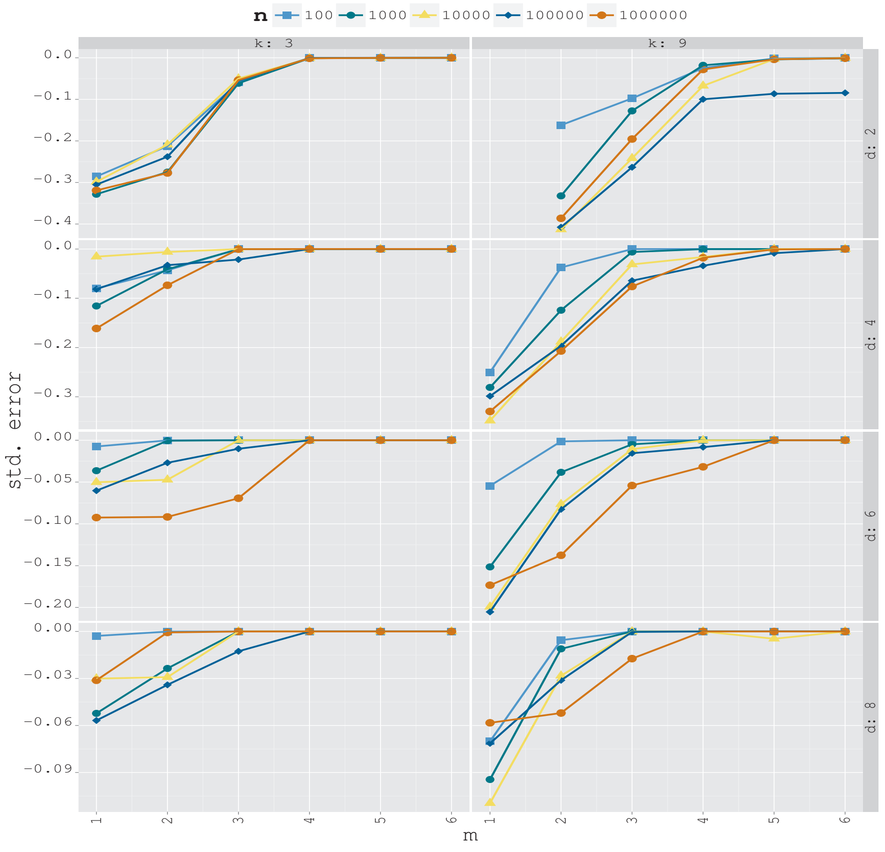

Fig. 3.5: Quality of the approximation (std.error) with respect to the RPKM step.

In most of the cases, we observe a monotone descent of the centroid error with respect to the full data set until convergence to the error associated to a solution of the K-means algorithm. This is remarkable since the approximation is constructed over a reduced number of representatives with respect to the total number of instances (see Fig.3.4). In particular, at the third RPKM step, the error with respect to the K-means solution is under 10%. Evidently, as we increase the dimension, this percentage decreases until it achieves an error percentage smaller than 10% and 5% for the first and second RPKM step, respectively. Moreover, for $d = 2$ and $K = 9$, we observe no result for RPKM 1, this is due to the fact that, in this case, the number of representatives is smaller than the number of clusters, i.e., $|\mathcal{P}_1| < K$.

3.3.1.3 Relation distance computations - quality of the approximation

In this section, we fuse the results obtained at the previous sections and analyze, for the different algorithms, the trade-off between the number of distances computed and the quality of the obtained solutions.

In Fig.3.6, we show the relation between the number of distance computations and the error of the obtained solutions for RPKM, KM++ and MB.

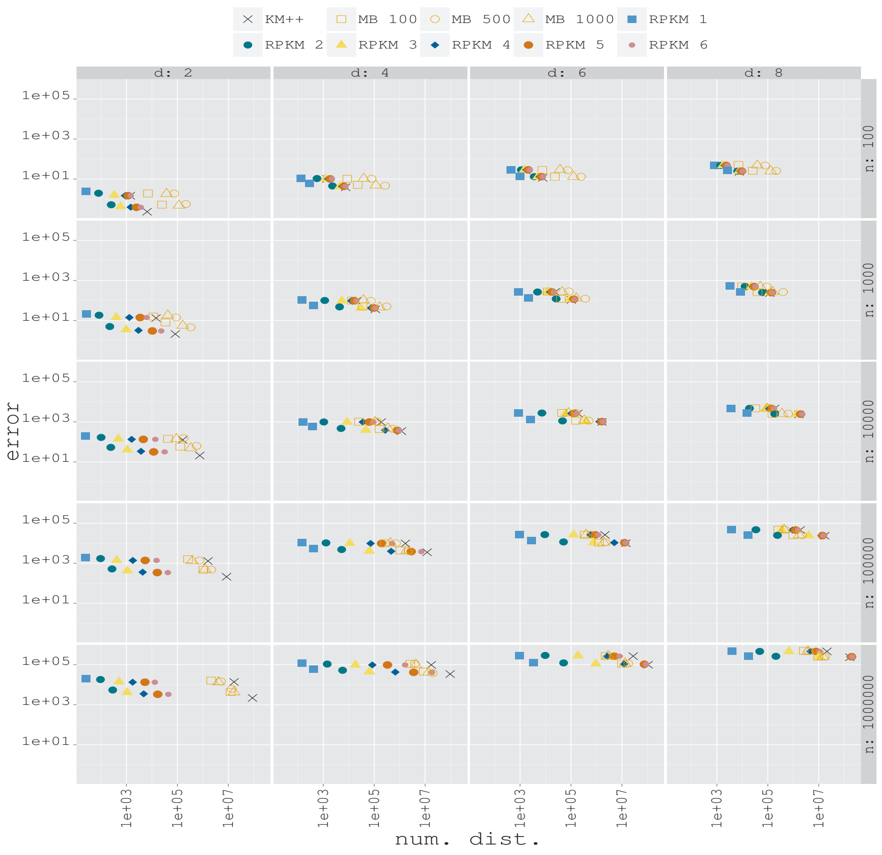

Fig. 3.6: Quality of the approximation vs number of distance computations.

Besides the dimensionality of the problem, as we increase the number of instances, the cloud of points associated to KM++ and MB separates from the ones associated to the RPKM. This means that, as we increase the number of instances, KM++ and MB require a much larger number of distance computations in order to achieve a solution of similar quality to those obtained by the RPKM. In the best case scenario ($n = 1000000$, $d = 2$), RPKM reduces, at least, 6 and 4 orders of magnitude with respect to the K-means++ and the minibatch K-means. Evidently, for larger dimensions, the clouds associated to the RPKM, for the latter stages, can overlap those of KM++ and MB. In this case, even for the largest number of instances, we did not need to execute all the RPKM steps: at the fifth RPKM step, we have already generated, approximately, as many representatives as instances.

In particular, consider the extreme cases: $d = 2$, $n = 1000000$ (case 1) and $d = 8$, $n = 100$ (case 2). In the first case, after the third RPKM step the standard error is already under 5% and, after fourth step, the error is practically null. Such approximations are obtained after computing under 10^{-3}% (10^{-2}%) and slightly over 10^{-3}% (10^{-2}%) of the distances calculated by KM++ (MB) for RPKM 3 and RPKM 4, respectively.

In the second case, already after the first RPKM step, the standard error is under 8% and, after the second step, the error is fairly close to zero. Such approximations are obtained after computing under 10^{-1}% (1%) and under 1% (10%) of the distances calculated by KM++ (MB) for RPKM 1 and RPKM 2, respectively.

In the case of lower dimensions and greater number of instances, we require more RPKM steps to achieve an approximation with a similar standard error with respect to the case with greater dimensions and lower number of instances. As previously mentioned, this is due to the exponential growth of the number of representatives with respect to the dimension of the problem. Moreover, in the second case, we have a lower number of instances, hence, we need fewer RPKM steps in order to generate as many representatives as instances (in this example, this occurs for $m = 3$). However, having a lower proportion of representatives with respect to the number of instances also implies a greater reduction in the number of distance computations with respect to the full data set for the first case, while obtaining a similar standard error in comparison to the second case.

3.3.2 Real data sets

In addition to the previous experimentation, we evaluate the performance of the grid based RPKM algorithm, KM++ and MB on a real-world data set: the gas sensor array under dynamic gas mixtures data set, which contains the acquired time series from 16 chemical sensors exposed to gas mixtures at varying concentration levels [77]. The data set consists of 4178504 instances and 19 attributes and is available in the UC-Irvine Machine Learning Repository. The same experiment was performed over different data sets from the UC-Irvine Machine Learning Repository, achieving similar conclusions. For the sake of brevity, the corresponding graphics are not included in this work.

Using this real-world data set, we generate different subsamplings that we use to analyze the features of the algorithms. In particular, we take $d \in$

$\{2, 4, 8\}$ random attributes and $n \in \{4000, 12000, 40000, 120000, 400000, 1200000, 4000000\}$ random instances. The number of clusters is $K \in \{3, 9\}$. For each setting, we generate 10 replicates of the data set.

3.3.2.1 Distance computations

For the real data set experimentation, we perform the same analysis as that carried out for the artificial data sets case.

In Fig.3.7, as in Fig.3.3, we present the relation between the number of distance computations and the data set size, this time for the real data set. In general, we observe a similar behavior with respect to the artificial data sets case: RPKM reduces, in many orders of magnitude, the number of distance computations with respect to KM++ and MB. However, in this case, even at the last step of RPKM, the number of distances does not always increase with respect to the number of instances.

In particular, as can be verified in Fig.3.8, the number of representatives does not necessarily increase as we consider higher data set sizes. This is due to the fact that the data points, in this case, are grouped into more condensed clouds than in the case of the artificial Gaussian data set case. Hence, it is plausible to observe that the number of distance computations, even at the latter stages of the RPKM, does not necessarily increase with respect to the data set size. Furthermore, for low dimensions, this fact implies that, even at the last RPKM step, we have a lower number of distance computations than any version of MB and KM++.

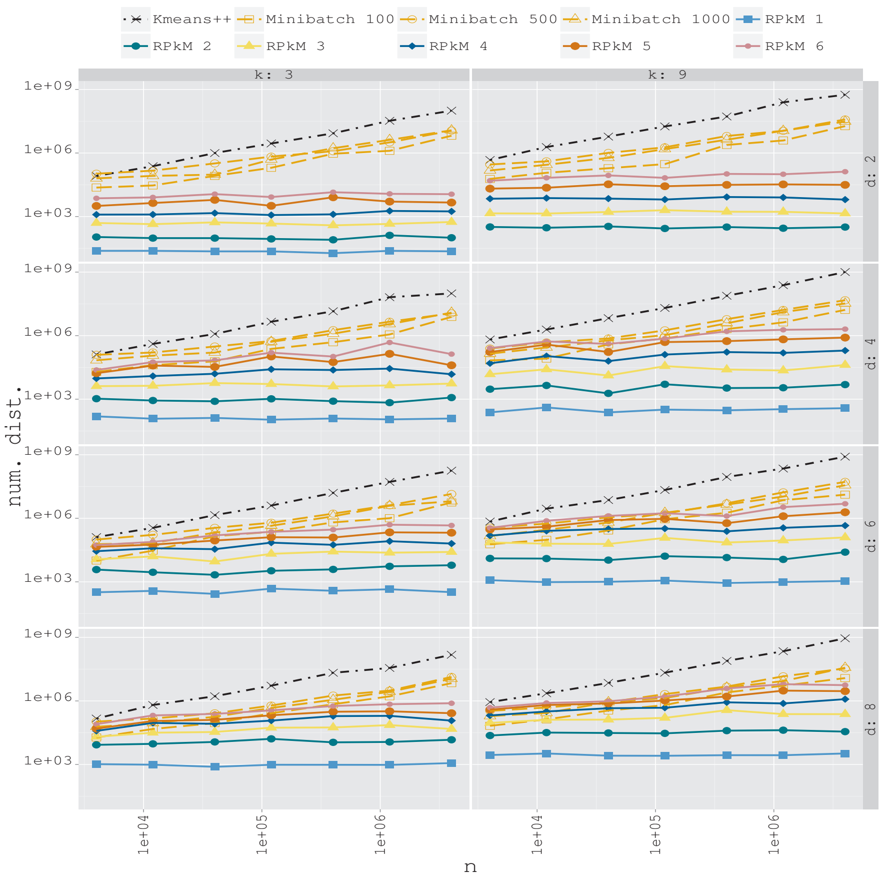

Fig. 3.7: Number of distance computations with respect to data set size, n

In more detail, we notice that for the largest number of instances, the last step of the RPKM executes less than 1% and 0.1% of the distances computed by MB and KM++, respectively. For the largest dimension considered ($d = 8$), the latter steps RPKM execute a similar order of distance computations with respect to MB. However, intermediate RPKM steps (RPKM 3) still computes less than 1% and 0.1% of the distances computed with respect to MB and KM++. It is important to remember that the number of distance computations for KM++ and MB is independent of the dimensionality of the problem, which is not usually true for the RPKM algorithm.

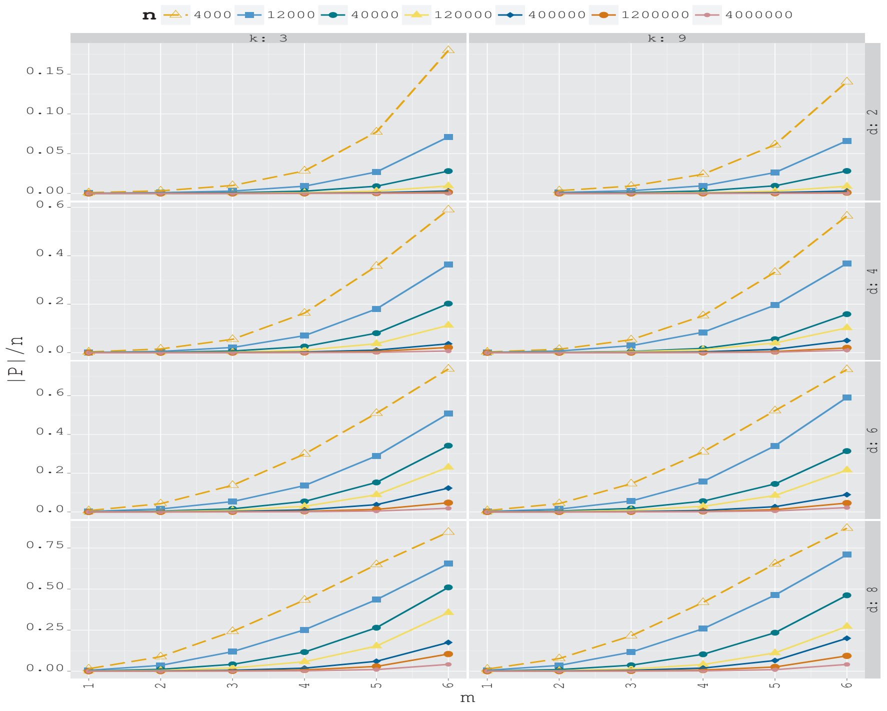

Fig. 3.8: Percentage of subsets with respect to RPKM step

For the analyzed data set, we observe that the number of representatives does not grow as rapidly as in the artificial sets case. In particular, for the lowest dimension, the number of representatives barely achieves 20% of the data set size at the last grid based RPKM step. As we increase the dimensionality, and therefore generate more representatives, this number grows to over 75% for the shortest data set size case.

3.3.2.2 Quality of the approximation

In Fig.3.9, we observe the evolution of the standarized error, for the full data set, with respect to the set of centroids obtained at the m-th step of the RPKM. As commented in the artificial data sets case, in most of the cases, there is a monotone descent of the centroid error with respect to the full data set until convergence to the error associated to a solution of the K-means algorithm over the full data set. Commonly, at the third RPKM step, the such standarized error is under 10%. This is remarkable since, as can be seen in Fig.3.7, the number of distances computed by RPKM 3, compared to MB and KM++, is under 10^{-3} % and 10^{-4} %, respectively.

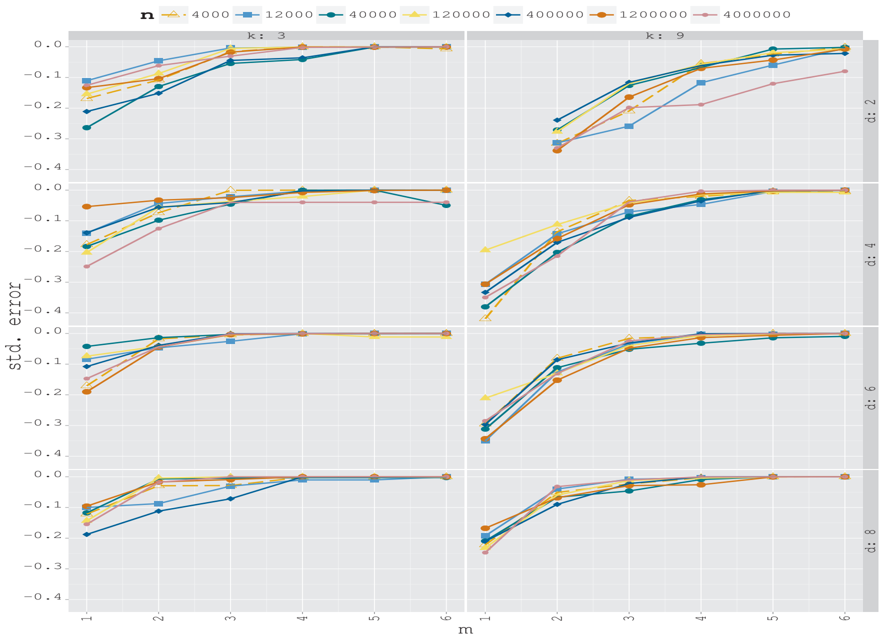

Fig. 3.9: Quality of the approximation with respect to RPKM step

As we increase the dimension the standarized error, in most of the cases, is under 10%, even for RPKM 2 on the largest data set size case.

3.3.2.3 Relation distance computations - quality of the approximation

In order to have a better understanding of the relation distance computations against overall error, we consider Fig.3.10. As we observed in the artificial data set case, when we increase the number of instances, the cloud of points associated to KM++ and MB separates from the ones associated to the RPKM. As we increase the number of instances, KM++ and MB require a much larger number of distance computations in order to achieve a solution of similar quality than those obtained by the RPKM. For the lowest dimensional case, even for an intermediate RPKM step, it reduces over 4 orders ($d = 2, n = 4000000$) and 5 orders of magnitude ($d = 2, n = 4000000$) with respect to MB and KM++, respectively. Evidently, as we increase the dimensionality of the problem and, therefore, generate a larger amount of representatives and compute more distances, we still observe a reduction of 4 and 5 orders of magnitude, but for the first RPKM step. In addition, we can see that the minibatch k-means with the smallest batch, MB 100, has, in some cases, a similar amount of distance computations with respect to the second step of RPKM.

Such a reduction in the number of distance computations is achieved while still obtaining competitive approximations. For instance, even in the case of greater dimension and lower number of instances ($d = 8$, $n = 4000$), after the second RPKM step the standard error is already under 5% and, after the third step, the error is practically null. These approximations are obtained after computing under $7 \cdot 10^{-4}$% ($2 \cdot 10^{-3}$%)and 10^{-3}% ($5 \cdot 10^{-3}$%) of the distances calculated by KM++ (MB), for RPKM 3 and RPKM 4, respectively.

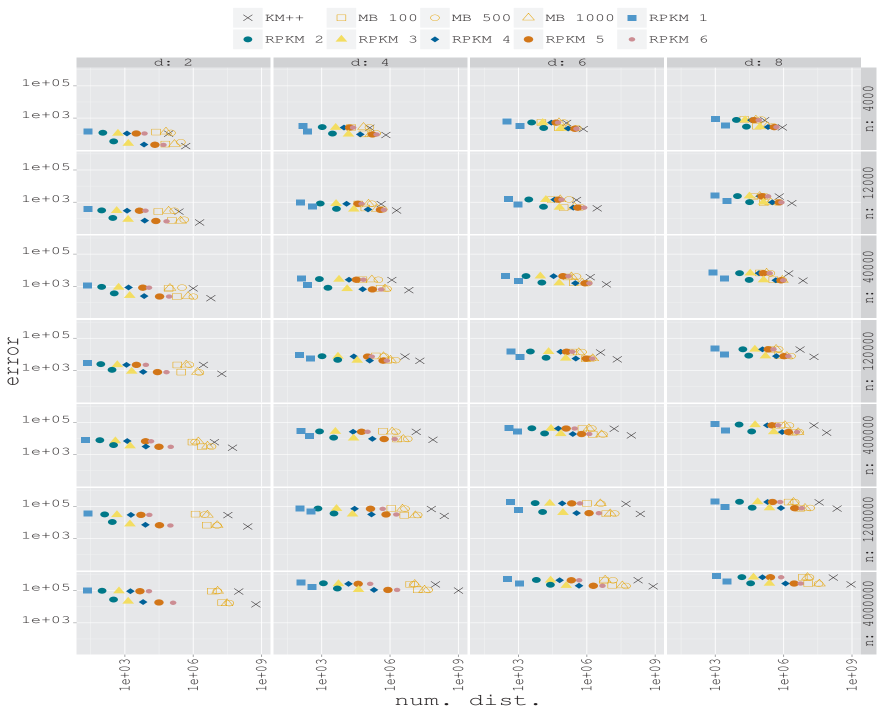

Fig. 3.10: Quality of the approximation vs number of distance computations

In general, we observe that the grid based RPKM algorithm is able to generate competitive approximations with a significant reduction in the number of distance computations. As the data set size increases, we observed a more drastic reduction in the number of distance computations as can be seen in Fig.3.6 and Fig.3.10. We observe the same behavior as we increase the dimension of the instances, however, the order of reduction with respect to MB and KM++ decreases, as expected. This is due to the fact that the number of RPKM representatives, in this case, grows exponentially with respect to the dimension of the data set.

3.4 Conclusions

In this chapter, we present an alternative to the K-means algorithm applicable to massive data problems called recursive partition based K-means (RPKM). This approach recursively partitions the entire data set into a small number of subsets, each of which is characterized by its representative (center of mass) and weight (cardinality), after that a weighted version of the Lloyd's algorithm is applied over this local representation. The objective is to describe the full data set by this representation, which ultimately leads to a reduction of distance computations. Indeed, in the experimental section, we observe that the RPKM algorithm generates competitive approximations, even at its earlier iterations, while reducing several orders of magnitude of distance computations.

In Section 3.2, we have derived some theoretical properties of the RPKM. Among other results, we can guarantee the non repetition of the clusterings generated at each RPKM iteration (except the last one), which ultimately implies the reduction of the total amount of Lloyd iterations, as well as leading, in most of the cases, to a monotone decrease of the overall error function.

In the experimental section, the grid based RPKM was compared with two well-known approaches: the K-means++ and minibatch K-means. In this analysis, we observed a dramatic reduction in the number of distance computations with respect to both of them, as well as a consistent monotone decrease of the error function. Since the RPKM algorithm seeks to reduce the number of representatives used per iteration, we observed a larger reduction in the number of distance computations as we enlarged the number of instances of the data set. Furthermore, at the earlier stages of the RPKM, the size of the data set did not have a relevant impact on the number of iterations or distance computations for the associated weighted K-means problem. Thus, the number of computations, especially for massive data applications, can be greatly reduced.

On the other hand, it is important to remark that the number of subpartitions generated at each iteration of the grid based RPKM grows exponentially with respect to the dimension, d, of the data set. As we can see in Fig.3.1, some of these subpartitions might have a small probability of changing their current cluster affiliation (with respect to the assignment given by the previous RPKM iteration). In this sense, it might be more valuable to partition the areas that are more likely to have subpartitions associated with different clusters.

One possible approach consists of characterizing the subsets that lie on a cluster boundary, i.e., subsets that are close to two or more clusters. In this approach, the number of representatives does not grow exponentially with respect to the dimension of the data set. For this reason, as a future step, we plan to define a low computational cost algorithm to determine the cluster boundary at each iteration. The subsets in this area will have a greater priority when selecting the regions that we want to partition in the next RPKM iteration.

One last, but still important, advantage of the RPKM algorithm is the fact that its parallelization is direct. The RPKM algorithm mainly depends on two steps: The data partition process and the application of the weighted version of Lloyd's algorithm. In the first step, each point can independently decide which subset it belongs to, hence the construction of the set of representatives and weights can be done in a parallel manner. Analogously, for the second phase, given a set of prototypes, each data point can separately decide which cluster it belongs to and the update of the centroid can be simply computed by averaging the points [21]. For this reason, we also plan to implement the RPKM algorithm on a parallel framework such as *Apache Spark*.

4

An efficient K-means clustering algorithm for tall data

In this chapter, we introduce an improvement to the partition strategy followed in Chaper 3. The experimental results presented in Chapter 3 refer to a RPKM variant called grid based RPKM. For such a partition strategy, the initial spatial partition is defined by the grid obtained after dividing each side of the smallest bounding box of X by half, i.e., a grid with 2^d equally sized blocks. In the same fashion, at the i-th grid based RPKM iteration, the corresponding spatial partition is updated by dividing each of its blocks into 2^d new blocks, i.e., $\mathcal{P}$ can have up to $2^{i \cdot d}$ representatives. It can be shown that this approach produces a (K,ε)-coreset with ε descending exponentially with respect to the number of iterations [1].

Taking this into consideration, three main problems arise for the grid based RPKM:

- **Problem 1.** *It does not scale well on the dimension d*: Observe that, for a relatively low number of iterations, $i \simeq \log_2(n)/d$, and/or dimensionality $d \simeq \log_2(n)$, applying this RPKM version can be similar to applying Lloyd's algorithm over the entire data set, i.e., no reduction of distance computations might be observed, as $|\mathcal{P}| \simeq n$. In fact, for the experimental section in Chapter 3, $d, i \leq 10$.
- **Problem 2.** *It is independent of the data set X*: As we mentioned before, regardless of the analyzed data set X, the sequence of partitions of the grid based RPKM is induced by an equally sized spatial partition of the smallest bounding box containing X. In this sense, the induced partition does not consider features of the data set, such as its density, to construct the sequence of partitions: A large amount of computational resources might be spent on regions whose misclassification does not add a significant error to our approximation. Moreover, the construction of every partition of the sequence has a $O(n \cdot d)$ cost, which is particularly expensive for massive data applications, as n can be huge.

[1] See Theorem 9 at Appendix .2.1 in the supplementary material.

- **Problem 3.** *It is independent of the problem*: The partition strategy of the grid based RPKM does not explicitly consider the optimization problem that K-means seeks to minimize. Instead, it offers a simple/inefficient way of generating a sequence of spatial thinner partitions. The reader should note that each block of the spatial partition can be seen as a restriction over the K-means optimization problem, that enforces all the instances contained in it to belong to the same cluster. Therefore, it is of our interest to design smarter spatial partitions oriented to focus most of the computational resources on those regions where the correct cluster affiliation is not clear. By doing so, not only can a large amount of computational resources be saved, but also some additional theoretical properties can be deduced. Among other properties that we discuss in Section 4.1, at first glance it can be observed that if all the instances in a set of points, P, are correctly assigned for two sets of centroids, C and C', then the difference between the error of both sets of centroids is equivalent to the difference of their weighted error, i.e., $E_P(C) - E_P(C') = E_{\{P\}}(C) - E_{\{P\}}(C')$ [2]. Moreover, if this occurs for each subset of a data set partition, $\mathcal{P}$, and the centroids are generated after consecutive weighted K-means iterations, then we can guarantee a monotone decrease of the error for the entire data set [3]. Likewise, we can actually compute the reduction of the error for the newly obtained set of centroids, without computing the error function for the entire data set, as in this case $E_X(C) - E_X(C') = E_{\mathcal{P}}(C) - E_{\mathcal{P}}(C')$. Last but not least, when every block contains instances belonging to the same cluster, the solution obtained by our weighted approximation is actually a local optima of Eq.1.1 [4].

In any case, independently of the partition strategy, RPKM algorithm offers some interesting properties such as the no clustering repetition. This is, none of the obtained groupings of the n instances into K groups can be repeated at the current RPKM iteration or for any thinner partition than the current one. This is a useful property since it can be guaranteed that the algorithm discards many possible clusterings at each RPKM iteration using a much reduced set of points than the entire data set. Furthermore, this fact enforces the decrease of the maximum number of Lloyd iterations that we can have for a given partition. In practice, it is also common to observe a monotone decrease of the error for the entire data set Chapter 3.

Bearing all these facts in mind, we propose a RPKM type approach called the *Boundary Weighted K-means* algorithm (**BWKM**). The idea behind it is to prioritize the use of resources on the cluster boundaries of our weighted approximation, which are constituted by those blocks that may not be well assigned.

[2] See Lemma 4 in Appendix .2.1 in the supplementary material.
[3] See Theorem 10 in Appendix .2.1 in the supplementary material.
[4] See Theorem 4 in Appendix .2.1 in the supplementary material.

Definition 7 (Well assigned blocks) *Let C be a set of centroids and X be a given data set. We say that a block B is well assigned, with respect to C and X, if every point $\boldsymbol{x} \in B(X)$ is assigned to the same centroid $c \in C$.*

The notion of well assigned blocks is of our interest as RPKM associates all the instances contained in a certain block to the same cluster, which corresponds to the one that its center of mass belongs to. Hence, our goal is to divide those blocks that are not well assigned. Moreover, in order to control the growth of the set of representatives and to avoid unnecessary distance computations, we have developed a non-expensive partition criterion that allows us to detect blocks that may not be well assigned. Our main proposal can be divided into three tasks:

- **Task 1**: Design of a partition criterion that decides whether or not to divide a certain block, using only information obtained from the weighted Lloyd's algorithm.
- **Task 2**: Construct an initial partition of the data set with a predefined number of blocks, which are mostly placed on the cluster boundaries.
- **Task 3**: Once a certain block is decided to be cut, guarantee a low increase on the number of representatives without affecting, if possible, the quality of the approximation. In particular, we propose a criterion that, in the worst case, has a linear growth in the number of representatives after an iteration.

Observe that, both **Task 2** and **Task 3**, ease the scalability of the algorithm with respect to the dimensionality of the problem, d (**Problem 1**). Furthermore, the goal of **Task 1** and **Task 2** is to generate partitions of the data set that, ideally, contain well assigned subsets, i.e., all the instances contained in a certain subset of the partition belong to the same cluster (**Problem 2** and **Problem 3**). As we previously commented, this fact implies additional theoretical properties in terms of the quality of our approximation.

At this point, it must be remarked that due to the nature of BWKM, and unlike the typical coreset approach, our proposal does not intend to generate competitive approximations of the K-means error function, Eq.1.1. Instead, BWKM controls the size of the corresponding weighted set of points by placing large grid cells on those regions where the correct cluster affiliation is known. This increases the difference between the K-means error and our weighted approximation in these regions [5] but, at the same time, allows us to guarantee a monotone descend and convergence to a local minima of Eq.1.1.

The rest of this article is organized as follows: In Section 4.1, we describe the proposed algorithm, introduce some notation and discuss some theoretical properties of our proposal. Finally, in Section 4.2, we present a set of experiments in which we analyze the effect of different factors, such as the size of

[5] However in Theorem 3 we propose a bound for this error and for the quality of the weighted approximation.

the data set and the dimension of the instances over the performance of our algorithm. Additionally we compare these results with the ones obtained by the state-of-the-art.

4.1 BWKM algorithm

In this section, we present the Boundary Weighted K-means algorithm. As we already commented, BWKM is a scalable improvement of the grid based RPKM algorithm [6], that generates competitive approximations to the K-means problem, while reducing the amount of computations that the state-of-the-art algorithms require for the same task. BWKM reuses all the information generated at each weighted Lloyd run to construct a sequence of thinner partitions that alleviates **Problem 1**, **Problem 2** and **Problem 3**.

Our new approach makes major changes in all the steps in Algorithm 2 except in `Step 2` and `Step 4`. In these steps, the weighted version of Lloyd's algorithm is applied over the set of representatives and weights of the current data set partition, $\mathcal{P}$. This process has a $O(|\mathcal{P}| \cdot K \cdot d)$ cost, hence it is of our interest to control the growth of $|\mathcal{P}|$, which is highlighted in both **Task 2** and **Task 3**.

In the following sections, we will describe in detail each step of BWKM. In Section 4.1.1, Section 4.1.2 and Section 4.1.3 we elaborate on **Task 1**, **Task 2** and **Task 3**, respectively.

4.1.1 A cheap criterion for detecting well assigned blocks

BWKM tries to efficiently determine the set of well assigned blocks in order to update the data set partition. In the following definition, we introduce a criterion that will help us verify this, mostly using information generated by our weighted approximation:

Definition 8 *Given a set of K centroids, C, a set of points $X \subseteq \mathbb{R}^d$ and $P = B(X) \neq \emptyset$ the subset of points contained in a block B, we define the misassignment function for B, given C and X, as:*

$$\epsilon_{C,X}(B) = \max\{0, 2 \cdot l_B - \delta_P(C)\}, \tag{4.1}$$

where $\delta_P(C) = \min_{\boldsymbol{c} \in C \setminus \boldsymbol{c}_{\overline{P}}} \|\overline{P} - \boldsymbol{c}\| - \|\overline{P} - \boldsymbol{c}_{\overline{P}}\|$ and l_B is the length of the diagonal of B. In the case $P = B(X) = \emptyset$, we set $\epsilon_{C,X}(B) = 0$.

The following result is used in the construction of both the initial and the sequence of thinner partitions:

Theorem 2 *Given a set of K centroids, C, a data set, $X \subseteq \mathbb{R}^d$, and a block B, if $\epsilon_{C,X}(B) = 0$, then $\boldsymbol{c}_{\boldsymbol{x}} = \boldsymbol{c}_{\overline{P}}$ for all $\boldsymbol{x} \in P = B(X) \neq \emptyset$.*[7]

[6] From now on, we assume each block $B \in \mathcal{B}$ to be a hyperrectangle.
[7] The proof of Theorem 2 is in Appendix .2.1 in the supplementary material.

In other words, if the misassignment function of a block is zero, then the block is well assigned. Otherwise, the block may not be well assigned. Even though the condition in Theorem 2 is a sufficient condition, we will use the following heuristic rule during the development of the algorithm: The larger the misassignment function of a certain block is, then the more likely it is to contain instances with different cluster memberships.

In particular, Theorem 2 offers an efficient and effective way of verifying that all the instances contained in a block B belong to the same cluster, using only information related to the structure of B and the set of centroids, C. Observe that we do not need any information associated to the individual instances in the data set, $\mathbf{x} \in P$. The criterion just requires some distance computations with respect to the representative of P, $\overline{P}$, that are already obtained from the weighted Lloyd's algorithm.

Definition 9 *Let X be a data set, C be a set of K centroids and $\mathcal{B}$ be a spatial partition. We define the boundary of $\mathcal{B}$, given C and X, as*

$$\mathcal{F}_{C,X}(\mathcal{B}) = \{B \in \mathcal{B} : \epsilon_{C,X}(B) > 0\} \tag{4.2}$$

The boundary of a spatial partition is just the subset of blocks with a positive misassignment function value, that is, the blocks that may not be well assigned. In order to control the size of the spatial partition and the number of distance computations, BWKM only splits blocks from the boundary.

In Fig.4.1, we observe the information needed for a certain block of the spatial partition, the one marked out in black, to verify the criterion presented in Theorem 2. In this example, we only set two cluster centroids (blue stars) and the representative of the instances in the block, $\overline{P}$, given by the purple diamond. In order to compute the misassignment function of the block, we require the length of the **three** segments: Distance between the representative with respect to its two closest centroids in C (blue dotted lines) and the diagonal of the block (purple dotted line). If the misassignment function is zero, then we know that all the instances contained in the block belong to the same cluster. Observe that, in this example, there are instances in both clusters, then the block is included in the boundary.

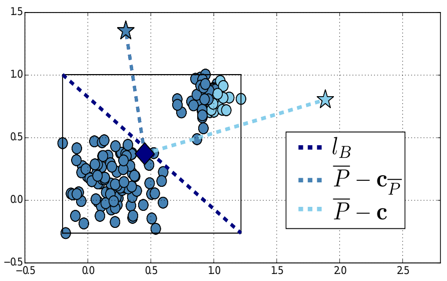

Fig. 4.1: Information required for computing the misassignment function of the block B, $\epsilon_{C,X}(B)$, for $K = 2$.

Theorem 3 *Given a data set, X, a set of K centroids C and a spatial partition $\mathcal{B}$ of the data set X, the following inequality is satisfied:*

$$|E_X(C) - E_{\mathcal{P}}(C)| \leq \sum_{B \in \mathcal{B}} 2 \cdot |P| \cdot \epsilon_{C,X}(B) \cdot (2 \cdot l_B + \|\overline{P} - \boldsymbol{c}_{\overline{P}}\|) + \frac{|P| - 1}{2} \cdot l_B^2,$$

where $P = B(X)$ and $\mathcal{P} = \mathcal{B}(X)$. Furthermore, for a well assigned partition $\mathcal{P}$, if we define $C^{\mathcal{P}}_{OPT} = \underset{C \subset \mathbb{R}^d, |C|=K}{\arg\min} \; E_{\mathcal{P}}(C)$ and $C_{OPT} = \underset{C \subset \mathbb{R}^d, |C|=K}{\arg\min} \; E_X(C)$, then

$$E_X(C^{\mathcal{P}}_{OPT}) \leq E_X(C_{OPT}) + (n - |\mathcal{P}|) \cdot l^2,$$

where $l = \max_{B \in \mathcal{B}} l_B$. [8]

According to this result, we must increase the amount of well assigned blocks and/or reduce the diagonal lengths of the blocks of the spatial partition, so that our weighted error function approximates better the K-means error function, Eq.1.1. Observe that by reducing the diagonal of the blocks, not only is the condition of Theorem 2 more likely to be satisfied, but also we are directly reducing both additive terms of the bound in Theorem 3. This last point gives the intuition for our new partition strategy: i) split only those blocks in the boundary and ii) split them on their largest side.

4.1.2 Initial Partition

In this section, we elaborate on the construction of the initial data set partition used by the BWKM algorithm (see `Step 1` of Algorithm 8, where the main

[8] The proof of Theorem 3 is in Appendix .2.1 in the supplementary material.

pseudo-code of BWKM is). Starting with the smallest bounding box of the data set, the proposed procedure iteratively divides subsets of blocks of the spatial partition with high probabilities of not being well assigned. In order to determine these blocks, in this section we develop a probabilistic heuristic based on the misassignment function, Eq.4.1.

As our new cutting criterion is mostly based on the evaluation of the misassignment function associated to a certain block, we firstly need to construct a starting spatial partition of size $m' \geq K$, from where we can select the set of K centroids with respect to which the misassignment function is computed (`Step 1`).

From then on, multiple sets of centroids C are selected via a weighted K-means++ run over the set of representatives of the data set partition, for different subsamplings. This will allow us to estimate a probability distribution that quantifies the chances of each block of not being well assigned (`Step 2`). Then, according to this distribution, we randomly select the most promising blocks to be cut (`Step 3`), and divide them until reaching a predefined number of blocks m (`Step 4`). In Algorithm 5, we show the pseudo-code of the algorithm proposed for generating the initial spatial partition.

Algorithm 5: Construction of the initial partition

```
Input: Dataset $X$, number of clusters $K$, integer $m' > K$, size of the
initial spatial partition $m > m'$.
Output: Initial spatial partition $\mathcal{B}$ and its induced data set partition,
$\mathcal{P} = \mathcal{B}(D)$.
Step 1: Obtain a starting spatial partition of size $m'$, $\mathcal{B}$ (Algorithm 6).
while $|\mathcal{B}| < m$ do
    Step 2: Compute the cutting probability, $Pr(B)$ for $B \in \mathcal{B}$
      (Algorithm 7).
    Step 3: Sample $\min\{|\mathcal{B}|, m - |\mathcal{B}|\}$ blocks from $\mathcal{B}$, with replacement,
      according to $\Pr(\cdot)$ to determine a subset of blocks $\mathcal{A} \subseteq \mathcal{B}$.
    Step 4: Split each $B \in \mathcal{A}$ and update $\mathcal{B}$.
end
Step 5: Construct $\mathcal{P} = \mathcal{B}(X)$.
return $\mathcal{B}$ and $\mathcal{P}$.
```

In `Step 1`, a partition of the smallest bounding box containing the data set X, B_D, of size $m' > K$ is obtained by splitting recursively the blocks according to the pseudo-code shown in Algorithm 6. Once we have the spatial partition of size m', we iteratively produce thinner partitions of the space as long as the number of blocks is lower than m. At each iteration, the process is divided into three steps: In `Step 2`, we estimate the cutting probability $Pr(B)$ for each block B in the current space partition $\mathcal{B}$ using Algorithm 7. Then, in `Step 3`, we randomly sample (with replacement) $\min\{|\mathcal{B}|, m - |\mathcal{B}|\}$ blocks from $\mathcal{B}$ according to $Pr(\cdot)$ to construct the subset of blocks $\mathcal{A} \subseteq \mathcal{B}$, i.e., $|\mathcal{A}| \leq \min\{|\mathcal{B}|, m - |\mathcal{B}|\}$. Afterwards, each of the selected blocks in $\mathcal{A}$ is replaced by two smaller blocks obtained by splitting B in the middle point of its longest side. Finally, the obtained spatial partition $\mathcal{B}$ and the induced data set partition $\mathcal{B}(X)$ (of size lower or equal to m) are returned.

Algorithm 6: Step 1 of Algorithm 5

```
Input: Dataset $X$, partition size $m' > K$, sample size $s < n$.
Output: A spatial partition of size $m'$, $\mathcal{B}$.
- Set $\mathcal{B} = \{B_X\}$.
while $|\mathcal{B}| < m'$ do
    - Take a random sampling of size $s$, $S \subset X$ .
    - Obtain a subset of blocks, $\mathcal{A} \subseteq \mathcal{B}$, by sampling, with replacement,
      $\min\{|\mathcal{B}|, m' - |\mathcal{B}|\}$ blocks according to a probability proportional
      to $l_B \cdot |B(S)|$, for each $B \in \mathcal{B}$.
    - Split the selected blocks $\mathcal{A}$ and update $\mathcal{B}$.
end
return $\mathcal{B}$.
```

Algorithm 6 generates the starting spatial partition of size m' of the data set X. This procedure recursively obtains thinner partitions by splitting a subset of up to $\min\{|\mathcal{B}|, m' - |\mathcal{B}|\}$ blocks selected by a random sampling with replacement according to a probability proportional to the product of the diagonal of the block B, l_B, by its weight, $|B(S)|$. At this step, as we can not estimate how likely it is for a given block to be well assigned with respect to a set of K representatives, the goal is to control both weight and size of the generated spatial partition, i.e., to reduce the possible number of cluster misassignments, as this cutting procedure prioritizes those blocks that might be large and dense. Ultimately, as we reduce this factor, we improve the accuracy of our weighted approximation- see Theorem 3.

This process is repeated until a spatial partition with the desired number of blocks, $m' \geq K$, is obtained. Such a partition is later used to determine the sets of centroids which we use to verify how likely it is for a certain block to be well assigned.

In Algorithm 7, we show the pseudo-code used in `Step 2` of Algorithm 5 for computing the cutting probabilities associated to each block $B \in \mathcal{B}$, $Pr(B)$. Such a probability function depends on the misassignment function associated to each block with respect to multiple K-means++ based set of centroids. To generate these sets of centroids, r subsamples of size s, with replacement, are extracted from the data set, X. In particular, the **cutting probabilities** are expressed as follows:

$$\Pr(B) = \frac{\sum_{i=1}^{r} \epsilon_{C^i,S^i}(B)}{\sum_{B' \in \mathcal{B}} \sum_{i=1}^{r} \epsilon_{C^i,S^i}(B')} \tag{4.3}$$

for each $B \in \mathcal{B}$, where S^i is the subset of points sampled and C^i is the set of K centroids obtained via K-means++ for $i = 1, ..., r$. As we commented before, the larger the misassignment function is, then the more likely it is for the corresponding block to contain instances that belong to different clusters. It should be highlighted that a block B with $Pr(B) = 0$ is well assigned for all S^i and C^i, with $i = 1, .., r$.

Algorithm 7: Step 2 of Algorithm 5

```
Input: A spatial partition $\mathcal{B}$ of size higher than $K$, data set $X$, number
of clusters $K$, sample size $s$, number of repetitions $r$.
Output: Cutting probability $\Pr(B)$ for each $B \in \mathcal{B}$.
for $i = 1, \ldots, r$ do
    -Take subsample $S^i \subseteq X$ of size $s$ and construct $\mathcal{P} = \mathcal{B}(S^i)$.
    -Obtain a set of centroids $C^i$ by applying $K$-means++ over the
     representatives of $\mathcal{P}$.
    - Compute $\epsilon_{S^i,C^i}(B)$ for all $B \in \mathcal{B}$ (Eq. 4.1).
end
Step 4: Compute $Pr(B)$ for every $B \in \mathcal{B}$, using $\epsilon_{S^i,C^i}(B)$ for $i = 1, .., r$
 (Eq. 4.3).
return $\Pr(\cdot)$.
```

Even when cheaper seeding procedures, such as a Forgy type initialization, could be used, K-means ++ avoids cluster oversampling, and so one would expect the corresponding boundaries not to divide subsets of points that are supposed to have the same cluster affiliation. Additionally, as previously commented, this initialization also tends to lead to competitive solutions. Later on, in Section 4.1.4.1, we will comment on the selection of the different parameters, used in the initialization (m, m', r and s).

4.1.3 Construction of the sequence of thinner partitions

In this section, we provide the pseudo-code of the BWKM algorithm and introduce a detailed description of the construction of the sequence of thinner partitions, which is the basis of BWKM. In general, once the initial partition is constructed via algorithm 5, BWKM progresses iteratively by alternating i) a run of weighted Lloyd's algorithm over the current partition and ii) the creation of a thinner partition using the information provided by the weighted Lloyd's algorithm. The pseudo-code of the BWKM algorithm can be seen in Algorithm 8.

In `Step 1`, the initial spatial partition $\mathcal{B}$ and the induced data set partition, $\mathcal{P} = \mathcal{B}(X)$, are generated via Algorithm 5. Then, the initial set of centroids is obtained through a weighted version of K-means++ over the set of representatives of $\mathcal{P}$.

Given the current set of centroids C and the partition of the data set $\mathcal{P}$, the set of centroids is updated in `Step 2` and `Step 4` by applying the weighted Lloyd's algorithm. It must be commented that the only difference between these two tasks is the fact that `Step 2` is initialized with a set of centroids obtained via weighted K-means++ run, while `Step 4` utilizes the set of centroids generated by the weighted Lloyd's algorithm over the previous data set partition. In addition, in order to compute the misassignment function $\epsilon_{C,X}(B)$ for all $B \in \mathcal{B}$ in `Step 3` (see Eq.4.1), we store the following information provided by the last iteration of the weighted Lloyd's algorithm: for each $P \in \mathcal{P}$, the two closest centroids to the representative $\overline{P}$ in C are saved (see Figure 4.1).

In `Step 3`, a spatial partition thinner than $\mathcal{B}$ and its induced data set partition are generated. For this purpose, the misassignment function, $\epsilon_{C,X}(B)$ for all $B \in \mathcal{B}$ is computed and the boundary $\mathcal{F}_{C,X}(\mathcal{B})$ is determined using the information stored at the last iteration of `Step 2`. Next, as the misassignment criterion in Theorem 2 is just a sufficient condition, instead of dividing all the blocks that do not satisfy it, we prioritize those blocks that are less likely to be well assigned: A set $\mathcal{A}$ of blocks is selected by sampling with replacement $|\mathcal{F}_{C,X}(\mathcal{B})|$ blocks from $\mathcal{B}$ with a (cutting) probability proportional to $\epsilon_{C,X}(B)$. Note that the size of $\mathcal{A}$ is at most $|\mathcal{F}_{C,X}(\mathcal{B})|$. Afterwards, in order to reduce as much as possible the length of the diagonal of the newly generated blocks and control the size of the thinner partition, each block in $\mathcal{A}$ is divided in the middle point of its largest side. Each block is split once into two equally shaped hyper-rectangles and it is replaced in $\mathcal{B}$ to produce the new thinner spatial partition. Finally, given the new spatial partition $\mathcal{B}$, its induced data set partition is obtained $\mathcal{P} = \mathcal{B}(X)$.

Algorithm 8: BWKM Algorithm

```
Input: Dataset $X$, number of clusters $K$ and initialization parameters
  $m'$, $m$, $s$, $r$.
Output: Set of centroids $C$.
Step 1: Initialize $\mathcal{B}$ and $\mathcal{P}$ via Algorithm 5, with input $m'$, $m$, $s$, $r$,
  and obtain $C$ by applying a weighted $K$-means++ run over the set of
  representatives of $\mathcal{P}$.
Step 2: $C = \texttt{WeightedLloyd}(\mathcal{P}, C, K)$.
while not Stopping Criterion do
    Step 3: Update data set partition $\mathcal{P}$:
    - Compute $\epsilon_{C,X}(B)$ for all $B \in \mathcal{B}$.
    - Select $\mathcal{A} \subseteq \mathcal{F}_{C,X}(\mathcal{B}) \subseteq \mathcal{B}$ by sampling, with replacement,
      $|\mathcal{F}_{C,X}(\mathcal{B})|$ blocks according to $\epsilon_{C,X}(B)$, for all $B \in \mathcal{B}$.
    - Cut each block in $\mathcal{A}$ and update $\mathcal{B}$ and $\mathcal{P}$.
    Step 4: $C = \texttt{WeightedLloyd}(\mathcal{P}, C, K)$.
end
return $C$
```

It should be noted that the cutting criterion, Eq.4.1, is more accurate, i.e., it detects more well assigned blocks, as long as we evaluate it over the smallest bounding box of each block of the spatial partition, since we minimize the maximum distance (diagonal) between any two points in the block. Therefore, when updating the data partition in `Step 3`, we also recompute the diagonal of the smallest bounding box of each subset.

`Step 2` and `Step 3` are then repeated until a certain stopping criterion is satisfied (for details on different stopping criteria, see Section 4.1.4.2).

4.1.3.1 Computational complexity of the BWKM algorithm

In this section, we provide the computational complexity of each step of BWKM, in the worst case.

The construction of the initial spatial partition, the corresponding induced data set partition and the set of centroids of BWKM (`Step 1`) has the following computational cost: $\mathcal{O}(\max\{r \cdot s \cdot m^2,\ r \cdot K \cdot d \cdot m^2,\ \mathcal{O}(n \cdot \max\{m,\ d\})\})$. Each of the previous terms corresponds to the complexity of `Step 1`, `Step 2` and `Step 5` in Algorithm 5, respectively, which are the most computationally demanding procedures of the initialization. Even when these costs are deduced from the worst possible scenario, which is overwhelmingly improbable, in Section 4.1.4.1, we will comment on the selection of the initialization parameters in such a way that the cost of this step is not more expensive than that of the K-means algorithm.

As we previously mentioned `Step 2` of Algorithm 8 (the weighted Lloyd's algorithm) has a computational complexity of $\mathcal{O}(|\mathcal{P}| \cdot K \cdot d)$. In addition, `Step 3` executes $\mathcal{O}(|\mathcal{P}| \cdot K)$ computations to verify the cutting criterion, since all the distance computations are obtained from the previous weighted Lloyd iteration. Moreover, assigning each instance to its corresponding block and updating the bounding box for each subset of the partition is $\mathcal{O}(n \cdot d)$. In summary, since $|\mathcal{P}| \leq n$, then BWKM algorithm has an overall cost of $\mathcal{O}(n \cdot K \cdot d)$ in the worst case.

4.1.4 Additional Remarks

In this section, we discuss additional features of the BWKM algorithm, such as the selection of the initialization parameters for BWKM, we also comment on different possible stopping criteria, with their corresponding computational costs and theoretical guarantees.

4.1.4.1 Parameter selection

The construction of the initial space partition and the corresponding induced data set partition of BWKM (see Algorithm 5 and `Step 1` of Algorithm 8) depends on the parameters m, m', r, s, K and X, while the core of BWKM (`Step 2` and `Step 3`) only depends on K and X. In this section, we propose how to select the parameters m, m', r and s, keeping in mind the following objectives: i) to guarantee BWKM having a computational complexity equal to or lower than $\mathcal{O}(n \cdot K \cdot d)$, which corresponds to the cost of Lloyd's algorithm, and ii) to obtain an initial spatial partition with a large amount of well assigned blocks.

In order to ensure that the computational complexity of BWKM's initialization is, even in the worst case, $\mathcal{O}(n \cdot K \cdot d)$, we must take m, m', r and s such that $r \cdot s \cdot m^2$, $r \cdot m^2 \cdot K \cdot d$ and $n \cdot m$ are $\mathcal{O}(n \cdot K \cdot d)$. On the other hand, as we want such an initial partition to minimize the number of blocks that may not be well assigned, we must consider the following facts: i) the larger the diagonal for a certain block $B \in \mathcal{B}$ is, then the more likely it is for B not to be well assigned, ii) as the number of clusters K increases, then any block $B \in \mathcal{B}$ has more chances of containing instances with different cluster affiliations, and iii) as s increases, the cutting probabilities become better indicators for detecting those blocks that are not well assigned.

Taking into consideration these observations, and assuming that r is a predefined small integer, satisfying $r \ll n/s$, we propose the use of $m = \mathcal{O}(\sqrt{K \cdot d})$ and $s = \mathcal{O}(\sqrt{n})$. Not only does such a choice satisfy the complexity constraints that we just mentioned (See Theorem 11 in Appendix .2.1), but also, in this case, the size of the initial partition increases with respect to both dimensionality of the problem and number of clusters: Since at each iteration, we divide a block only on one of its sides, then, as we increase the dimensionality, we need more cuts (number of blocks) to have a sufficient reduction of its diagonal (observation i)). Analogously, the number of blocks and the size of the sampling increases with respect to the number of clusters and the actual size of the data set, respectively (observation ii) and iii)). In particular, in the experimental section, Section 4.2, we used $m = 10 \cdot \sqrt{K \cdot d}$, $s = \sqrt{n}$ and $r = 5$.

4.1.4.2 Stopping Criterion

As we discussed earlier, one of the advantages of constructing spatial partitions with only well assigned blocks is that our algorithm, under this setting, converges to a local minima of the K-means problem over the entire data set and, therefore, there is no need to execute any further run of the BWKM algorithm as the set of centroids will remain the same for any thinner partition:

Theorem 4 *If C is a fixed point of the weighted K-means algorithm for a spatial partition $\mathcal{B}$, for which all of its blocks are well assigned, then C is a fixed point of the K-means algorithm on X.* [9]

To verify this criterion, we can make use of the concept of boundary of a spatial partition (Definition 9). In particular, observe that if $\mathcal{F}_{C,X}(\mathcal{B}) = \emptyset$, then one can guarantee that all the blocks of $\mathcal{B}$ are well assigned with respect to

[9] The proof of Theorem 4 is in Appendix .2.1 in the supplementary material.

both C and X. To check this, we just need to scan the misassignment function value for each block, i.e., it is just $\mathcal{O}(|\mathcal{P}|)$. In addition to this criterion, in this section we will propose three other stopping criteria:

- *A practical computational criterion*: We could set, in advance, the amount of computational resources that we are willing to use and stop when we exceed them. In particular, as the computation of distances is the most expensive step of the algorithm, we could set a maximum number of distances as a stopping criterion.
- *A Lloyd's algorithm type criterion*: As we mentioned in Section 1.1.1.2, the common practice is to run Lloyd's algorithm until the reduction of the error, after a certain iteration, is small. As in our weighted approximation we do not have access to the error $E_X(C)$, a similar approach is to stop the algorithm when the obtained set of centroids, in consecutive iterations, is smaller than a fixed threshold, ε_w. We can actually set this threshold in a way that the stopping criterion of Lloyd's algorithm is satisfied. For instance, for $\varepsilon_w = \sqrt{l^2 + \frac{\varepsilon^2}{n^2}} - l$, if $\|C - C'\|_\infty \leq \varepsilon_w$, then $|E_X(C) - E_X(C')| \leq \varepsilon$, holds [10]. However, this would imply additional $\mathcal{O}(K \cdot d)$ computations at each iteration.
- *A criterion based on the accuracy of the weighted error*: We could also consider the bound obtained at Theorem 3 and stop when it is lower than a predefined threshold. This will let us know how accurate our current weighted error is with respect to the error over the entire data set. All the information in this bound is obtained from the weighted Lloyd iteration and the information of the block and its computation is just $\mathcal{O}(|\mathcal{P}|)$.

4.2 Experiments

In this section, we perform a set of experiments so as to analyze the relation between the number of distances computed and the quality of the approximation for the **BWKM** algorithm proposed in Section 4.1. In particular, we compare the performance of BWKM with respect to different methods known for the quality of their approximations[11]: Lloyd's algorithm initialized via i) Forgy (**FKM**) and ii) K-means++ (**KM++**) [12]. We also consider the Minibatch K-means, with batches $b = \{100, 500, 1000\}$ [13] (**MB b**), which is particularly known for its efficiency due to the small amount of resources needed to generate its approximation, as well as the Markov chain Monte Carlo sampling based approximation of the K-means++ (**AFKMC2**) with a further MB 100 run as recommended by its authors [43].

[10] See Theorem 12 in Appendix .2.1 in the supplementary material

[11] Additionally, in Appendix .2.2, we comment on the grid based RPKM.

[12] The output of such an initialization is presented as **KM++_init**.

[13] Similar values were used in the original paper [30].

To have a better understanding of BWKM, we analyze its performance on a wide variety of well known real data sets (see Table 4.1) with different scenarios of the clustering problem. For each data set, we have considered a different number of clusters, $K = \{3, 5, 10, 25, 50\}$. Given the random nature of the algorithms, each experiment has been repeated 40 times for each data set and each K value.

Dataset	n	d
Corel Image Features (CIF)	$68,037$	17
3D Road Network (3RN)	$434,874$	3
Household Power Consumption (HPC)	$2,049,280$	7
Gas Sensor (GS)	$4,208,259$	19
SUSY	$5,000,000$	19
Web Users Yahoo! (WUY)	$45,811,883$	5

Table 4.1: Information of the data sets.

As stopping criterion, we have fixed the maximum number of BWKM iterations to 100. However it might converge before if the corresponding boundary is empty, in which case, we can guarantee that the obtained set of centroids is a fixed point of the weighted Lloyd's algorithm for any thinner partition of the data set, therefore, it is also a fixed point of Lloyd's algorithm on the entire data set X (see Theorem 4). Moreover, in order to compare the performance of the algorithms for different settings of the clustering problem, we decided to use the average of the relative error with respect to the best solution found at each repetition of the experiment, i.e., $\hat{E}_M = \frac{E_M - \min\limits_{M' \in \mathcal{M}} E_{M'}}{\min\limits_{M' \in \mathcal{M}} E_{M'}}$, where $\mathcal{M}$ is the set of algorithms being compared and E_M stands for the K-means error obtained by method $M \in \mathcal{M}$. Analogously, in terms of the amount of computational resources required, we show the proportion of distances computed by each method with respect to the method that computed the largest number of distances, i.e., $\hat{DC}_M = \frac{DC_M}{\max\limits_{M' \in \mathcal{M}} DC_{M'}}$, where DC_M is the number of distances computed by $M \in \mathcal{M}$.

In Fig. 4.2-4.7, we show the trade-off between the average relative number of distances computed *vs* the average relative error for all the algorithms. Observe that a single symbol is used for each algorithm, except for BWKM, in which we compute the trade-off at each iteration so as to observe the evolution of the quality of its approximation as the number of computed distances increases.

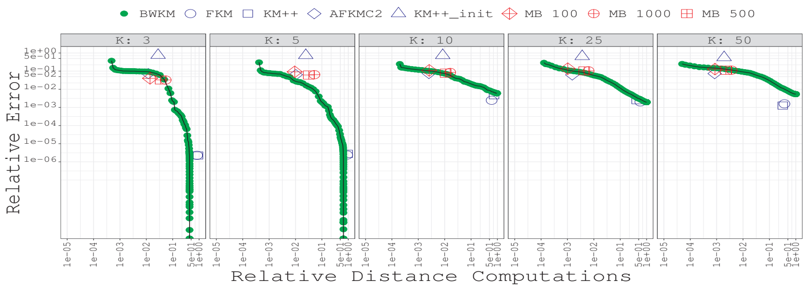

Fig. 4.2: Relative distance computations *vs* relative error on the *CIF* data set.

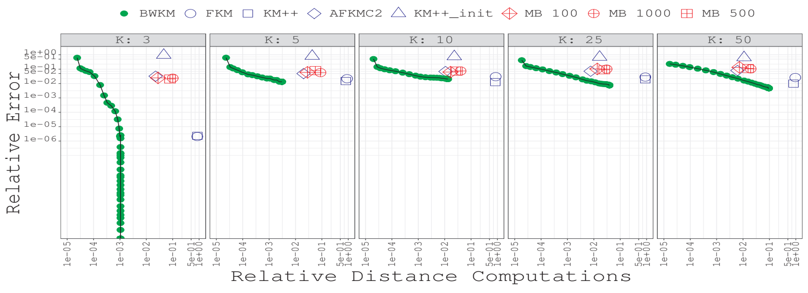

Fig. 4.3: Relative distance computations *vs* relative error on the *3RN* data set.

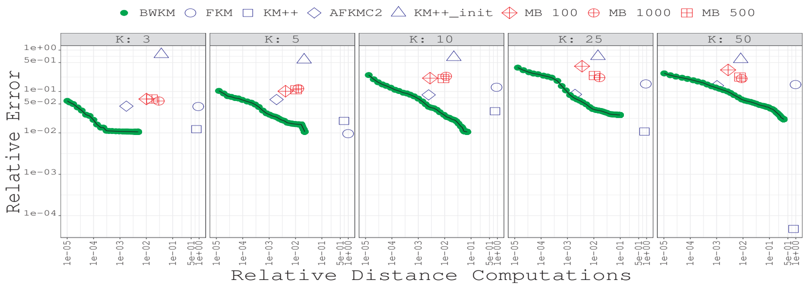

Fig. 4.4: Relative distance computations *vs* relative error on the *HPC* data set.

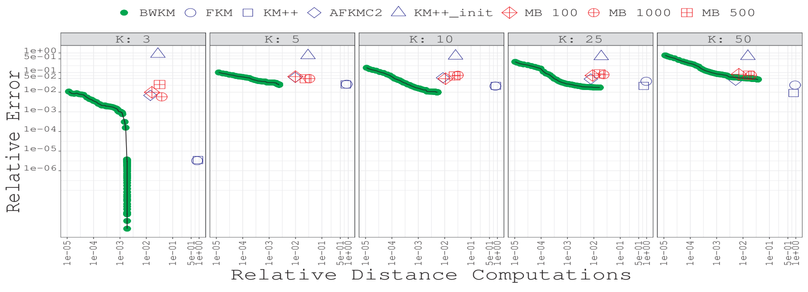

Fig. 4.5: Relative distance computations *vs* relative error on the *GS* data set.

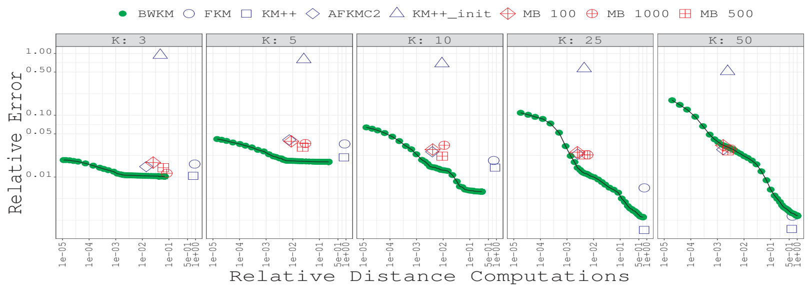

Fig. 4.6: Relative distance computations *vs* relative error on the *SUSY* data set.

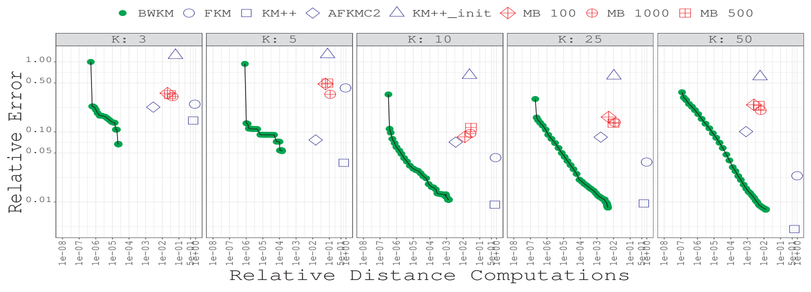

Fig. 4.7: Relative distance computations *vs* relative error on the *WUY* data set.

At first glance, we observe that, in 18 out of 35 different configurations of data sets and K values, BWKM obtained the best (average) solution among the considered methods. Furthermore, in Table .1 we observe that BWKM quite frequently (in 206 out of 210 cases) converged to sets of centroids that reached on average, at least, 1% of error with respect to all the considered methods and clustering configurations. If we increase such a threshold to 5%, BWKM reaches it in every case. It must be highlighted that such clusterings were generated while computing a massively reduced number of distances: Up to 5 and 7 orders of magnitude of distances less than the *Minibatch based methods* (MB 100, MB 500, MB 1000 and AFKMC2) and the *Lloyd's based methods* (FKM and KM++), respectively. In particular and as expected, the best performance of BWKM seems to occur on large data sets with small dimensions (*WUY*). On one hand, the decrease in the amount of distances computed is mainly due to the reduction in the number of representatives that BWKM uses in comparison to the actual size of the data set. On the other hand, given a set of points as the dimension decreases, the number of blocks required to obtain a partition completely well assigned tends to decrease (*WUY* and *3RN*).

Method	K	*CIF*	*3RN*	*HPC*	*GS*	*SUSY*	*WUY*
	3	5.6(−2); 9.5(−2)	1.2(−4); 2.8(−4)	1.3(−5); 1.8(−5)	8.6(−6); 1.7(−5)	1.1(−5); 1.1(−5)	7.1(−7); 1.0(−6)
	5	6.7(−3); 6.5(−2)	1.0(−4); 3.8(−4)	1.6(−4); 2.9(−3)	7.8(−5); 1.0(−3)	7.0(−6); 7.0(−6)	9.3(−7); 9.3(−7)
FKM	10	4.3(−2); 6.8(−1)	1.5(−4); 8.8(−4)	3.0(−5); 6.4(−5)	1.3(−4); 4.1(−4)	1.1(−5); 1.2(−3)	8.2(−7); 3.0(−6)
	25	3.3(−2); 1.9(−1)	2.3(−4); 1.3(−3)	4.5(−4); 6.5(−4)	2.0(−4); 4.1(−4)	1.1(−3); 3.9(−3)	1.7(−6); 7.2(−6)
	50	1.5(−1); 9.4(−1)	7.7(−4); 4.0(−3)	1.2(−4); 4.3(−4)	2.6(−3); ∗	1.1(−3); 1.0(−1)	3.4(−5); 2.0(−4)
	3	5.3(−2); 8.9(−2)	1.1(−4); 2.8(−4)	1.5(−5); 1.2(−4)	7.7(−6); 1.6(−5)	1.3(−5); 1.3(−5)	1.6(−6); 1.1(−5)
	5	6.3(−3); 6.5(−2)	1.8(−4); 9.4(−4)	9.7(−5); 1.5(−3)	5.3(−5); 5.6(−4)	7.4(−6); 5.7(−4)	1.3(−6); 1.3(−6)
KM++	10	2.6(−2); 2.9(−1)	5.7(−4); 1.2(−2)	6.0(−4); 3.1(−3)	1.1(−4); 4.2(−4)	2.3(−5); 1.8(−3)	1.6(−6); 7.2(−5)
	25	4.5(−2); 2.5(−1)	4.0(−4); 2.2(−3)	4.6(−3); ∗	3.4(−4); 7.6(−4)	1.1(−3); 1.2(−2)	1.0(−5); 1.8(−4)
	50	1.6(−1); 1.1(0)	1.6(−3); 1.3(−2)	5.2(−2); ∗	5.6(−3); ∗	1.3(−3); 1.1(−1)	1.2(−4); 2.1(−3)
	3	6.1(−2); 1.5(0)	3.0(−3); 7.2(−3)	7.9(−3); 7.9(−3)	4.9(−4); 8.0(−4)	6.7(−4); 6.7(−4)	2.2(−4); 2.2(−4)
	5	5.7(−2); 8.0(−2)	3.2(−3); 8.1(−3)	1.0(−2); 5.7(−2)	1.9(−3); 9.5(−3)	8.3(−4); 8.3(−4)	6.2(−5); 6.2(−5)
AFKMC2	10	5.5(−1); 4.0(0)	4.8(−3); 2.3(−2)	3.7(−2); 1.4(−1)	5.8(−3); 1.4(−2)	1.0(−3); 1.7(−1)	1.1(−4); 1.9(−4)
	25	1.1(0); 6.4(0)	8.3(−3); 2.6(−2)	4.6(−1); 6.6(−1)	2.0(−2); 4.2(−2)	2.0(−1); 5.6(−1)	2.0(−4); 6.2(−4)
	50	3.6(0); 2.3(1)	3.0(−2); 1.2(−1)	1.6(−1); 7.9(−1)	1.8(−1); 1.1(0)	1.8(−1); 6.6(−1)	3.5(−3); 9.8(−3)
	3	1.9(−2); 1.9(−2)	6.9(−4); 6.9(−4)	2.7(−4); 2.7(−4)	2.1(−4); 2.1(−4)	1.7(−4); 1.7(−4)	9.4(−6); 9.4(−6)
	5	1.9(−2); 1.9(−2)	7.7(−4); 7.7(−4)	3.3(−4); 3.3(−4)	2.6(−4); 2.6(−4)	2.2(−4); 2.2(−4)	1.2(−5); 1.2(−5)
KM++ init	10	2.6(−2); 2.6(−2)	1.2(−3); 1.2(−3)	4.4(−4); 4.4(−4)	3.4(−4); 3.4(−4)	3.0(−4); 3.0(−4)	1.5(−5); 1.5(−5)
	25	4.1(−2); 4.1(−2)	1.6(−3); 1.6(−3)	6.0(−4); 6.0(−4)	5.4(−4); 5.4(−4)	4.7(−4); 4.7(−4)	2.2(−4); 2.2(−4)
	50	3.4(−2); 3.4(−2)	1.5(−3); 1.5(−3)	7.4(−4); 7.4(−4)	9.2(−4); 1.1(−3)	6.5(−4); 6.5(−4)	2.5(−5); 2.5(−5)
	3	5.0(−1); 3.4(0)	2.4(−3); 5.8(−3)	1.0(−3); 1.0(−3)	4.4(−4); 7.2(−4)	3.7(−4); 3.7(−4)	2.3(−5); 2.3(−5)
	5	5.7(−2); 5.7(−2)	2.0(−3); 5.1(−3)	2.3(−3); 2.3(−3)	1.6(−3); 6.5(−3)	6.6(−4); 6.6(−4)	1.4(−5); 1.4(−5)
MB 100	10	1.3(−1); 7.2(−1)	3.2(−3); 1.5(−2)	9.1(−3); 1.3(−2)	5.1(−3); 1.2(−2)	7.9(−4); 9.6(−2)	3.0(−5); 3.9(−5)
	25	3.7(−1); 5.1(−1)	3.0(−3); 9.2(−3)	4.2(−3); 4.2(−3)	1.6(−2); 2.3(−2)	2.0(−1); 5.3(−1)	4.9(−5); 4.9(−5)
	50	2.4(−1); 2.2(0)	1.2(−2); 2.5(−2)	2.3(−3); 2.3(−3)	7.1(−2); 2.0(−1)	1.8(−1); 6.4(−1)	1.1(−4); 1.7(−4)
	3	5.0(−1); 1.2(0)	1.3(−3); 2.2(−3)	6.9(−4); 6.9(−4)	2.1(−4); 2.6(−4)	1.8(−4); 1.8(−4)	2.6(−5); 2.6(−5)
	5	1.5(−2); 5.0(−2)	7.3(−4); 1.6(−3)	7.4(−4); 1.0(−3)	9.7(−4); 7.5(−3)	2.7(−4); 6.2 − 4)	1.3(−5); 1.3(−5)
MB 500	10	9.0(−2); 6.6(−1)	1.9(−3); 4.8(−3)	1.6(−3); 2.4(−3)	1.5(−3); 4.5(−3)	6.4(−4); 6.6(−2)	1.5(−5); 1.5(−5)
	25	1.6(−2); 3.2(−1)	2.6(−3); 8.4(−3)	4.7(−3); 1.6(−2)	7.2(−3); 1.1(−2)	1.1(−1); 3.0(−1)	3.0(−5); 4.0(−5)
	50	1.4(−1); 1.2(0)	6.0(−3); 2.4(−2)	2.3(−3); 6.6(−3)	3.1(−2); 9.7(−2)	2.1(−1); 6.0(−1)	6.0(−5); 9.3(−5)
	3	5.2(−1); 1.3(0)	6.9(−4); 1.6(−3)	4.4(−4); 4.4(−4)	1.8(−4); 2.9(−4)	1.2(−4); 1.2(−4)	1.2(−5); 1.2(−5)
	5	1.0(−2); 3.1(−2)	7.1(−4); 1.1(−3)	5.9(−4); 8.5(−4)	5.5(−4); 4.0(−3)	2.1(−4); 2.1(−4)	8.7(−6); 8.7(−6)
MB 1000	10	3.4(−2); 2.5(−1)	1.3(−3); 3.3(−3)	1.0(−3); 1.6(−3)	1.1(−3); 2.1(−3)	2.8(−4); 2.0(−2)	1.5(−5); 1.9(−5)
	25	7.9(−2); 2.2(−1)	1.2(−3); 3.8(−3)	7.6(−3); 2.5(−2)	6.7(−3); 8.2(−3)	8.5(−2); 2.3(−1)	2.3(−5); 3.0(−5)
	50	8.9(−2); 7.6(−1)	4.7(−3); 1.9(−2)	3.0(−3); 8.4(−3)	2.4(−2); 7.7(−2)	1.6(−1); 4.6(−1)	7.1(−5); 1.7(−4)

Table 4.2: BWKM distance proportion with respect to the considered methods for reaching under $< 5\%; < 1\%$ of their relative error (In our notation, $5.6(-2) = 5.6 \times 10^{-2}$).

Regardless of this, even when considering the most unfavorable setting for BWKM (smallest data set size with large dimension, i.e., *CIF*), for small K values, our proposal still managed to converge to competitive solutions (under 1% of error when compared to the competition) at a fast rate (reducing on average up to 2 orders of distance computations). Note that for small K values, since the number of centroids is small, one may not need to reduce the diagonal of the blocks so abruptly to verify the well assignment criterion. On the other hand, for the largest numbers of clusters, BWKM still generated solutions of the same quality but struggled to reduce the number of computations: BWKM computed the same order of distances as the Minibatch based methods to generate approximations with a similar error.

In the case of small data sets with low dimensionality (*3RN*), BWKM performs much better in comparison to the previous case: In 4 out of 5 values of K, BWKM actually generates the most competitive solutions. In particular, in order to achieve a relative error of under 1% of error, BWKM reduces between

1 to 3 orders of magnitude of distances with respect to the Minibatch based methods, and 2 to 4 orders of magnitude against the Lloyd's based methods. Furthermore, for the medium size data sets with low dimensionality (*HPC*), BWKM has a similar performance, leading on average to the most qualitative solution in 3 out of 5 values of K, while reducing between 1 to 4 orders of magnitude of distances with respect to the Minibatch based methods, and 2 to 5 orders of magnitude against the Lloyd's based methods to generate solutions under 5% of their error.

If we consider the case of the medium to large data sets with larger dimensionality (*GS* and *SUSY*), in order to reach a 5% relative error, BWKM decreases between 1 to 4 orders of magnitude with respect to the Minibatch based methods and 3 to 6 orders in comparison to the Lloyd's based methods. Moreover, BWKM obtains the solutions with the lowest errors in 5 out of 10 configurations.

For the largest data set (*WUY*), BWKM got its best performance. Again, it usually generated the most competitive solutions (in 4 out of 5 cases), however, in this case, as expected BWKM computes an amount of distance from 4 to 6 and 3 to 7 orders of magnitude lower than the Minibatch and the Lloyd's based algorithms to reach a relative error of under 1% with respect to them, respectively.

Finally, we would like to highlight that BWKM, already at its first iterations, reaches a relative error much lower than KM++_init in all the configurations requiring to compute an amount of distances from 2 to 6 order of magnitude lower. This fact strongly motivates the use of BWKM as a competitive initialization strategy for Lloyd's algorithm.

Undoubtedly BWKM achieves its best results in terms of the trade-off between number of distance computations and the quality of the solution obtained, when dealing with large data sets with small dimensions (*tall data*), therefore its use is mostly recommended for this setting. In any case and in spite of considered configuration, BWKM shows a quite competitive performance when compared to the state-of-the-art, leading to reductions of several orders of distance computations while converging to sets of centroids with a similar and/or lower error than the one achieved by the considered algorithms. Furthermore, it must be highlighted that the modifications made in the cutting criterion allowed our algorithm to scale to dimensions that were intractable for the previous grid based RPKM in Chapter 3.

4.3 Conclusions

In this chapter, we have presented an alternative to the K-means algorithm, oriented to massive data problems, called the Boundary Weighted K-means algorithm (BWKM). This approach recursively applies a weighted version of the K-means algorithm over a sequence of spatial based partitions of the data set that ideally contains a large amount of *well assigned blocks*, i.e., cells of the spatial partition that only contain instances with the same cluster affiliation. It can be shown that our weighted error approximates the K-means error function, as we increase the number of well assigned blocks, see Theorem 3. Ultimately, if all the blocks of a spatial partition are well assigned at the end of a BWKM step, then the obtained clustering is actually a fixed point of the K-means algorithm, which is generated after using only a small number of representatives in comparison to the actual size of the data set (Theorem 4). Furthermore, if, for a certain step of BWKM, this property can be verified at consecutive weighted Lloyd's iterations, then the error of our approximation also decreases monotonically (Theorem 10).

In order to achieve this, in Section 4.1.1, we designed a criterion to determine those blocks that may not be well assigned. One of the major advantages of the criterion is its low computational cost: It only uses information generated by the weighted K-means algorithm -distances between the center of mass of each block and the set of centroids- and a feature of the corresponding spatial partition -diagonal length of each block-. This allows us to guarantee that, even in the worst possible case, BWKM does not have a computational cost higher than that of the K-means algorithm. In particular, the criterion is presented in Theorem 2 and states that, if the diagonal of a certain block is smaller than half the difference of the two the smallest distances between its center of mass and the set of centroids, then the block is well assigned.

In addition to all the theoretical guarantees that motivated and justify our algorithm (see Section 4.1 and Appendix .2.1), in practice, we have also observed its competitiveness with respect to the state-of-the-art (Section 4.2). BWKM has been compared to Lloyd's algorithm initialized with Forgy's approach and K-means++, the AFKMC2 algorithm and the Minibatch K-means.

The results, on different well known real data sets, show that BWKM in several cases (18 out of 35 configurations) has generated the most competitive solutions. Furthermore, in 206 out of 210 cases, BWKM has converged to solutions with a relative error of under 1% with respect to the considered methods, while using a much smaller amount of distance computations (up to 7 orders of magnitude lower).

As for the next steps, we plan to exploit different benefits of BWKM. First of all, observe that the proposed algorithm is embarrassingly parallel up to the K-means++ seeding of the initial partition (over a very tiny amount of representatives when compared to the data set size), hence we could implement this approach in a more appropriate platform for this kind of problems, as is the case of *Apache Spark*. Moreover, we must point out that BWKM is also compatible with the distance pruning techniques presented in [24, 25, 26, 28], therefore, we could also implement these techniques within the weighted Lloyd framework of BWKM and reduce, even more, the number of distance computations.

5

A cheap feature selection approach for the *K*-means algorithm

The increase in the number of features that can be perceived in a wide variety of areas, such as genome sequencing, computer vision and sensor networks, represents a challenge for the K-means algorithm due to the curse of dimensionality. On the other hand, even when the results presented in Chapter 4 are very competitive and show the advantages of using BWKM for data sets with a large number of in- stances, the quality guarantees of BWKM are still dependant on the diagonal length of the cells in the spatial partition. Since the partition strategy introduced in BWKM divides the selected blocks along a single dimension, it will take more BWKM iterations to generate partitions with mostly well assigned blocks, as the dimensionality of the data set increases. This problem not only affects the quality of the approximation, but also increases the number of distance computations, since the number of representatives is also higher, see Section .2.2. In this regard, different dimensionality reduction approaches for the K-means algorithm have been recently designed, leading to algorithms that have proved to generate competitive clusterings, see Section 1.1.1.3. Unfortunately, most of these techniques tend to have very high computational costs and may not be easy to parallelize [64, 69]. In this chapter, we address the problem of dimensionality reduction for the K-means problem via a feature selection-type approach. Our goal is to create a low cost, fully-parallel algorithm that is able to keep the quality of the clustering structure for large dimensional datasets. We additionally provide theoretical guarantees for the K-means error obtained by our proposal.

A first step in this direction is the design of a measure that allows us to score the importance of a given variable with respect to the corresponding K-means problem. In particular, given an approximation to the K-means problem (with the corresponding clustering and prototypes), the propose measure consists of evaluating the effect of eliminating a variable (fixing the associated entry of the cluster prototypes to a given value) on the obtained clustering and the K-means error increase that such an action carries on.

In Fig 5.1, we can observe an intuition of the proposed idea on a 2D mixture of Gaussians and 2 clusters. In the first figure, we show, in different colors, a clustering, $\mathcal{P}_i$, and its associated centers of mass (black dots), C_i. Moreover, in the last two figures, we observe the variation on the clusterings that takes place when the centers of mass, in C_i, are fixed to a given value in either dimension. In this case, it is clear that dimension 2 (y-axis) provides less information of the obtained clustering structure, since, when fixed, the clustering remains invariant with respect to the original one. Therefore, dimension 1 (x-axis) is a better candidate to be selected. Rather than analyzing the clustering re-assignments, in the practice, we evaluate the error increase with respect to the original clustering and select those variables with the largest scores.

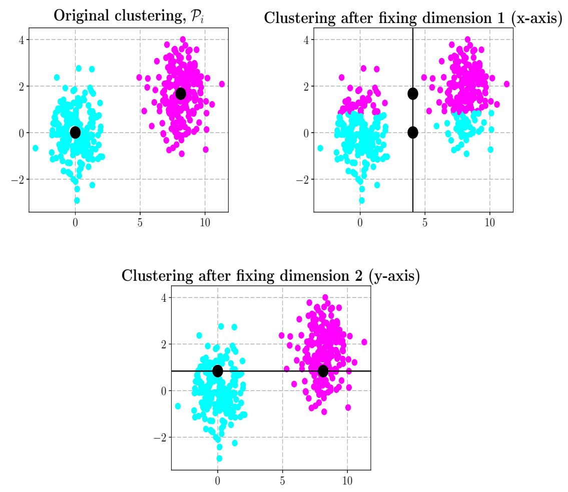

Fig. 5.1: Illustration on the proposed feature selecion rule, $K = 2$.

As we will explain more detailedly afterwards, such a feature selection rule will be used in parallel on different groups of dimensions. In particular, in Fig.5.2, we provide a sketch of the proposed algorithm for the K-means problem.

As shown in Fig.5.2, our approach mainly consists of three steps. First, given the number of features that we wish to select, m, the set of dimensions $\{1, \ldots, d\}$ is divided into the smallest number of chunks possible, $\{D_1, \ldots, D_t\}$, so that all parties have similar cardinalities and are upper-bounded by m. Observe that, for the first requirement, the lowest number of parties needed is given by $t = \left\lceil \frac{d}{m} \right\rceil$. Moreover, for the latter requirement, it is always possible to find a partition of the dimension set, for which

$\max_{i\in\{1,\ldots,t\}} |D_i| - \min_{i\in\{1,\ldots,t\}} |D_i| \leq 1$ [1]. The partition of the data set is then given by $\{X_{D_1},\ldots,X_{D_t}\}$, where X_{D_i} stands for extracting the columns in D_i of X. We refer to this process as `Split step`.

Afterwards, we make use of the variable importance score that we previously described. In particular, a λ-approximate K-means algorithm (algorithm $\mathcal{A}$) is applied, in parallel, over each chunk X_{D_i} and, as commented before, the obtained clustering is used to score the importance of each feature in X_{D_i}, by approximating the error increment that would happen if a given feature is eliminated (`Local approximation step`). After submitting the score of each variable to a central server, the m variables that seem to affect the clustering quality the most, are selected, and algorithm $\mathcal{A}$ is applied over them to obtain our clustering approximation (`Global approximation step`). As we will discuss in Section 5.1.1, this process leads to different theoretical guarantees in terms of the quality of the obtained solution.

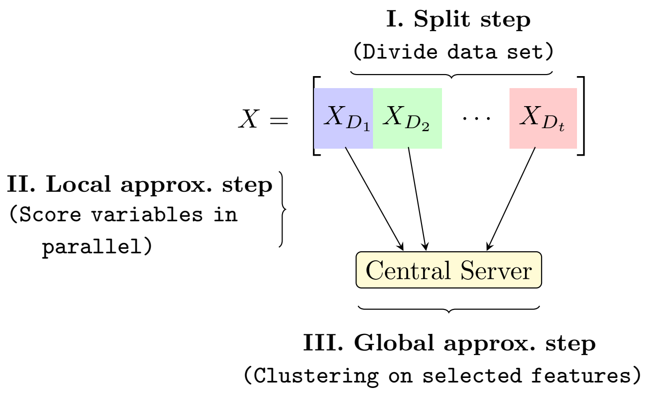

Fig. 5.2: Sketch of the proposal.

This proposal is partially motivated to the existance of different competitive approximations to the K-means algorithm, such as [78, 79, 59, 62], that do not scale well on the dimensionality of the problem. This approach would then allow the use of these techniques to approximate the solution of the clustering problem on subsets with a tractable number of dimensions. It must be pointed out that there exists other approaches, such as [80], in which the dimensions are also partitioned across multiple machines. However, in [80], the clustering information obtained, after executing K-means algorithm on each partition, is used to construct a coreset of K^t instances, rather than to perform a dimensionality reduction. On the other hand, we must remark that other feature

[1] For instance, if we set $f = m\cdot t - d$, then we can take $f - \lfloor \frac{f}{t} \rfloor \cdot t$ parties, with $m - \lfloor \frac{f}{t} \rfloor - 1$ dimensions, and, the remaining $t - f + \lfloor \frac{f}{t} \rfloor \cdot t$ parties, with $m - \lfloor \frac{f}{t} \rfloor$ dimensions.

selection/extraction- distributed techniques, such as [81, 70], splits the dataset X along its instances (rows) rather than along its dimensions (columns).

The rest of this article is organized as follows: In Section 5.1, we describe in detail the proposed algorithm and discuss some of its properties. In Section 5.2, we analyze the performance of the proposed algorithm with respect to different feature selection/extraction techniques, in terms of the clustering quality and the computational time. Finally, in Section 5.3, we define the next steps and possible improvements to our current work.

5.1 *K*-means relevance for feature selection

In this section, we formally describe our proposal, the *K-means relevance for feature selection* algorithm, or just K**MR**. As we previously commented, KMR is an alternative to the classical K-means algorithm, that uses the clustering information obtained on different subsets of dimensions, to discard those features that affect the least the clustering structure of the original dataset, when not considered. Each step of KMR is constructed in such a way that the heuristic used to obtain the clustering, algorithm $\mathcal{A}$, is always applied on subsets of dimensions upper-bounded by m. As we commented before, this last characteristic might be of special interest for different heuristics for the K-means problem, that are quite competitive on low dimensionalities, but that do not scale well on this factor, see [78, 79, 59, 62].

As we already commented, the initial phase of KMR cosists of partitioning the data set along its dimensions into the lowest number of regularly-sized groups that are upper-bounded by m (`Split step`). Such a partition is then used to perform the feature selection (`Local approximation step`) and the respective clustering approximation over the selected variables (`Global approximation step`). As these last two phases share important similarities, in the following section we comment on both of them.

5.1.1 `Local/Global approximation step`

As we described earlier, given a partition of the dimensionality set by `Split step`, in `Local approximation step`, we evaluate the effect of each variable on the clustering structure in a given set of dimensions, D_i. In particular, we propose a simple criterion, that leads to different theoretical guarantees in terms of the K-means error. Such a criterion consists of primarily running algorithm $\mathcal{A}$ on X_{D_i}, to unveil the clustering structure for a predefined subset of dimensions D_i. Afterwards, the obtained solution, C_i, and its corresponding clustering, $\mathcal{P}_i$, is used to determine a subset of dimensions in D_i, for which the clustering structure does not have a major variation (measured via the K-means error), if discarded.

Unfortunately, as we may wish to select a set of dimensions from a given subset $D \subseteq \{1, \ldots, d\}$, evaluating the clustering obtained for all the possible combinations of fixed dimensions could be an expensive task. To tackle this difficulty, in Theorem 5, we present a simple way of computing the importance of a variable on, D, inspired on the previous idea, but that does not require the computation of the clustering re-assignements that occur when fixing a dimension. This measurement, additionally, provide us with a bound to the K-means error increment that the new modified clustering carries on.

Theorem 5 *Given a dimensions subset $D \subseteq \{1, \dots, d\}$, a set of centroids $C = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_K\}$ in $\mathbb{R}^{|D|}$ and its associated clustering $\mathcal{P} = \{P_1, \dots, P_K\}$, then for any subset $S = \{s_1, \dots, s_m\} \subseteq D$ and vector $\boldsymbol{v} = (v_1, \dots, v_m)$, the set of centroids $C' = \{\boldsymbol{c}'_1, \dots, \boldsymbol{c}'_K\}$, defined as*

$$\boldsymbol{c}'_{l,j} = \begin{cases} \boldsymbol{c}_{l,j}, & \text{if } j \in S \\ v_j, & \text{otherwise} \end{cases} \tag{5.1}$$

satisfies $E^{X_D}(C') \leq E^{X_D}(C) + \sum_{j \in D \setminus S} t_j$, where

$$t_j = \sum_{l=1}^{K} |P_l| \cdot (\boldsymbol{c}_{l,j} - v_j)^2 \tag{5.2}$$

Theorem 5 offers a simple way of quantifying the importance of a certain dimension, in terms of its impact on the quality of the obtained clustering. Such a measurement consists on fixing the corresponding entry, on each center of mass, to a given value, v_j[2], and estimating the increase of the error that it implies. Observe that computing the scores of all the variables $j \in D$, t_j, can be done in just $\mathcal{O}(K \cdot |D|)$ time. A particular benefit of this approach, relies on the fact that the effect of each variable is additive (Eq.5.2), meaning that the corresponding error increase of each dimension is independent of the subset of fixed dimensions, $D \setminus S$. In Remark 1, we comment on a straightforward feature selection process, based on Theorem 5, that leads to a $1+\varepsilon$- approximation of $E^{X_D}(C)$.

Remark 1 *Using Theorem 5, we can minimize the size of the set of features selected, $S \subseteq D$, needed to keep a $1+\varepsilon$- approximation of C. This can be easily done by computing the set $\{t_j\}_{j=1}^{|D|}$ and sorting it increasingly, $\{t_{j:|D|}\}_{j=1}^{|D|}$. We then determine the largest index $k \in \{1, \dots, |D|\}$ for which, $\sum_{l=1}^{k} t_{l:|D|} \leq \varepsilon \cdot E^{X_D}(C_i)$ holds, i.e., $E^{X_D}(C') \leq (1+\varepsilon) \cdot E^{X_D}(C)$. Therefore, the features selected, $S \subseteq D$, correspond to those dimensions associated to the last $|D| - k$ entries in $\{t_{j:|D|}\}_{j=1}^{|D|}$.* [3].

[2] From now on, we will consider v_j to be the center of mass of the j^{th} variable of X, as it minimizes t_j and so, the bound presented in Theorem 5.

[3] In Section .3.3, we briefly present some additional practical results showing the accuracy of the proposed feature selection procedure.

The construction proposed in Remark 1 is fairly simple and can be done in $\mathcal{O}(|D| \cdot \max\{n \cdot K, \log |D|\})$ time (incluiding the computation of the K-means error). Furthermore, if we use Remark 1 to select a set of variables $S_i \subseteq D_i$, for all $i \in \{1, \ldots, t\}$, then applying algorithm $\mathcal{A}$ on $S = \bigcup_{i=1}^{t} S_i$ leads to a $\mathcal{O}(1+\varepsilon)$- approximation of the K-means problem in X, see Theorem 6.

Theorem 6 *Given a data set X, a constant $\varepsilon > 0$ and a partition of the dimensions $\{1, \ldots, d\}$ into t disjoint groups $\{D_1, \ldots, D_t\}$, if the set of features selected $S_i \subseteq D_i$ is obtained via Remark 1, for all $i \in \{1, \ldots, t\}$, then the output of a λ-approximate K-means algorithm (algorithm $\mathcal{A}$) on $S = \bigcup_{i=1}^{t} S_i$, C^*, satisfies*

$$E^X(C^*) \leq \varphi \cdot (1+\varepsilon) \cdot E^X(C_{opt}) \tag{5.3}$$

where $C_{opt} = \underset{C \subseteq \mathbb{R}^d, |C|=K}{\arg\min} E^X(C)$, $\varphi = \lambda^2 \cdot \frac{E_{K=1}^{X_S, opt}}{\sum_{i=1}^{t} E^{X_{S_i}}(C_i)}$ and, $E_{K=1}^{X_S, opt}$, is the optimal value of the 1-means error on X_S.

The result presented in Theorem 6 shows that, by performing such a feature selection procedure, that locally controls the increase of the K-means error in different groups of dimensions, allows us to preserve the same order of accuracy when compared over the entire set of dimensions. In particular, one must observe that the approach presented in Remark 1, in fact, intends to minimize the quality ratio, $\varphi \geq 1$, as it tends to select those features that maximizes $E^{X_{S_i}}(C_i)$ for all $i \in \{1, \ldots, t\}$. In addition, and as commented in Section 5.1, we would like to highlight that, for $K = \mathcal{O}(1)$, there exist different polynomial-time approximation schemes for the K-means algorithm [27, 60]. This fact indicates that the approximation ratio, λ, can be taken to be $1+\varepsilon$, for any constant $\varepsilon > 0$ [80]. In other words, we can additionally reduce the number of instances being used in both phases of the algorithm, i.e., `Local approximation step` and `Global approximation step`, and the obtained set of centroids, C^*, would still be a $\mathcal{O}((1+\varepsilon)^3)$- approximation.

Besides the fact that Remark 1 offers a simple and yet effective way of selecting the features in a parallel manner, while controlling the quality of the approximation, for this approach there is no guarantee that the set of features selected, S, satisfies the equality $|S| = m$. To tackle this difficulty, we will consider different values $\varepsilon_i > 0$, for each party D_i in $i \in \{1, \ldots, t\}$, such that the total number of selected features is $|S| = m$. In this regard, we can take into consideration the following corollary.

Corollary 2 *Given a set of positive constants $\{\varepsilon_1, \ldots, \varepsilon_t\}$, then if the set of features selected $S_i \subseteq D_i$ is obtained via Remark 1, with $\varepsilon = \varepsilon_i$, for all $i \in \{1, \ldots, t\}$, then the output of a λ-approximate K-means algorithm (algorithm $\mathcal{A}$) on $S = \bigcup_{i=1}^{t} S_i$, C^*, satisfies*

$$E^X(C^*) \leq \varphi \cdot (1 + \max_{i \in \{1,\ldots,t\}} \varepsilon_i) \cdot E^X(C_{opt}) \tag{5.4}$$

Corollary 2 provides us a heuristic to decide how many features should be selected from each of chunk of dimensions D_i: Apply the approach discussed in Remark 1 with a set of epsilons $\{\varepsilon_1, \ldots, \varepsilon_t\}$, for which $|S| = \sum_{i=1}^{t} |S_i| = m$ and $\max_{i \in \{1,\ldots,t\}} \varepsilon_i$ is minimized. In particular, using Remark 1, we observe that, in order to select d_i variables from D_i, we just need to set $\varepsilon_i = \sum_{l=1}^{|D_i|-d_i} \frac{t^i_{l:|D_i|}}{E^{X_{D_i}}(C_i)}$. Hence, we can approach the feature selection by solving the following problem:

Problem 1 *Determine a set of non-negative integers* $\{d_1, \ldots, d_t\}$*, such that* $\sum_{i=1}^{t} d_i = m$ *and* $\max_{i \in \{1,\ldots,t\}} \sum_{l=1}^{|D_i|-d_i} \frac{t^i_{l:|D_i|}}{E^{X_{D_i}}(C_i)}$ *is minimized.*

If we define the lists $\mathcal{E}_i = \{\sum_{l=1}^{|D_i|-d_i} \frac{t^i_{l:|D_i|}}{E^{X_{D_i}}(C_i)}\}_{d_i=0}^{|D_i|-1}$, for all $i \in \{1, \ldots, t\}$, we can solve Problem 1 by selecting the m largest entries in $\{\mathcal{E}_1, \ldots, \mathcal{E}_t\}$. However, as the lists $\mathcal{E}_i$, for all $i \in \{1, \ldots, t\}$, are previously sorted in Remark 1, Problem 1 can be easily solved by recursively comparing if $\max_{i \in \{1,\ldots,t\}} \mathcal{E}_i^{d_i}$ decreases, where d_i is the number of variables selected from the D_i, when we decide selecting one more variable from any subset of variables, which is taken from the one corresponding to the largest error. This process has a $\mathcal{O}(t)$ time complexity per iteration [4].

Finally, after selecting m features via the previous approach, the last step of KMR is to apply algorithm $\mathcal{A}$ on X restricted to the features selected (`Global approximation step`). In Algorithm 9, we put together all the steps that were just discussed, which constitutes the KMR algorithm.

[4] See Section .3.2, for more details

Algorithm 9: ***K*-means relevance for feature selection**

```
Input: Dataset $X$, number of clusters $K$ and $m < d$.
Output: Approximation of Eq.1.1, $C^*$.
• Split step:
- Set $t = \lceil \frac{d}{m} \rceil$, $f = m \cdot t - d$ and divide $\{1, \dots, d\}$ into $t$ disjoint parties,
  $\{D_1, \dots, D_t\}$, where $|D_i| = m - \lfloor \frac{f}{t} \rfloor - 1$, for $i \leq f - \lfloor \frac{f}{t} \rfloor \cdot t$, and
  $|D_i| = m - \lfloor \frac{f}{t} \rfloor$, otherwise.
• Local approximation step:
for $i = 1, \dots, t$ do
    - Apply algorithm $\mathcal{A}$ on $X$ (restricted to $D_i$)$\rightarrow$ Obtains set of
      centroids $C_i = \{\mathbf{c}_1, \dots, \mathbf{c}_K\}$ and clusterings $\mathcal{P}_i = \{P_1, \dots P_K\}$.
    - Compute $t^i_j = \sum_{l=1}^{K} |P_l| \cdot (\mathbf{c}_{l,j} - v_j)^2$ for all $j \in D_i$.
    - Sort $\{t^i_1, \dots, t^i_{|D_i|}\}$ increasingly $\rightarrow \{t^i_{1:|D_i|}, \dots, t^i_{|D_i|:|D_i|}\}$.
    - Set $\mathcal{E}_i = \{\sum_{l=1}^{1} \frac{t^i_{l:|D_i|}}{E^X_{D_i}(C_i)}, \dots, \sum_{l=1}^{|D_i|} \frac{t^i_{l:|D_i|}}{E^X_{D_i}(C_i)}\}$.
end
• Feature selection step:
- Select $m$ largest variables from $\{\mathcal{E}_1, \dots, \mathcal{E}_t\} \rightarrow$ Set of features $D$.
• Global approximation step:
-Apply algorithm $\mathcal{A}$ on $X$, restricted to the dimensions $D \rightarrow$ Obtains
  clustering $\mathcal{P} = \{P_1, \dots P_K\}$.
Return $C^* = \{\overline{P_1}, \dots, \overline{P_K}\}$.
```

The complexity of algorithm 9 depends of the cost of algorithm $\mathcal{A}$ [5]. In particular, if we set algorithm $\mathcal{A}$ to be the standard K-means algorithm [14], K-means++ algorithm [49] or even the Boundary Weighted K-means algorithm [79], the cost of algorithm $\mathcal{A}$ is bounded by $\mathcal{O}(n \cdot K \cdot m)$. Hence, the computational cost of `Local approximation step`, on each machine, is $\mathcal{O}(m \cdot \max\{n \cdot K, \log m\})$. Afterwards, selecting the m features can be done in $\mathcal{O}(t)$ and applying `Global approximation step` is $\mathcal{O}(n \cdot K \cdot m)$. A further interesting remark is that KMR, in the extreme case $K = n$, is equivalent to selecting the m features with the largest variances, which is another commonly used feature selection strategy [65, 66].

5.2 Experiments

In this section, we perform a series of experiments so as to analyze the trade-off between the computational time and the quality of the approximation

[5] Due to its quality guarantees and competitive performance, from now on we take algorithm $\mathcal{A}$ as the K-means algorithm initialized via K-means++.

obtained by the K-means algorithm, after applying it on a wide variety of pre-processed data sets via different dimensionality reduction techniques.

Given a predefined number of features to be extracted/selected, we compare the performance of the K-means relevance for feature selection (K**MR**) with respect to the K-means algorithm[6] applied on m features selected via i) Laplacian Scores (**LS**), ii) Maximum variance (**MaxVar**) and iii) Uniformly at random (**Rand**), or extracted via i) Random Projections (**RP**), ii) Principal Component Analysis (**PCA**) and iii) Singular Value Decomposition (**SVD**). We analyze the performance of these methods on a wide variety of well-known real data sets (see Table 5.1) with different scenarios of the clustering problem. The number of features to be selected/extracted is fixed as $m \in \{10, 25, 50, 75, 100\}$[7]. Furthermore, due to the random nature of the experimental setting, each experiment has been repeated 20 times.

Table 5.1: Information of the data sets.

Data Set	n	K	d	Data Set	n	K	d
KC1 Binary	145	2	91	mfeat Fourier	2000	10	76
Amazon Mechanical	180	10	500	Scene	2407	3	299
Micro Mass	571	20	1300	GINA Agnostic	3468	2	970
Breast Cancer	604	2	10936	Bio Response	3751	2	1776
Arcene NIPS	700	2	10000	Spambase	4601	2	57
Gene RNA-Seq	801	5	20531	Waveform Generator	5000	3	40
Ova Uterus	1545	2	10936	Satellite Image	6430	6	36
Madelon NIPS	2000	2	500	USPS LeCun	7291	10	256

For each experimental setting, we evaluate the relative K-means error obtained for each method M, $\hat{E}_M = \frac{E_M - E_{KM++}}{E_{KM++}}$, where E_M and E_{KM++} stand for the K-means error of method $M \in \{K\text{MR, LS, MaxVar, Rand, PCA, RP, SVD}\}$ and KM++, over all the dimensions of the data set, respectively. To analyze how well each method preserves the clustering quality when compared to KM++, we additionally present the Adjusted Rand Index (ARI) [82, 83] of each method M with respect to the clustering obtained by KM++. In terms of the computational resources, we show the proportion of the computational time of each method M, t_M, with respect to that of KM++, $\hat{t}_M = \frac{t_M}{t_{KM++}}$. To start the analysis, in Tab.5.2-5.3, we show the average for these factors, over all the data sets and considered methods.

[6] In order to guarantee the obtained clustering to be competitive, we use K-means algorithm initialized via K-means++ (K**M++**).

[7] So that the performed dimensionality reduction is not neglegible, for those data sets with $d \leq 100$, we set $m \leq \frac{3}{4} \cdot d$, e.g., for mfeat Fourier ($d = 76$), we analyze reductions to $m \in \{10, 25, 50\}$ features.

Table 5.2: **Relative error -** average over all data sets-.

Method	m=10	m=25	m=50	m=75	m=100
KMR	4.1×10^{-2}	1.2×10^{-2}	6.4×10^{-3}	4.0×10^{-3}	2.3×10^{-3}
LS	2.7×10^{-1}	9.8×10^{-2}	7.6×10^{-2}	4.8×10^{-2}	5.0×10^{-2}
MaxVar	8.6×10^{-2}	2.5×10^{-2}	1.3×10^{-2}	1.0×10^{-2}	9.0×10^{-3}
Rand	2.9×10^{-1}	2.0×10^{-1}	1.6×10^{-1}	1.2×10^{-1}	1.2×10^{-1}
PCA	1.6×10^{-3}	1.7×10^{-3}	8.4×10^{-6}	5.3×10^{-5}	7.7×10^{-5}
RP	9.3×10^{-2}	3.5×10^{-2}	1.2×10^{-2}	1.3×10^{-2}	7.8×10^{-3}
SVD	4.0×10^{-4}	5.9×10^{-4}	7.2×10^{-5}	8.4×10^{-5}	7.9×10^{-5}

Table 5.3: **(ARI, Relative computational time)** - average over all data sets-.

Method	m=10	m=25	m=50	m=75	m=100
KMR	(0.69, 0.28)	(0.75, 0.34)	(0.77, 0.27)	(0.80, 0.25)	(0.83, 0.20)
LS	(0.42, 5.78)	(0.60, 5.82)	(0.56, 1.56)	(0.63, 1.60)	(0.60, 1.51)
MaxVar	(0.63, 0.27)	(0.72, 0.32)	(0.70, 0.25)	(0.71, 0.23)	(0.68, 0.19)
Rand	(0.23, 0.29)	(0.37, 0.33)	(0.27, 0.27)	(0.45, 0.25)	(0.46, 0.20)
PCA	(0.93, 0.42)	(0.95, 0.54)	(0.94, 0.52)	(0.95, 0.53)	(0.95, 0.55)
RP	(0.49, 0.31)	(0.63, 0.35)	(0.66, 0.29)	(0.68, 0.28)	(0.74, 0.23)
SVD	(0.92, 0.40)	(0.94, 0.50)	(0.94, 0.50)	(0.95, 0.54)	(0.96, 0.60)

At first glance, we observe that, among the feature selection techniques, KMR obtains on average both the lowest relative error and largest ARI with respect to the clusterings achieved by KM++. In particular, for the different numbers of features to be selected, m, KMR consistently obtains average error with a relative error under 0.05 with respect to KM++. This is, the clustering obtained after executing K-means++, on the features selected by KMR, usually had an error increment of under 5% of the lowest error achieved by KM++ over the original data set. On the other hand, MaxVar also generated competitive approximations, however, for the largest numbers of features selected $m = \{50, 75, 100\}$, it has 1 order of magnitude of additional error when compared to KMR, as well as a significantly smaller ARI (for $m = 100$, under 0.15 smaller than KMR). On the other hand, both, LS and Rand, obtained the largest relative errors (for all settings, at least, one order of magnitued larger than KMR) and the smallest ARI (under 0.15 with respect to KMR, in every case). For the feature extraction techniques, we observe that SVD and PCA obtained fairly competitive approximations, preserving almost identically the clustering structure achieved by KM++, while RP generates least accurate approximations, which are commonly improved by KMR. For all the methods, it is clear that as we increase the number features to be extracted/selected, the quality of the approximations improves drastically (at least one order of magnitude of relative error for each method). On the other hand, in Tab.5.3, we observe that all the feature selection approaches, except for LS, have similar computational times, which are, on average, under 35% of the time required by KM++. For the feature extraction methods, we observe that PCA and SVD required, on average, up to 3 times the computational time needed by KMR when extracting the largest number of variables.

5.2.1 Feature Selection

In Fig.5.3, we can observe the results obtained in all 16 data sets for feature selection. As we just commented, among these methods, KMR obtained the most accurate approximations, reducing in average, at least one order of relative error, for $m \in \{50, 75, 100\}$ and regularly reaching an ARI w.r.t. KM++ clustering over 0.05 higher than that achieved by the other approaches. On the downside, LS and Rand consistently obtained by far the least competitive clusterings.

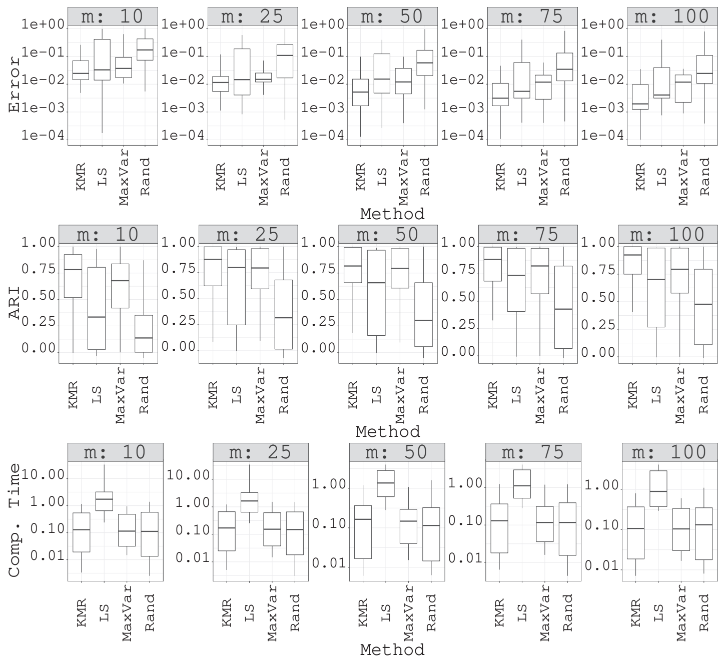

Fig. 5.3: Feature Selection output for all data sets -boxplot-.

In spite of the competitive performance in terms of the quality of the approximation, KMR required the same order of computational time as MaxVar and Rand. Such a behavior is expected, as all these three methods have a time complexity dominated by the K-means run, over the selected variables, which is linear w.r.t. n, K and m. To observe this in more detail and, as the data sets presented in Tab.5.1 have a quite different characteristics, we have divided each factor (dimensionality, number of instances and classes) into three regularly-sized groups and computed the relative error, relative computational time and ARI, for all the feature selection methods, see Tab.5.4-5.6.

Table 5.4: **(Relative error, ARI, Relative computational time)** - average over groups of dimensions-.

Method	$d \leq 100$	$100 < d < 1000$	$d \geq 1000$
KMR	$(1.1 \times 10^{-2}, 0.88, 0.80)$	$(2.1 \times 10^{-2}, 0.70, 0.28)$	$(1.0 \times 10^{-2}, 0.81, 0.04)$
LS	$(2.6 \times 10^{-1}, 0.59, 12.47)$	$(6.2 \times 10^{-2}, 0.48, 2.09)$	$(1.1 \times 10^{-1}, 0.60, 0.91)$
MaxVar	$(6.4 \times 10^{-2}, 0.81, 0.77)$	$(3.3 \times 10^{-2}, 0.60, 0.24)$	$(1.9 \times 10^{-2}, 0.71, 0.05)$
Rand	$(2.3 \times 10^{-1}, 0.57, 0.78)$	$(7.6 \times 10^{-2}, 0.34, 0.30)$	$(2.6 \times 10^{-1}, 0.31, 0.04)$

Table 5.5: **(Relative error, ARI, Relative computational time)** - average over groups of classes-.

Method	$K \leq 3$	$3 < K < 10$	$K \geq 10$
KMR	$(5.1 \times 10^{-3}, 0.75, 0.18)$	$(1.6 \times 10^{-2}, 0.91, 0.40)$	$(3.0 \times 10^{-2}, 0.70, 0.41)$
LS	$(1.3 \times 10^{-1}, 0.52, 2.82)$	$(8.3 \times 10^{-2}, 0.63, 7.16)$	$(1.1 \times 10^{-1}, 0.60, 2.04)$
MaxVar	$(1.1 \times 10^{-2}, 0.65, 0.18)$	$(6.5 \times 10^{-2}, 0.84, 0.38)$	$(5.4 \times 10^{-2}, 0.66, 0.37)$
Rand	$(1.4 \times 10^{-1}, 0.28, 0.17)$	$(1.4 \times 10^{-1}, 0.56, 0.45)$	$(3.5 \times 10^{-1}, 0.44, 0.39)$

Table 5.6: **(Relative error, ARI, Relative computational time)** - average over groups of instances-.

Method	$n \leq 750$	$750 < n < 2500$	$n \geq 2500$
KMR	$(1.1 \times 10^{-2}, 0.81, 0.28)$	$(5.0 \times 10^{-3}, 0.89, 0.24)$	$(3.0 \times 10^{-2}, 0.62, 0.31)$
LS	$(1.7 \times 10^{-1}, 0.42, 0.66)$	$(8.6 \times 10^{-2}, 0.70, 1.75)$	$(1.0 \times 10^{-1}, 0.60, 8.79)$
MaxVar	$(1.8 \times 10^{-2}, 0.79, 0.26)$	$(5.1 \times 10^{-3}, 0.85, 0.23)$	$(7.7 \times 10^{-2}, 0.40, 0.30)$
Rand	$(3.0 \times 10^{-1}, 0.31, 0.32)$	$(1.4 \times 10^{-1}, 0.37, 0.23)$	$(1.0 \times 10^{-1}, 0.42, 0.28)$

According to Tab.5.4-5.6, it is clear that regardless of the clustering scenario, KMR, on average, provides both the lowest relative error and largest ARI w.r.t. KM++. However, in spite of the groups selected for each factor (dimensionality, number of instaces and classes), we do not observe a major variation in terms of the quality of the obtained clusterings. As shown, in both Tab.5.2-5.3 and Fig.5.3, the number of features selected indeed has a larger effect in the comparison of the clusterings obtained by the different methods considered w.r.t. KM++. As previosuly discussed, in terms of the computational time, for the different configurations considered, KMR, MaxVar and Rand have comparable time requirements, however LS also has a poor performance in this regard, specially as increasing the number of instances, e.g., for the group "$n \geq 2500$", LS had a computational time 28.35 times larger than that of KMR, on average.

5.2.2 Feature Extraction

In this section we perform the same analysis as in Section 5.2.1, but comparing KMR to the different feature extraction techniques considered: In Fig. 5.4, we observe the results obtained for all the data sets w.r.t. the number of features extracted. Moreover, in Tab. 5.7-5.9, we present the relative error, relative computational time and ARI and for the different groups of dimensionality, number of instances and classes.

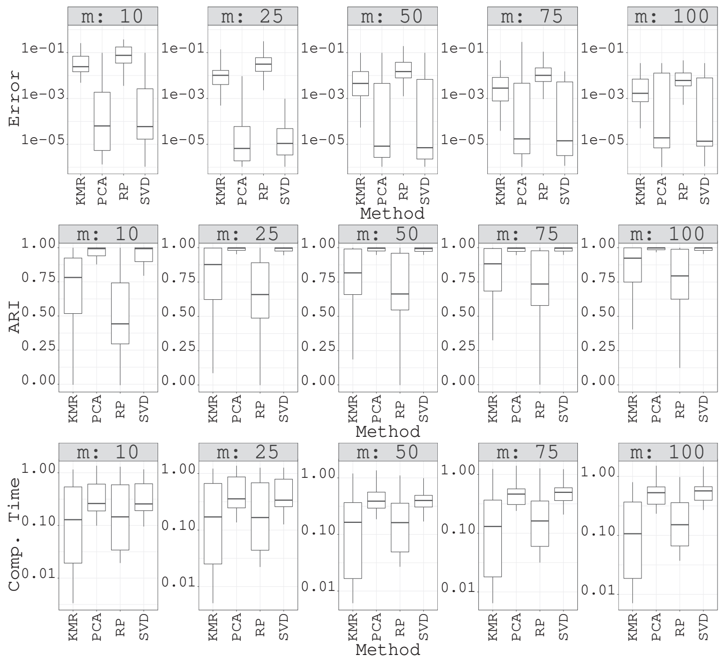

Fig. 5.4: Feature Extraction output for all data sets -boxplot-.

Table 5.7: **(Relative error, ARI, Relative computational time)** - average over groups of dimensions-.

Method	$d \leq 100$	$100 < d < 1000$	$d \geq 1000$
KMR	$(1.1 \times 10^{-2}, 0.88, 0.80)$	$(2.1 \times 10^{-2}, 0.70, 0.28)$	$(1.0 \times 10^{-2}, 0.81, 0.04)$
PCA	$(4.0 \times 10^{-3}, 0.94, 0.90)$	$(1.0 \times 10^{-3}, 0.96, 0.52)$	$(2.2 \times 10^{-5}, 0.93, 0.34)$
RP	$(5.1 \times 10^{-2}, 0.73, 0.79)$	$(3.0 \times 10^{-2}, 0.50, 0.31)$	$(3.7 \times 10^{-2}, 0.69, 0.08)$
SVD	$(1.4 \times 10^{-3}, 0.93, 0.91)$	$(1.2 \times 10^{-3}, 0.96, 0.48)$	$(3.0 \times 10^{-5}, 0.93, 0.37)$

Table 5.8: **(Relative error, ARI, Relative computational time)** - average over groups of classes-.

Method	$K \leq 3$	$3 < K < 10$	$K \geq 10$
KMR	$(5.1 \times 10^{-3}, 0.75, 0.18)$	$(1.6 \times 10^{-2}, 0.91, 0.40)$	$(3.0 \times 10^{-2}, 0.70, 0.41)$
PCA	$(1.3 \times 10^{-4}, 0.97, 0.46)$	$(1.6 \times 10^{-3}, 0.97, 0.72)$	$(1.4 \times 10^{-5}, 0.85, 0.47)$
RP	$(1.9 \times 10^{-2}, 0.59, 0.20)$	$(4.5 \times 10^{-2}, 0.80, 0.49)$	$(8.1 \times 10^{-2}, 0.58, 0.41)$
SVD	$(1.0 \times 10^{-5}, 0.97, 0.47)$	$(1.8 \times 10^{-3}, 0.96, 0.64)$	$(2.0 \times 10^{-5}, 0.85, 0.48)$

Table 5.9: **(Relative error, ARI, Relative computational time)** - average over groups of instances-.

Method	$n \leq 750$	$750 < n < 2500$	$n \geq 2500$
KMR	$(1.1 \times 10^{-2}, 0.81, 0.28)$	$(5.0 \times 10^{-3}, 0.89, 0.24)$	$(3.0 \times 10^{-2}, 0.62, 0.31)$
PCA	$(2.0 \times 10^{-3}, 0.88, 0.62)$	$(1.6 \times 10^{-5}, 0.97, 0.43)$	$(9.3 \times 10^{-5}, 0.98, 0.48)$
RP	$(3.4 \times 10^{-2}, 0.68, 0.33)$	$(5.6 \times 10^{-2}, 0.65, 0.25)$	$(4.2 \times 10^{-2}, 0.55, 0.33)$
SVD	$(2.7 \times 10^{-4}, 0.87, 0.56)$	$(2.2 \times 10^{-5}, 0.97, 0.46)$	$(1.7 \times 10^{-4}, 0.98, 0.49)$

In this case, the analysis of the results, especially in terms of the computational time, is more interesting than in Section 5.2.1, as the time demands are not neccesarily dominated by the corresponding K-means run for each method. We can see in Tab.5.7-5.9 that there is no clear improvement/ diminishment, in terms of clustering quality (relative error and ARI), as we increase either of the three factors. As we show in Tab.5.2-5.3 and Fig.5.4, the quality of the obtained clustering seems to be largely dominated by the number of features extracted, regardless of the method, as in Section 5.2.1. In any case, despite of the considered setting, we must remark that KMR outperforms the clustering quality of RP, in both ARI and error.

In terms of the computational time required w.r.t. KM++, we observe a large effect of the orginal dimensionality of the data set, d. Unfortunately, as the dimensionality reduction step via SVD and PCA is supralinear w.r.t. d, this phase is far more time consuming than running the K-means algorithm on the extracted/selected variables. For this reason, we observe that in the group "$d \leq 100$", PCA and SVD just need, on average, 1.17 and 1.16 times the amount computational time of KMR, while, for "$d \geq 1000$", the time demands increase to 8.50 and 9.25 times, respectively. In other words, methods like KMR and RP can be more suitable for pre-processing massive data sets.

On the other hand, KMR also outperforms time-wise RP. However, in this case the time reduction is not significant. This is due to the fact that the computational time of RP also grow linearly with respect to the different factors and that, its feature extraction step, is independent of the number of classes. Hence, as we consider larger values of clusters, we expect a more competitive perfomance, in terms of computational resources for RP, specially when $d \leq K$, which is a very unlikely case as most of the applications considered for this analysis tend to have massive dimensionalities.

Regardless of the characteristics of the clustering problem, we observe that KMR shows a very competitive performance, in both quality of the obtained solution and computational time, when compared to different feature selection techniques, with particular emphasis on LS. Furthermore, our extensive experimental analysis also shows that, even when KMR is a feature selection technique, it is able to outperform, in both accuracy and computational time, well-known feature extraction technique used for for the K-means problem, such as RP, which makes it a very suitable pre-processing alternative to other approaches that may not scale well on massive data sets, such as SVD and PCA.

5.3 Conclusions

In this chapter, we propose a fully-parellelizable cheap feature selection technique intended for the K-means algorithm called the K-means relevance for feature selection algorithm, or just KMR. It consists of solving the K-means problem, on small subsets of dimensions of the original dataset, across multiple machines and using the obtained clustering information to upper-bound the increase in the K-means error when a certain feature is not selected. The cost of the proposed method is $\mathcal{O}(m \cdot \max\{n \cdot K, \log m\})$, where m is the number of features selected, per machine. Besides providing a bound to the K-means error obtained by these technique, in the practice we compare KMR to different well-known feature selection and feature extraction techniques, commonly used for the K-means problem, on a wide variety of real-life data sets. The obtained results testify that KMR regularly obtains solutions with lower K-means than all the considered feature selection techniques: Laplacian scores, maximum variance and random selection, while also requiring similar or lower computational times than these approaches. Even more interesting, KMR when compared to feature extraction techniques, such as Random Projections, also shows a noticeable improvement in both error and computational time.

As a future step, since our technique consists of solving multiple small-dimensional K-means problems, we would like to analyze the effect of using different coreset techniques, such as [22, 23, 78, 27, 59, 60], to further reduce the computational requirements of our approach. Moreover, we also plan to design a feature extraction technique of the same nature of KMR, that allows us not to fully lose the information of given feature when not selected.

6

An efficient Split-Merge re-start for the *K*-means algorithm

As it is well documented in the literature, one factor that must be considered when reducing the computational requirements of the K-means algorithm, is the quality of its initialization [49, 44, 45, 46, 47]. A poor initialization not only may affect the quality of the obtained approximation, but may also increase tremendously the number of iterations required to convergence [74, 47]. In order to ease the convergence to competitive approximations of the K-means problem, the common practice is to re-start multiple times Lloyd's algorithm, with different seeds, and to keep the solution with the lowest error [45, 46]. Taking this into consideration, in this chapter, we address the problem of K-means algorithm convergence to uncompetitive approximations. In particular, we propose a low-cost re-start strategy that, by using the clustering information of any fixed point of Lloyd's algorithm, constructs a set of centroids with a similar or lower error than that of the given fixed point and that is also likely to lie on a different basin of attraction.

As we already mentioned, the standard approach is to apply Lloyd's algorithm several times, with a predetermined seeding strategy, and to pick the set of centroids with the lowest K-means error [45, 46]. Unfortunately, one of the biggest drawbacks of this approach is that no information is transmitted from one re-start to another, which could be a key element to avoid converging to the same local minima several times, as well as for reducing the number of iterations executed when running Lloyd's algorithm.

In this sense, we propose a re-start to the K-means algorithm, based on a Split-Merge approach, that only uses information associated to the current K-means local minima. Such a technique generates a new initialization to the K-means algorithm by placing an additional centroid on a certain cluster P_s, from which a large amount of error can be reduced (`SplitStep`). Afterwards, two other clusters P_i and P_j are merged. These clusters are ideally close to each other, and so their union does not add a significant error to our approximation (`MergeStep`). Under certain conditions (see for instance Theorem 8), this process itself already generates a set of centroids C' with an associated error lower than that of the previous fixed point. Although such a descent may not always occur, this process still generates a re-initialization that is likely to be located at a different basin of attraction and with a K-means error close to that of the current local minima, which may allow the subsequent Lloyd's algorithm run to reduce the error.

In terms of the split step, our proposal has some similarities to that of the X-means algorithm [84]. However, this method, as well as others of similar nature, such as ISODATA [85], deals with different problems, e.g., determining an adequate number of clusters. Contrary to these techniques, our approach uses different seeding techniques (based on the K-means++ algorithm) which commonly lead to more competitive results [49]. Moreover, we also provide different error descent conditions for our Split-Merge process, while the aforementioned algorithms do not provide any kind of theoretical guarantees.

The rest of this article is organized as follows: We first describe in detail the proposed algorithm and discuss some of its properties. Afterwards, we analyze its performance w.r.t. the state-of-the-art (Multi-start Forgy K-means, K-means++, Hartigan) and define the next steps and possible improvements to our current work.

6.1 The Split-Merge K-means algorithm

In this section, we formally introduce the Split-Merge K-means algorithm, or just **SMK-means** (Algorithm 10). The SMK-means can be seen as a multi-start approach that re-initializes the K-means algorithm by applying a Split-Merge step over the clustering obtained in the previous iteration, $\mathcal{P}$.

Algorithm 10: SMK-means Algorithm

```
Input: Data set $X$, number of clusters $K$ and an initial set of centroids $C'$.
Output: Local minima of Eq.1.1, $C$.
• Step 1 (Alg.1): $C', \mathcal{P} = \texttt{Lloyd}(X, K, C')$
while not Stopping Criterion do
    • Set $C = C'$.
    • Step 2 (Alg.11): $C', \mathcal{P}' = \texttt{SplitStep}(C, \mathcal{P})$
    • Step 3 (Alg.12): $C' = \texttt{MergeStep}(C', \mathcal{P}')$
    • Step 4 (Alg.1): $C', \mathcal{P} = \texttt{Lloyd}(X, K, C')$
end
Return $C$
```

In `Step 1` and `Step 4`, Lloyd's algorithm is applied and the obtained set of centroids (and their corresponding clusters) are used in `Step 2` and `Step 3` to generate the new re-initialization: `Step 2` divides the cluster from where the largest error reduction is achieved, while `Step 3` merges the pair of clusters whose fusion produces the smallest increase in the error. The goal is that this re-initialization mechanism facilitates the convergence of Lloyd's algorithm to a competitive local minima by detecting those regions that might be under and/or over-represented[1].

We would like to point out that, in the first iteration of Algorithm 10, the while loop is always executed. Afterwards, the proposed *Stopping Criterion* is to run the algorithm as long as the error obtained in `Step 4` decreases with respect to this value in the previous iteration, i.e., $E_X(C') < E_X(C)$. In general, note that our Split-Merge procedure modifies, at most, three of the current clusters, therefore one would also expect a faster convergence of Lloyd's algorithm, in `Step 4`, for our new set of centroids than for Forgy's or K-means++ re-start (see the experiments section).

6.1.1 Cluster Split (`SplitStep`)

The `SplitStep` (Algorithm 10, `Step 2`) is the phase of the re-initialization in which we intend to determine a convenient region to place an additional centroid. To do so, one can re-use the distances computed in the last iteration of `Step 1` and `Step 4` to determine the cluster from where the largest error reduction is attained by placing an extra centroid. In particular, in order to quantify such an error decrease, we propose to apply a 2-means algorithm run on each cluster.

Formally speaking, given a set of centroids, $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$, and its corresponding clustering, $\mathcal{P} = \{P_1, \ldots, P_K\}$, `SplitStep` (Algorithm 11) approximates a solution to the 2-means problem over each cluster P_k via a 2-means algorithm run. This process generates pairs of centroids $\{\mathbf{c}_k^1, \mathbf{c}_k^2\}$ for $k \in \{1, \ldots, K\}$ and then the new centroid is placed on the cluster P_s satisfying

$$s = \underset{k\in\{1,\ldots,K\}}{\arg\max} \underbrace{E_{P_k}(\{\mathbf{c}_k\}) - E_{P_k}(\{\mathbf{c}_k^1, \mathbf{c}_k^2\})}_{g_k} \tag{6.1}$$

Observe that if we denote by $P_k^1 = \{\mathbf{x} \in P_k : \|\mathbf{x} - \mathbf{c}_k^1\|^2 \leq \|\mathbf{x} - \mathbf{c}_k^2\|^2\}$ and $P_k^2 = P_k \setminus P_k^1$, then $g_k = |P_k^1| \cdot \|\mathbf{c}_k - \mathbf{c}_k^1\|^2 + |P_k^2| \cdot \|\mathbf{c}_k - \mathbf{c}_k^2\|^2$. In other words, as we want to maximize the error reduction, we are looking for a large cluster P_s whose 2-means centroids are as far as possible from the center of mass of the cluster P_s, $\overline{P_s}$.

[1] Regions of the space that can be represented with a similar quality (error) by a lower number of centroids.

```
Algorithm 11: SplitStep
  Input: Set of centroids, $C = \{\mathbf{c}_1, \dots, \mathbf{c}_K\}$, and its corresponding
  clustering, $\mathcal{P} = \{P_1, \dots, P_K\}$.
  Output: Updated clustering $\mathcal{P}'$ and centers of mass, $C'$.
  • Step 1: Apply 2-means on $P_k$ for $k \in \{1, \dots, K\}$.
  • Step 2: Select cluster $P_s$ satisfying
        $s = \underset{k \in \{1,\dots,K\}}{\arg\max}\ E_{P_k}(\{\mathbf{c}_k\}) - E_{P_k}(\{\mathbf{c}_k^1, \mathbf{c}_k^2\})$
  • Step 3: Update centers of mass and clusterings:
        $C' = \{C \setminus \{\mathbf{c}_s\}\} \cup \{\mathbf{c}_s^1, \mathbf{c}_s^2\},\ \mathcal{P}' = \{\mathcal{P} \setminus \{P_s\}\} \cup \{P_s^1, P_s^2\}.$
  Return $C'$, $\mathcal{P}'$
```

Even when the split step described in Algorithm 11 has some similarities with the one proposed in the X-means algorithm [84], we must remark that, in [84], the local 2-means approximation is initialized via a perturbation of each center of mass along a random direction, while, in our case, we use 2-means++. As can be seen in the upcoming sections, the effectiveness of our re-initialization process highly depends upon the quality of the approximation obtained for the different 2-means problems. Hence, the use of a seeding technique with strong quality guarantees, such as K-means++, is of our interest. Overall, the cost of `SplitStep` is $\mathcal{O}(n \cdot d)$.

6.1.2 Clusters Merge (`MergeStep`)

In `MergeStep` (Algorithm 10, `Step 3`), we intend to determine a region of the space that is likely to be overrepresented. In particular, our proposed `MergeStep` (Algorithm 12) consists of putting together the pair of clusters that minimizes the error increase after being fused. One way of estimating such an error increase, without recomputing all the pairwise instance-centroid distances, consists of merging the clusters P_i and P_j that minimize

$$f_{i,j} = E_{P_{i,j}}(\{\mathbf{c}_{i,j}\}) - (E_{P_i}(\{\mathbf{c}_i\}) + E_{P_j}(\{\mathbf{c}_j\})), \tag{6.2}$$

where $P_{i,j} = P_i \cup P_j$ and $\mathbf{c}_{i,j} = \overline{P_{i,j}} = \frac{|P_i|\cdot\mathbf{c}_i + |P_j|\cdot\mathbf{c}_j}{|P_i|+|P_j|}$. The next result shows that we can compute $f_{i,j}$ using only the distances $\|\mathbf{c}_i - \mathbf{c}_j\|^2$:

Theorem 7 *Given two clusters P_i and P_j and their corresponding centers of mass $\mathbf{c}_i$ and $\mathbf{c}_j$, then $f_{i,j} = \frac{|P_i|\cdot|P_j|}{|P_i|+|P_j|} \cdot \|\mathbf{c}_i - \mathbf{c}_j\|^2$.*

Therefore, `MergeStep` can be done exactly in $\mathcal{O}(K^2 \cdot d)$ and the final cost of the re-initialization is $\mathcal{O}(\max\{n, K^2\} \cdot d)$, which is cheaper than a single Lloyd's algorithm run.

```
Algorithm 12: MergeStep
  Input: Set of centers of mass, $C = \{\mathbf{c}_1, \dots, \mathbf{c}_{K+1}\}$, and its associated
  clustering, $\mathcal{P} = \{P_1, \dots, P_{K+1}\}$.
  Output: Re-initialization $C'$.
  • Step 1: Select clusters $P_i$ and $P_j$, such that $i, j = \arg\min_{l<k,\ l \neq K} f_{l,k}$.
  • Step 2: Update centers of mass:
      $C' = \{C \smallsetminus \{\mathbf{c}_i, \mathbf{c}_j\}\} \cup \{\frac{|P_i| \cdot \mathbf{c}_i + |P_j| \cdot \mathbf{c}_j}{|P_i| + |P_j|}\}$
  Return $C'$
```

Observe that the input of `MergeStep` is a set of $K + 1$ centroids and their corresponding clusters as it comes after `SplitStep` (see Algorithm 10). Without loss of generality, we assume P_K and P_{K+1} to be the pair of clusters generated by `SplitStep`. Note that the conditions $l < k$ and $l \neq K$ ensure that we are not merging the centroids generated by `SplitStep`, $\mathbf{c}_K$ and $\mathbf{c}_{K+1}$.

6.1.3 Error descent conditions

In comparison to more restrictive techniques, such as Hartigan's heuristic [51] and the "first variation" for the spherical K-means [86], where only a single instance is used to split one cluster and merge to another, our Split-Merge procedure is not necessarily meant to generate a re-initialization with a lower error than that of a given local minima. Instead, its goal is to construct a competitive seed that is likely to be located at a different basin of attraction. In this section, however, we comment on some conditions for which our Split-Merge procedure itself improves the quality of the approximation.

First of all, it should be observed that, as there might be instances in $P_{i,j}$ which are reassigned to other clusters in the application of Lloyd's algorithm after `MergeStep`, the following inequality holds for the obtained set of centroids $C' = \{C \smallsetminus \{\mathbf{c}_s, \mathbf{c}_i, \mathbf{c}_j\}\} \cup \{\mathbf{c}_{s_1}, \mathbf{c}_{s_2}, \frac{|P_i| \cdot \mathbf{c}_i + |P_j| \cdot \mathbf{c}_j}{|P_i| + |P_j|}\}$: $E_X(C') \leq E_X(\{C \smallsetminus \{\mathbf{c}_s\}\} \cup \{\mathbf{c}_s^1, \mathbf{c}_s^2\}) + f_{i,j} \leq E_X(C) + f_{i,j} - g_s$. In other words, if our Split-Merge process determines clusters P_s, P_i and P_j for which $f_{i,j} - g_s < 0$, then already our re-initialization, C', has a lower K-means error than the departing fixed point, C. Moreover, in the following result, we provide an error descent condition after the Split-Merge re-initialization, which can be verified before `Step 4` of Algorithm 10, using only the information of the local minima.

Theorem 8 *Given a Lloyd's algorithm fixed point, C, if the SMK-means algorithm splits a cluster P_s, taking the corresponding 2-means initialization* *via D^2-sampling*[2] *and* $\min_{l \neq k \neq s} \frac{|P_l| \cdot |P_k|}{|P_l|+|P_k|} \cdot \|\boldsymbol{c}_l - \boldsymbol{c}_k\|^2 \leq \frac{E_{P_s}(\{\boldsymbol{c}_s\})}{|P_s|}$*, then the re-initialization C' satisfies $E_X(C') \leq E_X(C)$, on average.*

Theorem 8 implies that the larger the average error is in the cluster to be split and the smaller the closest distance between the different pairs of centroids is, then the more likely it is for our Split-Merge criterion to generate a set of centroids that, without any Lloyd's iteration, has a K-means error smaller than that of the precedent local minima. At the end of the experimental section, we can observe that in practice such a descent is very likely to occur, especially when the number of clusters is large.

6.2 Experiments

In this chapter, we perform a set of experiments so as to analyze the trade-off between the number of distances computed[3] and the quality of the approximation obtained by the SMK-means algorithm (**SMKM**) and different well-known multi-start approaches. In particular, we compare the performance of SMKM, initialized via Forgy (**SMKMr**) and K-means++ (**SMKM++**), to multiple re-starts[4] of Lloyd's algorithm initialized via Forgy (**FKM**) and K-means++ (**KM++**), as well as to Lloyd's algorithm re-initialized via Hartigan's heuristic (**HKM++** and **HKMr**)[5].

In order to have a better understanding of SMKM, we analyze its performance on a variety of real data sets (see Table 6.1) with different scenarios of the clustering problem: size of the data set, n, and dimension of the instances, d. For each data set we also considered a different number of clusters, $K = \{10, 25, 50, 100, 250\}$[6]. Given the random nature of the algorithms, each experiment has been repeated 20 times.

[2] For the sake of simplicity, in Theorem 8, we consider a variation of the 2-means++ initialization that consists of taking the initial centroid as the center of mass of the cluster, $\mathbf{c}_s$, and, as the second centroid, $\mathbf{x} \in C_s$ with probability $\Pr(\mathbf{x}) \propto \|\mathbf{x} - \mathbf{c}_s\|^2$ (D^2-sampling).

[3] As previously commented, the computational load of the considered methods largely relies on the distance computations. Similar analyses can be found in [43, 26].

[4] In this section, we present results for up to 10 re-starts of both KM++ and FKM, since, for our experimental setting, the number of distance computations of SMKM normally does not reach those of KM++ and FKM, for such a number of repetitions, see Fig.6.2 and Table 6.4-6.6.

[5] Additional results, with respect to Hartigan's K-means, can be found in the supplemental material.

[6] Similar values have been used in different works [49, 43, 52].

Table 6.1: Information of the data sets.

Data Set	n	d
Breast Cancer (*BC*)	569	30
Digits (*DIG*)	1,797	64
Anuran Calls (*AC*)	7,193	22
Human Activity Recognition (*HAR*)	7,351	561
Sensor Vehicle Accoustic (*SVA*)	78,823	50
3D Road Network (*3RN*)	434,874	3
Household Power (*HP*)	2,049,280	7
Gas Sensor (*GS*)	4,208,259	19

In Fig.6.1-6.6 and Table 6.2-6.8, we compare the different methods in terms of the quality of their approximation and the number of distance computations. To make such a comparison, we use the relative error with respect to the best solution found at each repetition of the experiment, for each data set and number of clusters, $\hat{E}_M = \frac{E_M - \min\limits_{M' \in \mathcal{M}} E_{M'}}{\min\limits_{M' \in \mathcal{M}} E_{M'}}$, where $\mathcal{M}$ is the set of algorithms being compared and E_M the K-means error obtained by $M \in \mathcal{M}$. In terms of the computational resources, we show the proportion of the distances computed by each method w.r.t. the method that computed the lowest number of distances, i.e., $\hat{DC}_M = \frac{DC_M}{\min\limits_{M' \in \mathcal{M}} DC_{M'}}$, where DC_M is the number of distances computed by $M \in \mathcal{M}$, as well as the relative number of Lloyd's iterations, i.e., $\hat{It}^i_M = \frac{It^i_M}{\min\limits_{M' \in \mathcal{M}, \forall j} It^j_{M'}}$, where It^i_M is the number of Lloyd iterations up to the i^{th} re-start of $M \in \mathcal{M}$.

6.2.1 Quality of the approximation

In Fig.6.1 and Table 6.2-6.3, the quality of the clusterings obtained by each method is analyzed. In general, one can observe that both versions of SMKM (SMKMr and SMKM++) systematically converge to the approximations with the lowest K-means error. In particular, for the different 800 settings considered in our experimental setting, Table 6.3 shows that, among the methods that are initialized via Forgy (FKM, HKMr and SMKMr) SMKMr obtained the best clustering in 71.625% of the cases, while SMKM++ achieved the lowest error in 77.250% of the cases when compared to the methods initialized via K-means++ (KM++, HKM++ and SMKM++). On the other hand, the approach that converged to the best solution the least times is Lloyd's algorithm re-initialized via Hartigan's heuristic (HKMr and HKM++).

Table 6.2: Relative error after last re-start (average over all data sets).

Method	K=10	K=25	K=50	K=100	K=250
FKM	$2.8{\times}10^{-2}$	$1.7{\times}10^{-1}$	$3.6{\times}10^{-1}$	$5.3{\times}10^{-1}$	$7.7{\times}10^{-1}$
KM++	$1.6{\times}10^{-3}$	$6.1{\times}10^{-3}$	$1.1{\times}10^{-2}$	$1.4{\times}10^{-2}$	$2.4{\times}10^{-2}$
HKMr	$7.7{\times}10^{-2}$	$2.8{\times}10^{-1}$	$4.2{\times}10^{-1}$	$6.1{\times}10^{-1}$	$9.0{\times}10^{-1}$
HKM++	$1.6{\times}10^{-2}$	$2.0{\times}10^{-2}$	$1.8{\times}10^{-2}$	$1.3{\times}10^{-2}$	$1.1{\times}10^{-2}$
SMKMr	$2.5{\times}10^{-3}$	$3.6{\times}10^{-3}$	$3.4{\times}10^{-3}$	$3.7{\times}10^{-3}$	$6.4{\times}10^{-3}$
SMKM++	$2.9{\times}10^{-3}$	$4.1{\times}10^{-3}$	$2.7{\times}10^{-3}$	$3.5{\times}10^{-3}$	$4.2{\times}10^{-3}$

Table 6.3: Lowest error approximation (percentage).

FKM	HKMr	SMKMr
17.250%	11.125%	71.625%

KM++	HKM++	SMKM++
13.625%	9.125%	77.250%

As we can verify in Table 6.2 and Fig.6.1, not only are the clusterings obtained by SMKMr and SMKM++ normally the most competitive ones, but their errors tend to be much lower than those of the other methods. In particular, regardless of the number of clusters, the relative error of either version of SMKM is usually of the order 10^{-3}, i.e., SMKMr and SMKM++ obtain approximations with under 1% of added relative error w.r.t. to the best solution found. On the other hand, for the largest numbers of clusters, $K \in \{100, 250\}$, FKM, KM++, HKMr and HKM++ generated on average clusterings with one or two additional orders of magnitude of relative error when compared to SMKMr and SMKM++.

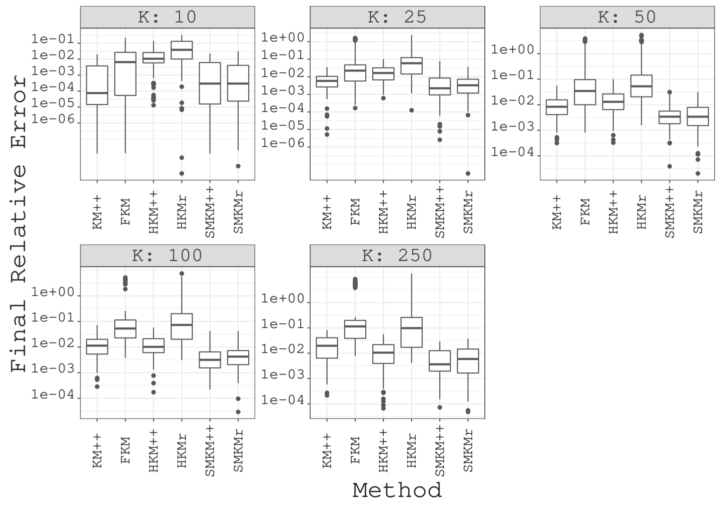

Fig. 6.1: Final relative error for each method (boxplot).

In order to better appreciate the potential of the proposed Split-Merge process, in terms of the quality of its approximation, we could focus the analysis on those methods that are initialized via Forgy, FKM, HKMr and SMKMr, since the error obtained after their first iteration tends to be much higher than the optimal error. In this case, we can observe that the average final relative error of both HKMr and FKM is two orders of magnitude higher than that of SMKMr, for all the number of clusters, except $K = 10$. In the supplemental material, we present additional figures that show the performance of the algorithms on the different data sets. In these figures, we can also observe that both SMKMr and SMKM++ have a much faster rate of convergence than the competition, e.g., for *SVA* and $K = 250$, SMKMr only needed, on average, 16% and 6% of the distances computed by FKM and HKMr, respectively, to generate solutions with the same error, while SMKM++ only computed 16% and 2% of the distances of KM++ and HKM++, respectively.

6.2.2 Distance Computations

Besides the remarkable clustering quality achieved by both SMKM++ and SMKMr, these methods commonly required a much smaller number of distance computations than KM++, FKM, HKM++ and HKMr to generate solutions of better or similar quality to them. Furthermore, as we can see in Fig.6.2 and Table 6.4-6.6, such a statement does not change if we consider instead the total number of distance computations for each method - for which our proposal reduces, on average, up to 2 orders of magnitude of relative error w.r.t. the other algorithms-.

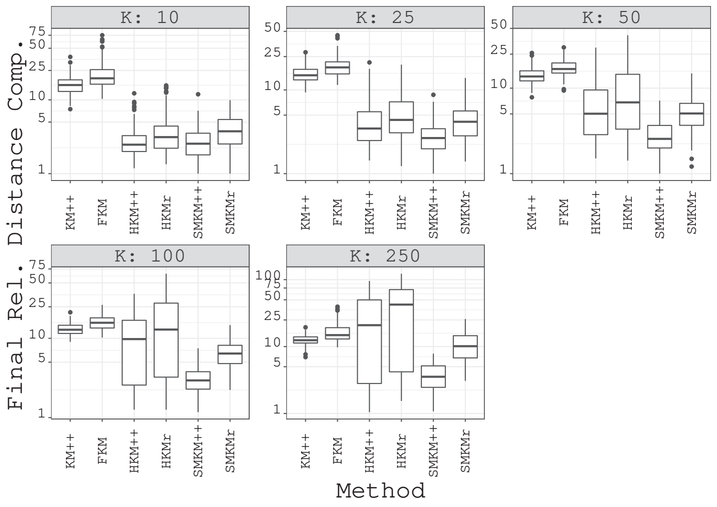

Fig. 6.2: Final relative distance computations for each method (boxplot).

Table 6.4: Final relative distances computed (average over all data sets).

METHOD	K=10	K=25	K=50	K=100	K=250
FKM	22.75	19.54	17.39	15.88	16.84
KM++	16.45	15.79	14.24	13.15	12.67
HKMR	3.99	5.86	9.54	16.04	43.06
HKM++	2.94	4.43	6.80	11.76	30.26
SMKMR	4.06	4.55	5.45	6.77	10.90
SMKM++	2.84	2.91	2.85	3.12	3.84

Table 6.5: Relative error descent per iteration (average over all data sets).

Method	K=10	K=25	K=50	K=100	K=250
HKMr	$2.1{\times}10^{-4}$	$1.3{\times}10^{-3}$	$3.4{\times}10^{-3}$	$3.7{\times}10^{-3}$	$6.9{\times}10^{-3}$
HKM++	$8.4{\times}10^{-4}$	$4.0{\times}10^{-4}$	$3.6{\times}10^{-4}$	$2.3{\times}10^{-4}$	$2.4{\times}10^{-4}$
SMKMr	$3.2{\times}10^{-2}$	$3.7{\times}10^{-2}$	$3.2{\times}10^{-2}$	$2.7{\times}10^{-2}$	$2.9{\times}10^{-2}$
SMKM++	$1.8{\times}10^{-2}$	$1.0{\times}10^{-2}$	$6.0{\times}10^{-3}$	$3.5{\times}10^{-3}$	$1.7{\times}10^{-3}$

Table 6.6: Lowest number of relative distances computed (percentage).

FKM	HKMr	SMKMr
2.250%	35.500%	62.250%

KM++	HKM++	SMKM++
0.125%	34.250%	65.625%

According to Fig.6.2 and Table 6.4, SMKM regularly computes the lowest number of distances, especially as we consider larger numbers of clusters: In particular, for $K = 250$, SMKM++ computed, on average, 3.29 and 7.88 times less number of distances than KM++ and HKM++, respectively. When compared to the typical multi-start Lloyd's algorithm (FKM and KM++), the reduction of distance computations has to do with the fact that SMKM only modifies, at most, three of the current clusters to generate the following re-initialization, therefore the ratio of Lloyd iterations to convergence of FKM and KM++ w.r.t. SMKM tends to increase for a larger number of clusters, see Table 6.7 and Fig.6.3. In particular, observe that for $K = 250$, FKM and KM++ computes, on average, over 5 and 7 times the number of Lloyd iterations of SMKM per re-start.

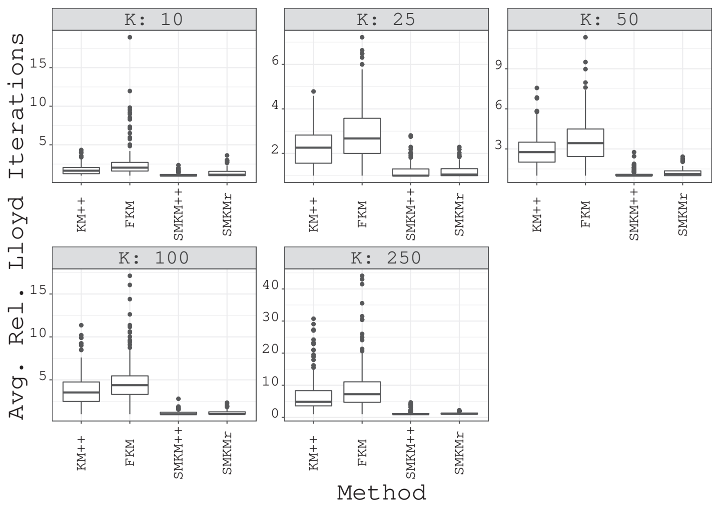

Fig. 6.3: Relative Lloyd it./re-start for SMKM and multi-start Lloyd's algorithm (boxplot).

Table 6.7: Relative Lloyd it./re-start (average over all datasets).

Method	K=10	K=25	K=50	K=100	K=250
FKM	2.59	2.86	3.53	4.60	8.76
KM++	1.75	2.23	2.78	3.68	6.59
SMKMr	1.33	1.18	1.22	1.15	1.21
SMKM++	1.12	1.19	1.12	1.13	1.26

On the other hand, in comparison to the Hartigan-based methods (HKMr and HKM++), their large number of distance computations is mainly related to the number of re-starts they execute until convergence, see Fig.6.4. In this sense, we would like to point out that each run of SMKM commonly implies a much larger reduction of the error in comparison to HKMr and HKM++, see Fig.6.5 and Table 6.5.

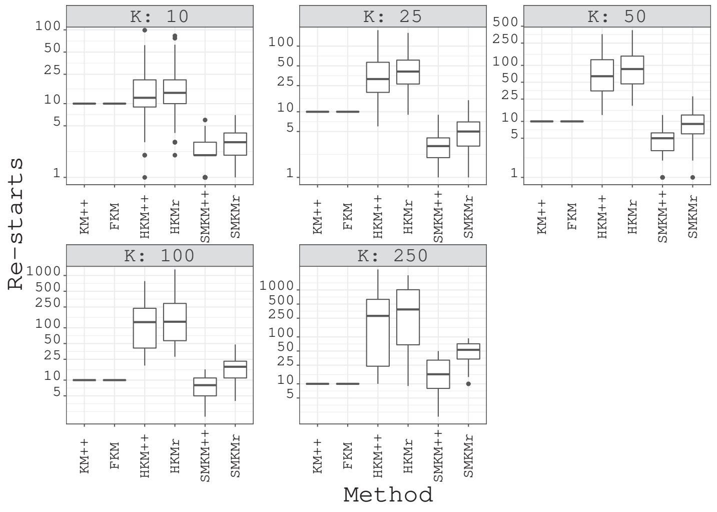

Fig. 6.4: Number of re-starts for each method (boxplot).

In Fig.6.5 and Table 6.5, we can observe that, regardless of the number of clusters, the relative error reduction reached by SMKM is, at least, one order of magnitude larger than that of HKM. For this reason, SMKM with fewer iterations, and therefore fewer distance computations, is able to obtain solutions of similar quality to those of HKM.

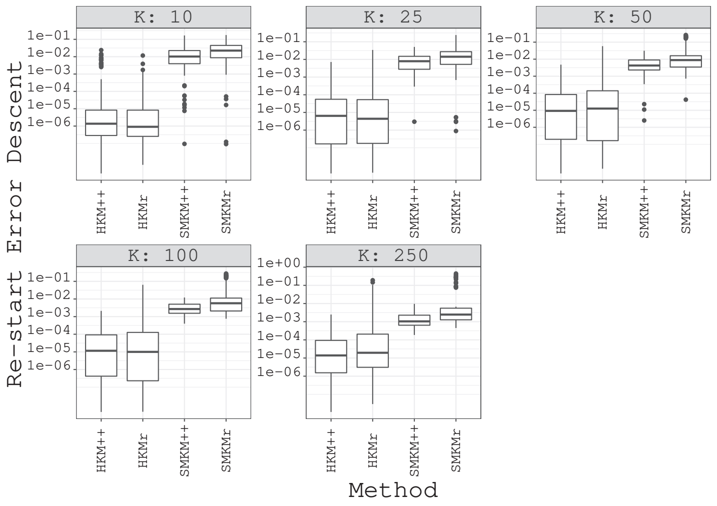

Fig. 6.5: Relative error descent per iteration of SMKM and HKM (boxplot).

6.2.3 Error Descent

Not only does the increase in the number of clusters imply a better trade-off between the quality of the solution vs. the number of distances computed for SMKM w.r.t. the other methods, but in this case we can also observe an increase in the number of cases in which our Split-Merge re-initialization generates an approximation with a lower error than that of the current local minima, see Fig.6.6 and Table 6.8.In particular, for $K = 250$, we can observe that in over 80% of the cases the quality of our re-initialization was, on average, better than that of the local optima solution that generated it. Moreover, observe that the Split-Merge process allowed the change of the basin of attraction in almost all the cases, see Table 6.8.

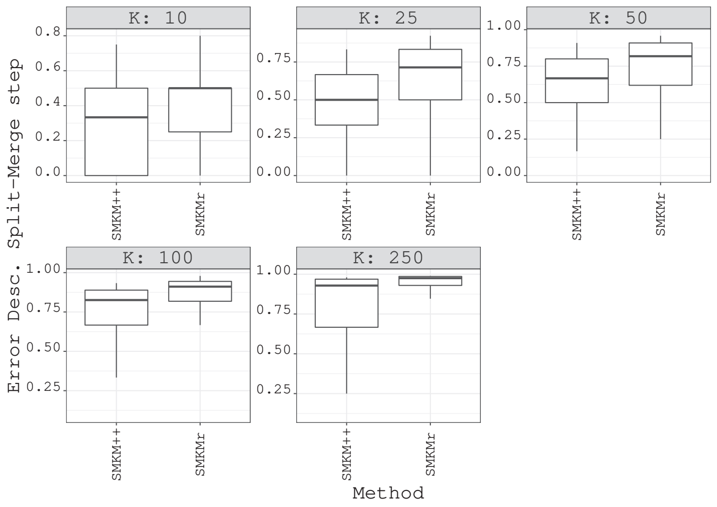

Fig. 6.6: Re-initialization error descent w.r.t. previous local minima (boxplot).

Such a behavior is expected as commented in Theorem 8, since, as we increase the number of clusters, we would expect the centers of mass to be closer one to another, and therefore the error increase given by `MergeStep` is also more likely to be smaller than the error decrease of `SplitStep`.

In general, we observe that, regardless of the initialization being used, SMKM is able to converge to highly competitive approximations to the K-means problem, while substantially reducing the computational load w.r.t. the

Table 6.8: Re-initialization error descent w.r.t. previous local minima/ Basin of attraction change (percentage).

Method	K=10	K=25	K=50	K=100	K=250
SMKMr	38.89% / 99.19%	63.20% / 99.78%	71.32% / 99.81%	86.09% / 99.96%	93.87% / 100.00%
SMKM++	27.14% / 98.92%	44.90% / 99.08%	59.12% / 99.63%	74.18% / 99.77%	81.64% / 99.96%

common practice of executing FKM and KM++ several times and HKM. Its performance is a direct consecuence of its re-initialization process, which, by only making a few modifications over a local minima of the K-means problem, is able to explore different basins of attractions while controlling the number of Lloyd's iterations to convergence. Not only that, but as the number of clusters increases, such a re-initialization is very likely to improve the quality of the approximation itself. As can be seen in Appendix .4.2, such a behavior can be seen consistently in all the data sets considered.

6.3 Conclusions

In this chapter, we propose a re-start process called Split-Merge K-means algorithm, or just SMK-means, which is an alternative to the typical multi-start K-means algorithm. Given a local minima of the K-means problem, SMK-means detects those regions that may have an excessive (over-represented) and insufficient (under-represented) amount of centroids. SMK-Means follows a two step re-initialization process: i) `SplitStep` divides the cluster from where the largest error reduction can be reached and generates two new centroids. ii) `MergeStep` condenses the pair of clusters (and their corresponding centroids), whose fusion produces the smallest increment in the error. `SplitStep` reduces the K-means error of our approximation, while `MergeStep` increases it, hence if the difference between both phases is negative, then the quality of the previous K-means local minima is already improved. SMK-means utilizes the information of previous Lloyd's algorithm runs to facilitate the convergence to a more competitive local minima. The proposed procedure has a computational complexity of $\mathcal{O}(\max\{n, K^2\} \cdot d)$, which is cheaper than a single Lloyd's algorithm iteration.

Experimentally, SMK-means was compared in both quality of the achieved local minima and number of distances computed w.r.t. competitive multi-start approaches (K-means++ and Forgy K-means, with a fixed number of restarts, and Hartigan K-means) on a wide variety of real-life data sets. In terms of the quality of the approximation, SMK-means consistently generated the local minima with the lowest K-means error, reducing, on average, over 1 and 2 orders of magnitude of relative error w.r.t. K-means++ and Hartigan K-means and Forgy K-means, respectively. Not only does the quality of the solution obtained by SMK-means tend to be much lower than the previously commented methods, but, in terms of computational resources, SMK-means also required a much lower number of distance computations (about an order of magnitude less) to reach the lowest errors that they achieved.

As for the next steps, we will analyze the effect of using the Split-Merge mechanism on different K-means speed-up strategies, such as those mentioned in the introduction, which are also susceptible to converging to poor local minima.

Part III

Final Remarks

7

General Conclusions and Future Work

In this chapter, both general conclusions of this PhD work and potential future paths to extend the contributions are presented.

7.1 Conclusions

Due to the progressive growth of the amount of data available in a wide variety of scientific fields, it has become more difficult to use and analyze such information. In this sense, cluster analysis algorithms are a key element of exploratory data analysis. Among these algorithms, the K-means algorithm stands out as the most popular approach, in spite of its high dependency on the initial conditions, as well as to the fact that it might not scale well on massive datasets. This PhD is devoted to developing efficient approximations for the K-means problem for such large data sets. The achieved contributions are listed as follows:

- Development of an approximation to the K-means algorithm for tall data applications: the Recursive Partition based K-means algorithm (RPKM). This technique consists of applying a weighted version of the K-means algorithm over a sequence of thinner partitions of the data set. Regardless of the partition strategy considered, we demonstrate different properties that emphasize the reduction of K-means iterations that occur at each iteration of this approach.

  Later on, we defined a partition strategy focused on placing more computational resources, in those areas of the space, in which the correct cluster assignment of the instances is uncertain. This process leads to the Boundary Weighted K-means algorithm (BWKM). Among different theoretical properties, such as those of any RPKM-type approach, BWKM tends to maximize the number of cells with instances belonging to the same cluster (well assigned blocks). This property ultimately leads its convergence to a fixed point of the K-means algorithm over the entire data set, but using a number of representatives which is usually much smaller than the actual number of instances of the data set. In the practice, BWKM was compared to fairly popular approaches, in terms of both the quality of the obtained solution for the K-means algorithm (K-means++ algorithm, Forgy K-means) and speed-ups of the K-means algorithm (Minibatch K-means algorithm, Markov Chain Monte Carlo K-means). The obtained results demonstrate that BWKM achieves massive reductions of distances computed in comparison to the aformentioned methods and, in multiple ocasions, actually converged to the sets of centroids with the lowest K-means errors.

- Design of a fully parallelizable feature selection approach for the K-means problem: the univariate K-means relevance for feature selection algorithm (KMR). The proposed method consists of applying any heuristic for the K-means problem on different disjoint subsets of the set of dimensions, of at most $m \ll d$ features, and using the obtained clustering to bound the error increase that occurs when not considering a given feature (when it is fixed to a given value). Those m features with the largest error increase bound are selected and used to approximate the optimal K-means clustering. For such a procedure, we provide different error guarantees for the obtained solution, as well as compare its performance to well-known feature selection techniques (Laplacian Scores, Maximum Variance, Random) and feature extraction tecniques (Singular Value Decomposition, Principal Component Analysis, Random Projections). In particular, it can be seen that KMR consistently obtains results with lower K-means error than all the considered feature selection methods, while also requiring similar or lower computational times than these approaches. In comparison to the feature extraction methods, KMR commonly required much lower computational times than all the considered techniques, while also converging to more competitive solutions than those generated via Random Projections.
- A final contribution is the development of a cheap Split-Merge heuristic that can be used to re-start the K-means algorithm, the Split-Merge K-means algorithm (SMKM). Given a local minima of the K-means problem, SMKM seeks to determine those regions of the space that are over-represented (regions of the space with more centroids than needed) and under-represented. To do so, in the first step of SMKM, a centroid is added in the cluster that generates the largest intra-cluster error. Afterwards, the pair of clusters that generates the lowest error increase, when merged, are combined into one cluster. We propose different conditions under which the previously described process reduces the error of the given fixed point without requiring any additional iteration of the K-means algorithm.

A wide variety of experimental results show that SMKM is able to generate approximations with an associated error that is hard to reach for different multi-start methods, such as multi-start Forgy K-means, K-

means++ and Hartigan K-means. In particular, SMKM consistently converged to approximations with the lowest K-means error, reducing, on average, over 1 and 2 orders of magnitude of relative error with respect to K-means++ and Hartigan K-means and Forgy K-means, respectively. Moreover, in terms of computational resources, SMKM also required a much lower number of distance computations (about an order of magnitude less) to reach the lowest errors among the aforementioned methods.

7.2 Future Work

As we have commented in the previous sections, most of the strategies that are proposed in this dissertation have different benefits (mostly computational) that could be further exploited to improve their performance. The potential future extensions of this PhD work are briefly summarized in the following paragraphs.

- One of the main advantages of all the methods proposed in this dissertation is that their parallelization is straightforward. For this reason, we plan to implement all these techniques in a parallel framework, such as *Apache Spark* [87, 88].
- As all the proposed algorithms make use of Lloyd's algorithm (with or without weights), we could further reduce the number of distances computed by considering different competitive distance pruning techniques, such as [24, 25, 26, 28].
- In spite of the competitiveness of the practical results of BWKM, its quality guarantees may be affected by the curse of dimensionality. In particular, as we increase the dimensionality of the problem, BWKM requires more iterations to construct a spatial partition of the data set with mostly well assigned blocks. In this sense, we plan to develop an approximation to the K-means problem that can be competitive on applications with both large number of instances and dimensions. To reach this goal, we plan to couple BWKM and KMR.
- In terms of the proposed dimensionality reduction technique, KMR, we plan to design a feature extraction approach of the same nature, in order to fully consider the information provided from each variable. More importantly, since KMR consists of solving multiple small-dimensional K-means problems, we would like to analyze the effect of using different coreset techniques, such as [22, 23, 78, 79, 27, 59, 60], to further reduce the computational requirements of our approach.
- In regard to the proposed re-start technique for the K-means algorithm, we plan to analyze the effect of using the Split-Merge re-start of SMKM on different K-means speed-up strategies, such as [22, 23, 78, 79, 27, 59, 60, 30], which are also susceptible to converging to poor local minima.

- In particular, since the re-initialization cost of SMKM is cheaper than a Lloyd's algorithm iteration, we could use this approach to re-start BWKM (whether or not coupled with KMR) to verify if the obtained approximation can be further improved. As we previously commented, since SMKM makes slight modifications on a given local minima of the K-means problem, the number of Lloyd's iterations needed to converge, after such a re-start, tends to be small. This is, we can generate even more competitive approximations, at the expense of a small amount of additional distance computations.
- There are multiple applications related to images, videos and sensor networks that can be interpreted as time series and/or streming signals that requires the analysis and decision-making in real time [89, 90, 91]. In this sense, we plan to adapt and develop techniques to perform partitional clustering on this kind of applications.
- The work developed in this dissertation is devoted to the K-means algorithms, which is one of the most popular algorithms in data mining. However, we plan to extend our contribution to other methods such as Support Vector Machines [92] and DBSCAN [93].

7.3 Main Achievements

7.3.1 Publications

- **M. Capó**, A. Pérez, and J. A. Lozano, "A recursive K-means initialization algorithm for massive data," *Proceedings of the Spanish Association for Artificial Intelligence*, pp. 929-938, 2015.
- **M. Capó**, A. Pérez, and J. A. Lozano, "An efficient approximation to the K-means clustering for massive data," *Knowledge-Based Systems*, vol. 117, pp. 56-69, 2017.
- **M. Capó**, A. Pérez, and J. A. Lozano, "An efficient K-means clustering algorithm for massive data," *Submitted to Data Mining and Knowledge Discovery*, 2019.
- **M. Capó**, A. Pérez, and J. A. Lozano, "An efficient Split-Merge re-start for the K-means algorithm," *Submitted to IEEE Transactions on Cybernetics*, 2018.
- **M. Capó**, A. Pérez, and J. A. Lozano, "A cheap feature selection approach for the K-means algorithm," *In preparation*, 2019.

7.3.2 Conferences and Workshops

The work presented in this dissertation has been presented in the following national and international conferences and workshops.

- XVI Conferencia de la Asociación Española para la Inteligencia Artificial, 12 November 2015, presentation entitled ”A recursive K-means initialization algorithm for massive data”, Albacete, Spain.
- 9^{th} NIPS Workshop on Optimization for Machine Learning, 10 December 2016, poster entitled ”A distance saving approach to the K-means problem for massive data”, Barcelona, Spain.
- 12^{th} International Symposium on Intelligent Distributed Computing, 17 October 2018, tutorial entitled ”The K-means algorithm on Big Data domains”, Bilbao, Spain.

7.3.3 Short Visits

- 01 September- 01 December 2018: LIAAD-INESC TEC, Porto University, Portugal. Supervisor: Prof. Dr. Joao Gama.

Appendices

.1 Appendix of Chapter 3

In this section, we present the proofs to all the theoretical results introduced in Chapter 3, referred to the RPKM algorithm.

Lemma 1 *Given a set of points X in $\mathbb{R}^d$ and a partition of it, $\mathcal{P}$. Then the function $f(\boldsymbol{c}) = |X| \cdot \|\overline{X} - \mathbf{c}\|^2 - \sum_{R \in \mathcal{P}} |R| \cdot \|\overline{R} - \mathbf{c}\|^2$ is constant.*

Proof. First of all observe that $|X| = \sum_{R \in \mathcal{P}} |R|$ and $\overline{X} = \frac{\sum_{R \in \mathcal{P}} |R| \cdot \overline{R}}{|X|}$. Given any pair $\mathbf{c}$, $\mathbf{c}^{'} \in \mathbb{R}^d$, we have that

$$\begin{aligned}
f(\mathbf{c}) &= |X| \cdot \|\overline{X} - \mathbf{c}\|^2 - \sum_{R \in \mathcal{P}} |R| \cdot \|\overline{R} - \mathbf{c}\|^2 \\
&= |X| \cdot (\|\overline{X} - \mathbf{c}^{'}\|^2 + 2 \cdot (\mathbf{c}^{'} - \mathbf{c})^t (\overline{X} - \mathbf{c}^{'}) + \|\mathbf{c}^{'} - \mathbf{c}\|^2) \\
&\quad - \sum_{R \in \mathcal{P}} |R| \cdot (\|\overline{R} - \mathbf{c}^{'}\|^2 + 2 \cdot (\mathbf{c}^{'} - \mathbf{c})^t (\overline{R} - \mathbf{c}^{'}) + \|\mathbf{c}^{'} - \mathbf{c}\|^2) \\
&= f(\mathbf{c}^{'}) + 2 \cdot |X| \cdot (\mathbf{c}^{'} - \mathbf{c})^t (\overline{X} - \mathbf{c}^{'}) - 2 \cdot \sum_{R \in \mathcal{P}} |R| \cdot (\mathbf{c}^{'} - \mathbf{c})^t (\overline{R} - \mathbf{c}^{'}) \\
&= f(\mathbf{c}^{'})
\end{aligned}$$

Lemma 2 *Let $\mathcal{P}$ and $\mathcal{P}'$ be two partitions of the data set X, with $\mathcal{P} \succ \mathcal{P}^{'}$, and let $\mathcal{G}$ and $\mathcal{G}'$ be two clusterings of $\mathcal{P}$. Then, the difference between both clustering errors is constant with respect to the partitions $\mathcal{P}$ and $\mathcal{P}'$:*

$$E_{\mathcal{P}}(\mathcal{G}) - E_{\mathcal{P}}(\mathcal{G}^{'}) = E_{\mathcal{P}'}(\mathcal{G}) - E_{\mathcal{P}'}(\mathcal{G}^{'})$$

Proof. Let the index of $S \in \mathcal{P}$ on the cluster $\mathcal{G} = \{G_k\}_{k=1}^K$, $i(S, \mathcal{G})$, be defined as $i(S, \mathcal{G}) = \{k \mid S \subseteq G_k\}$, then

$$\begin{aligned}
E_{\mathcal{P}}(\mathcal{G}) - E_{\mathcal{P}'}(\mathcal{G}) &= \sum_{S\in\mathcal{P}} |S|\cdot\|\overline{S}-\overline{G_{i(S,\mathcal{G})}}\|^2 - \sum_{S'\in\mathcal{P}'} |S^{'}|\cdot\|\overline{S'}-\overline{G_{i(S',\mathcal{G})}}\|^2\\
&= \sum_{S\in\mathcal{P}} (|S|\cdot\|\overline{S}-\overline{G_{i(S,\mathcal{G})}}\|^2 - \sum_{R\in\mathcal{P}'[S]} |R|\cdot\|\overline{R}-\overline{G_{i(R,\mathcal{G})}}\|^2)\\
&= \sum_{S\in\mathcal{P}} (|S|\cdot\|\overline{S}-\overline{G_{i(S,\mathcal{G})}}\|^2 - \sum_{R\in\mathcal{P}'[S]} |R|\cdot\|\overline{R}-\overline{G_{i(S,\mathcal{G})}}\|^2)\\
&= \sum_{S\in\mathcal{P}} (|S|\cdot\|\overline{S}-\overline{G^{'}_{i(S,\mathcal{G}')}}\|^2 - \sum_{R\in\mathcal{P}'[S]} |R|\cdot\|\overline{R}-\overline{G^{'}_{i(S,\mathcal{G}')}}\|^2)\\
&= \sum_{S\in\mathcal{P}} (|S|\cdot\|\overline{S}-\overline{G^{'}_{i(S,\mathcal{G}')}}\|^2 - \sum_{R\in\mathcal{P}'[S]} |R|\cdot\|\overline{R}-\overline{G^{'}_{i(R,\mathcal{G}')}}\|^2)\\
&= \sum_{S\in\mathcal{P}} |S|\cdot\|\overline{S}-\overline{G^{'}_{i(S,\mathcal{G}')}}\|^2 - \sum_{S'\in\mathcal{P}'} |S^{'}|\cdot\|\overline{S'}-\overline{G^{'}_{i(S',\mathcal{G}')}}\|^2\\
&= E_{\mathcal{P}}(\mathcal{G}^{'}) - E_{\mathcal{P}'}(\mathcal{G}^{'})
\end{aligned}$$

Before prooving Corollary 1 and Lemma 3, we analyze the error of a cluster with respect to a weighted K-means iteration. We observe that following inequality is satisfied at the r-th weighted Lloyd's algorithm iteration:

$$E_{\mathcal{P}}(C_r) \geq E_{\mathcal{P}}(\mathcal{G}_r) \geq E_{\mathcal{P}}(C_{r+1}) \tag{.1}$$

Furthermore, observe that, if $E_{\mathcal{P}}(C_r) = E_{\mathcal{P}}(\mathcal{G}_r)$, then, after the update step of the weighted Lloyd's algorithm, we obtain $C_r = C_{r+1}$ and the algorithm stops at the $(r+1)$-th iteration. On the other hand, if $E_{\mathcal{P}}(C_r) > E_{\mathcal{P}}(\mathcal{G}_r) = E_{\mathcal{P}}(C_{r+1})$, then, in the following weighted Lloyd's algorithm iteration, we obtain $E_{\mathcal{P}}(C_{r+1}) = E_{\mathcal{P}}(\mathcal{G}_{r+1}) = E_{\mathcal{P}}(C_{r+2})$ and $C_{r+1} = C_{r+2}$, hence the algorithm stops, at most, at the $(r+2)$-th iteration.

Hence, we have the following chain of inequalities for any weighted Lloyd's algorithm run

$$\begin{aligned}E_{\mathcal{P}}(C_0) > E_{\mathcal{P}}(\mathcal{G}_0) > E_{\mathcal{P}}(C_1) > E_{\mathcal{P}}(\mathcal{G}_1) > E_{\mathcal{P}}(C_2) >\\ \ldots > E_{\mathcal{P}}(\mathcal{G}_{l-2}) \geq E_{\mathcal{P}}(C_{l-1}) \geq E_{\mathcal{P}}(\mathcal{G}_{l-1}) \geq E_{\mathcal{P}}(C_l),\end{aligned} \tag{.2}$$

where l is the total number of weighted Lloyd iterations.

Theorem 3 *Given a data set, X, a set of K centroids C and a spatial partition $\mathcal{B}$ of the data set X, the following inequality is satisfied:*

$$|E_X(C) - E_{\mathcal{P}}(C)| \leq \sum_{B\in\mathcal{B}} 2\cdot|P|\cdot\epsilon_{C,X}(B)\cdot(2\cdot l_B + \|\overline{P}-\boldsymbol{c}_{\overline{P}}\|) + \frac{|P|-1}{2}\cdot l_B^2,$$

where $P = B(X)$ and $\mathcal{P} = \mathcal{B}(X)$. Furthermore, for a well assigned partition $\mathcal{P}$, if we define $C^{\mathcal{P}}_{OPT} = \underset{C\subset\mathbb{R}^d,|C|=K}{\arg\min} E_{\mathcal{P}}(C)$ and $C_{OPT} = \underset{C\subset\mathbb{R}^d,|C|=K}{\arg\min} E_X(C)$, then

$$E_X(C^{\mathcal{P}}_{OPT}) \leq E_X(C_{OPT}) + (n - |\mathcal{P}|) \cdot l^2,$$

where $l = \max_{B\in\mathcal{B}} l_B$. [1]

Proof. From Lemma 2 and Eq..2, we have the following inequalities:

$$E_{\mathcal{P}_i}(\mathcal{G}^{i-1}_{l_{i-1}-1}) \geq E_{\mathcal{P}_i}(C_{i-1}) \geq E_{\mathcal{P}_i}(\mathcal{G}^{i}_{l_i-1}) \geq E_{\mathcal{P}_i}(C_i) \qquad (.3)$$

$$E_{\mathcal{P}_i}(\mathcal{G}^{i-1}_{l_{i-1}-1}) - E_{\mathcal{P}_i}(\mathcal{G}^{i}_{l_i-1}) = E_{\mathcal{P}_j}(\mathcal{G}^{i-1}_{l_{i-1}-1}) - E_{\mathcal{P}_j}(\mathcal{G}^{i}_{l_i-1}) = \xi_i \geq 0 \qquad (.4)$$

Eq..3 holds for $i \in \{1,\ldots,m\}$ and Eq..4 for any $j > i$. From Eq..4, we can see that the difference between both clusterings remain constant for any thinner partition $\mathcal{P}_j$. A consequence of this is that, if we take $\mathcal{P}_j$ as a partition thin enough such that there is only one instance per subset, then adding the difference clustering error (associated to both centroids) for the partitions $\mathcal{P}_j$ and $\mathcal{P}_i$, we have the following relation over the error for the entire data set (observe that the following relation holds in general for any partition thinner than $\mathcal{P}_i$):

$$E_X(C_i) \leq E_X(C_{i-1}) \Longleftrightarrow E_X(\mathcal{G}^{i-1}_{l_{i-1}-1}) - E_X(C_{i-1}) \leq \xi_i + (E_X(\mathcal{G}^{i}_{l_i-1}) - E_X(C_i))$$

Lemma 3 *At the i-th step of the RPKM, if $\mathcal{G}^i_r = \mathcal{G}^j_s$, with $j < i$, for some $r \in \{1,\ldots,l_i - 1\}$ and $s \in \{1,\ldots,l_j - 1\}$, then $l_{j+1} = \ldots = l_i = 1$. Moreover, in that case, $s = l_j - 1$.*

Proof. Using the chain of inequalities (.2), we observe that the following inequalities hold at the first iteration of the RPKM:

$$\begin{aligned} &E_{\mathcal{P}_1}(C^1_0 = C_0) > E_{\mathcal{P}_1}(\mathcal{G}^1_0) > E_{\mathcal{P}_1}(C^1_1) > E_{\mathcal{P}_1}(\mathcal{G}^1_1) > E_{\mathcal{P}_1}(C^1_2) > \\ &\ldots > E_{\mathcal{P}_1}(\mathcal{G}^1_{a_1-2}) \geq E_{\mathcal{P}_1}(C^1_{l_1-1}) \geq E_{\mathcal{P}_1}(\mathcal{G}^1_{l_1-1}) \geq E_{\mathcal{P}_1}(C_1 = C^1_{l_1}) \end{aligned} \qquad (.5)$$

Analogously, for the subsequent i-th iteration of the RPKM algorithm, we obtain the following set of inequalities

$$\begin{aligned} E_{\mathcal{P}_i}(\mathcal{G}^{i-1}_{a_{i-1}-1}) > E_{\mathcal{P}_i}(C^i_0 = C_{i-1}) > E_{\mathcal{P}_i}(\mathcal{G}^i_0) > E_{\mathcal{P}_i}(C^i_1) > E_{\mathcal{P}_i}(\mathcal{G}^i_1) > \\ E_{\mathcal{P}_i}(C^i_2) > \ldots > E_{\mathcal{P}_i}(\mathcal{G}^i_{l_i-2}) \geq E_{\mathcal{P}_i}(C^i_{l_i-1}) \geq E_{\mathcal{P}_i}(\mathcal{G}^i_{l_i-1}) \\ \geq E_{\mathcal{P}_i}(C_i = C^i_{l_i}),\ i \in \{2,\ldots,m\} \end{aligned} \qquad (.6)$$

[1] The proof of Theorem 3 is in Appendix .2.1 in the supplementary material.

First of all, observe that the error associated to all the clusters generated at the j-th RPKM iteration keep the same ordering for the error associated to a thinner partition $\mathcal{P}_i$. In particular, we have $E_{\mathcal{P}_i}(\mathcal{G}^j_{l_j-1}) \leq E_{\mathcal{P}_i}(\mathcal{G}^j_s)$ for $s < l_j - 1$. To verify this we make use of Lemma 2, from which we know that $E_{\mathcal{P}_i}(\mathcal{G}^j_{l_j-1}) - E_{\mathcal{P}_i}(\mathcal{G}^j_s) = E_{\mathcal{P}_j}(\mathcal{G}^j_{l_j-1}) - E_{\mathcal{P}_j}(\mathcal{G}^j_s) \leq 0 \rightarrow E_{\mathcal{P}_i}(\mathcal{G}^j_{l_j-1}) \leq E_{\mathcal{P}_i}(\mathcal{G}^j_s)$ for $s < l_j - 1$. This means that, the last clustering found at the j-th RPKM iteration also has the smallest error, with the partition $\mathcal{P}_i$, with respect to the previous clusters obtained at the j-th RPKM iteration.

From the chain of inequalities (.5) and (.6), we know that

$$E_{\mathcal{P}_h}(\mathcal{G}^{h-1}_{l_{h-1}-1}) \geq E_{\mathcal{P}_h}(\mathcal{G}^h_{l_h-1}) \; \forall h \in \{j+1, \dots, i-1\}, \tag{.7}$$

where, if the equality holds, then $a_h = 1$. From Lemma 2, (.7) implies

$$E_{\mathcal{P}_i}(\mathcal{G}^j_{l_j-1}) \geq E_{\mathcal{P}_i}(\mathcal{G}^{j+1}_{l_{j+1}-1}) \geq \dots \geq E_{\mathcal{P}_i}(\mathcal{G}^{i-1}_{l_{i-1}-1}) \tag{.8}$$

In other words, the error with respect to the current partition (or any thinner partition) of the optimal patterns obtained at each RPKM iteration decreases monotonically.

By *reductio ad absurdum*, if we assume that $\mathcal{G}^i_r = \mathcal{G}^j_s$, for some $r \in \{1, \dots, l_i - 1\}$ and $s \in \{1, \dots, l_j - 1\}$ and that there exists $j < h < i$ such that $l_h > 1$, then $E_{\mathcal{P}_i}(\mathcal{G}^j_{l_j-1}) > E_{\mathcal{P}_i}(\mathcal{G}^{i-1}_{l_{i-1}-1}) \Rightarrow E_{\mathcal{P}_i}(\mathcal{G}^j_s) > E_{\mathcal{P}_i}(\mathcal{G}^i_r) = E_{\mathcal{P}_i}(\mathcal{G}^j_s)$ ($\Rightarrow\Leftarrow$). Therefore, from now on we assume $l_{j+1} = \dots = l_{i-1} = 1$.

Analogously, if we assume that $\mathcal{G}^i_r = \mathcal{G}^j_s$, for some $r \in \{1, \dots, l_i - 1\}$ and $s \in \{1, \dots, l_j - 1\}$ and that $l_i > 1$, then $E_{\mathcal{P}_i}(\mathcal{G}^j_{l_j-1}) \geq E_{\mathcal{P}_i}(\mathcal{G}^{i-1}_{l_{i-1}-1}) > E_{\mathcal{P}_i}(\mathcal{G}^i_r) = E_{\mathcal{P}_i}(\mathcal{G}^j_s) \Rightarrow E_{\mathcal{P}_i}(\mathcal{G}^j_s) > E_{\mathcal{P}_i}(\mathcal{G}^j_s)$ ($\Rightarrow\Leftarrow$).

In the case that $l_{j+1} = \dots = l_i = 1$, the error associated to $\mathcal{G}^i_r$ satisfies $E_{\mathcal{P}_i}(\mathcal{G}^j_s) \geq E_{\mathcal{P}_i}(\mathcal{G}^j_{l_j-1}) \geq E_{\mathcal{P}_i}(\mathcal{G}^{i-1}_{l_{i-1}-1}) \geq E_{\mathcal{P}_i}(\mathcal{G}^i_r)$, hence the only possible choice is $s = l_j$.

Theorem 1 *An upper bound to the number of Lloyd iterations at the i-th RPKM step is given by* $l_i \leq \left\{ {|\mathcal{P}_i| \atop K} \right\} - \sum\limits_{j=1}^{i-1}(l_j - 1)$, *where* $\left\{ {|\mathcal{P}_i| \atop K} \right\}$ *is a Stirling number of the second kind.*

Proof. Lemma 3 implies that, at each RPKM iteration, no previous clustering can be repeated (except the last one from the clustering sequence). Therefore, we can eliminate, at least, all clusters generated at the previous RPKM iterations except the last one ($\sum\limits_{j=1}^{i-1}(l_j - 1)$). Moreover, the number of different partitions for $|\mathcal{P}_i|$ observations into K groups is a Stirling number of the second kind, $\left\{ {|\mathcal{P}_i| \atop K} \right\}$ [34], then $l_i \leq \left\{ {|\mathcal{P}_i| \atop K} \right\} - \sum\limits_{j=1}^{i-1}(l_j - 1)$.

.2 Appendix of Chapter 4

In this section, we comment on the proofs to the theoretical results presented in Chapter 4 and an extension to the empirical results in Section 4.2.

.2.1 Proofs

In the first result, we present a complimentary property of the grid based RPKM proposed in Chapter 3. Each iteration of the RPKM can be proved to be a coreset with an exponential decrease in the error with respect to the number of iterations. This result could actually bound the BWKM error, if we fix i as the minimum number of cuts that a block, of a certain partition generated by BWKM, $\mathcal{P}$, has.

Theorem 9 *Given a set of points X in $\mathbb{R}^d$, the i-th iteration of the grid based RPKM produces a (K,ε)-coreset with $\varepsilon = \frac{1}{2^{i-1}} \cdot (1 + \frac{1}{2^{i+2}} \cdot \frac{n-1}{n}) \cdot \frac{n \cdot l^2}{OPT}$, where $OPT = \min\limits_{C \subseteq \mathbb{R}^d, |C|=K} E_X(C)$ and l the length of the diagonal of the smallest bounding box containing X.*

Proof. Firstly, we denote by $\mathbf{x}'$ to the representative of $\mathbf{x} \in D$ at the i-th grid based RPKM iteration, i.e., if $\mathbf{x} \in P$ then $\mathbf{x}' = \overline{P}$, where P is a block of the corresponding dataset partition $\mathcal{P}$ of X. Observe that, at the i-th grid based RPKM iteration, the length of the diagonal of each cell is $\frac{1}{2^i} \cdot l$ and we set a positive constant, c, as the positive real number satisfying $\frac{1}{2^i} \cdot l = \sqrt{c \cdot \frac{OPT}{n}}$. By the triangular inequality, we have

$$
\begin{aligned}
|E_X(C) - E_{\mathcal{P}}(C)| &\leq \sum_{\mathbf{x} \in X} |\|\mathbf{x} - \mathbf{c}_{\mathbf{x}}\|^2 - \|\mathbf{x}' - \mathbf{c}_{\mathbf{x}'}\|^2| \\
&\leq \sum_{\mathbf{x} \in X} |(\|\mathbf{x} - \mathbf{c}_{\mathbf{x}}\| - \|\mathbf{x}' - \mathbf{c}_{\mathbf{x}'}\|)(\|\mathbf{x} - \mathbf{c}_{\mathbf{x}}\| + \|\mathbf{x}' - \mathbf{c}_{\mathbf{x}'}\|)|
\end{aligned}
$$

Analogously, observe that the following inequalities hold $\|\mathbf{x}' - \mathbf{c}_{\mathbf{x}'}\| + \|\mathbf{x} - \mathbf{x}'\| \geq \|\mathbf{x} - \mathbf{c}_{\mathbf{x}}\|$ and $\|\mathbf{x} - \mathbf{c}_{\mathbf{x}}\| + \|\mathbf{x} - \mathbf{x}'\| \geq \|\mathbf{x}' - \mathbf{c}_{\mathbf{x}'}\|$. Thus, $\|\mathbf{x} - \mathbf{x}'\| \geq |\|\mathbf{x} - \mathbf{c}_{\mathbf{x}}\| - \|\mathbf{x}' - \mathbf{c}_{\mathbf{x}'}\||$:

$$
|E_X(C) - E_{\mathcal{P}}(C)| \leq \sum_{\mathbf{x} \in X} \|\mathbf{x} - \mathbf{x}'\| \cdot (2 \cdot \|\mathbf{x} - \mathbf{c}_{\mathbf{x}}\| + \|\mathbf{x} - \mathbf{x}'\|)
$$

On the other hand, we know that $\sum\limits_{\mathbf{x} \in X} \|\mathbf{x} - \mathbf{x}'\|^2 \leq \frac{n-1}{2^{2i+1}} \cdot l^2$ and that, as both $\mathbf{x}$ and $\mathbf{x}'$ must be located in the same cell, $\|\mathbf{x} - \mathbf{x}'\| \leq \frac{1}{2^i} \cdot l$. Therefore, as $\mathbf{d}(\mathbf{x}, C) \leq l$,

$$\begin{aligned}|E_X(C) - E_{\mathcal{P}}(C)| &\leq (\frac{n-1}{2^{2i+1}} + \frac{n}{2^{i-1}}) \cdot l^2 \\ &\leq (\frac{n-1}{2^{2i+1}} + \frac{n}{2^{i-1}}) \cdot 2^{2i} \cdot c \cdot \frac{OPT}{n} \\ &\leq (\frac{1}{2^{i+2}} \cdot \frac{n-1}{n} + 1) \cdot 2^{i+1} \cdot c \cdot E_X(C)\end{aligned}$$

In other words, the i-th RPKM iteration is a (K, ε)- coreset with $\varepsilon = (\frac{1}{2^{i+2}} \cdot \frac{n-1}{n} + 1) \cdot 2^{i+1} \cdot c = \frac{1}{2^{i-1}} \cdot (1 + \frac{1}{2^{i+2}} \cdot \frac{n-1}{n}) \cdot \frac{n \cdot l^2}{OPT}$.

The two following results show some properties of the error function when having well assigned blocks.

Lemma 4 *If $\boldsymbol{c_x} = \boldsymbol{c}_{\overline{P}}$ and $\boldsymbol{c}'_{\boldsymbol{x}} = \boldsymbol{c}'_{\overline{P}}$ for all $\boldsymbol{x} \in P$, where $P \subseteq D$ and C, C' are a pair of sets of centroids, then $E_P(C) - E_{\{P\}}(C) = E_P(C') - E_{\{P\}}(C')$.*

Proof. From Lemma 1 in Chapter 3, we can say that the following function is constant $f(\mathbf{c}) = |P| \cdot \|\overline{P} - \mathbf{c}\|^2 - \sum_{\mathbf{x} \in P} \|\mathbf{x} - \mathbf{c}\|^2$, for $\mathbf{c} \in \mathbb{R}^d$. In particular, since $f(\overline{P}) = -\sum_{\mathbf{x} \in P} \|\mathbf{x} - \overline{P}\|^2$, we have that $|P| \cdot \|\overline{P} - \mathbf{c}_{\overline{P}}\|^2 = \sum_{\mathbf{x} \in P} \|\mathbf{x} - \mathbf{c}_{\overline{P}}\|^2 - \sum_{\mathbf{x} \in P} \|\mathbf{x} - \overline{P}\|^2$ and so we can express the weighted error of a dataset partition, $\mathcal{P}$, as follows

$$E_{\mathcal{P}}(C) = \sum_{P \in \mathcal{P}} \sum_{\mathbf{x} \in P} (\|\mathbf{x} - \mathbf{c}_{\overline{P}}\|^2 - \|\mathbf{x} - \overline{P}\|^2) \tag{.9}$$

In particular, for $P \in \mathcal{P}$, we have

$$\begin{aligned}E_P(C) - E_{\{P\}}(C) &= \sum_{\mathbf{x} \in P} (\|\mathbf{x} - \mathbf{c_x}\|^2 - \|\mathbf{x} - \mathbf{c}_{\overline{P}}\|^2 + \|\mathbf{x} - \overline{P}\|^2) \\ &= \sum_{\mathbf{x} \in P} \|\mathbf{x} - \overline{P}\|^2 \\ &= \sum_{\mathbf{x} \in P} (\|\mathbf{x} - \mathbf{c}'_{\mathbf{x}}\|^2 - \|\mathbf{x} - \mathbf{c}'_{\overline{P}}\|^2 + \|\mathbf{x} - \overline{P}\|^2) \\ &= E_P(C') - E_{\{P\}}(C')\end{aligned}$$

In the previous result we observe that, if all the instances are correctly assigned in each block, then the difference of the weighted and the entire dataset error, of both sets of centroids, is the same. In other words, if all the blocks of a given partition are correctly assigned, not only can we then actually guarantee a monotone descend of the entire error function for our approximation, a property that can not be guaranteed for the typical coreset type approximations of K-means, but we know exactly the reduction of such an error after a weighted Lloyd iteration.

Theorem 10 *Given two set of centroids C, C', where C' is obtained after a weighted Lloyd's iteration (on a partition $\mathcal{P}$) over C and $\boldsymbol{c}_{\boldsymbol{x}} = \boldsymbol{c}_{\overline{P}}$ and $\boldsymbol{c}'_{\boldsymbol{x}} = \boldsymbol{c}'_{\overline{P}}$ for all $\boldsymbol{x} \in P$ and $P \in \mathcal{P}$, then $E_X(C') \leq E_X(C)$.*

Proof. Using Lemma 4 over all the subsets $P \in \mathcal{P}$, we know that $E_X(C') - E_X(C) = \sum_{P \in \mathcal{P}}(E_P(C') - E_P(C)) = \sum_{P \in \mathcal{P}}(E_{\{P\}}(C') - E_{\{P\}}(C)) = E_{\mathcal{P}}(C') - E_{\mathcal{P}}(C)$. Moreover, from the chain of inequalities $A.1$ in Chapter 3, we know that $E_{\mathcal{P}}(C') \leq E_{\mathcal{P}}(C)$ at any weighted Lloyd iteration over a given partition $\mathcal{P}$, thus $E_X(C') \leq E_X(C)$.

In Theorem 2, we prove the cutting criterion that we use in BWKM. It consists of an inequality that, only by using information referred to the partition of the dataset and the weighted Lloyd's algorithm, helps us guarantee that a block is well assigned.

Theorem 2 *Given a set of K centroids, C, a data set, $X \subseteq \mathbb{R}^d$, and a block B, if $\epsilon_{C,X}(B) = 0$, then $\boldsymbol{c}_{\boldsymbol{x}} = \boldsymbol{c}_{\overline{P}}$ for all $\boldsymbol{x} \in P = B(X) \neq \emptyset$.*

Proof. From the triangular inequality, we know that $\|\mathbf{x} - \mathbf{c}_{\overline{P}}\| \leq \|\mathbf{x} - \overline{P}\| + \|\overline{P} - \mathbf{c}_{\overline{P}}\|$. Moreover, observe that $\overline{P}$ is contained in the block B, since B is a convex polygon. Then $\|\mathbf{x} - \overline{P}\| \leq l_B$.

For this reason, $\|\mathbf{x}-\mathbf{c}_{\overline{P}}\| \leq l_B - \delta_P(C) + \|\overline{P} - \mathbf{c}\| \leq (2 \cdot l_B - \delta_P(C)) + \|\mathbf{x}-\mathbf{c}\|$ holds. As $\epsilon_{C,X}(B) = \max\{0, 2 \cdot l_B - \delta_P(C)\} = 0$, then $2 \cdot l_B - \delta_P(C) \leq 0$ and, therefore, $\|\mathbf{x}-\mathbf{c}_{\overline{P}}\| \leq \|\mathbf{x}-\mathbf{c}\|$ for all $\mathbf{c} \in C$. In other words, $\mathbf{c}_{\overline{P}} = \arg\min_{\mathbf{c} \in C} \|\mathbf{x}-\mathbf{c}\|$ for all $\mathbf{x} \in P$.

As can be seen in Section 4.1.2, there are different parameters that must be tuned. In the following result, we set a criterion to choose the initialization parameters of Algorithm 5 in a way that its complexity, even in the worst case scenario, is still the same as that of Lloyd's algorithm.

Theorem 11 *Given an integer r, if $m = \mathcal{O}(\sqrt{K \cdot d})$ and $s = \mathcal{O}(\sqrt{n})$, then Algorithm 5 is $\mathcal{O}(n \cdot K \cdot d)$.*

Proof. It is enough to verify the conditions presented before. Firstly, observe that $r \cdot s \cdot m^2 = \mathcal{O}(\sqrt{n} \cdot K \cdot d)$ and $n \cdot m = \mathcal{O}(n \cdot \sqrt{K \cdot d})$. Moreover, as $K \cdot d = \mathcal{O}(n)$, then $r \cdot m^2 = \mathcal{O}(n)$.

Up to this point, most of the quality results assume the case when all the blocks are well assigned. However, in order to achieve this, many BWKM iterations might be required. In the following result, we provide a bound to the weighted error with respect to the full error. This result shows that our weighted representation improves as more blocks of our partition satisfy the criterion in Algorithm 2 and/or the diagonal of the blocks are smaller.

Theorem 3 *Given a data set, X, a set of K centroids C and a spatial partition $\mathcal{B}$ of the data set X, the following inequality is satisfied:*

$$|E_X(C) - E_{\mathcal{P}}(C)| \le \sum_{B \in \mathcal{B}} 2 \cdot |P| \cdot \epsilon_{C,X}(B) \cdot (2 \cdot l_B + \|\overline{P} - \mathbf{c}_{\overline{P}}\|) + \frac{|P|-1}{2} \cdot l_B^2,$$

where $P = B(X)$ *and* $\mathcal{P} = \mathcal{B}(X)$*. Furthermore, for a well assigned partition* $\mathcal{P}$*, if we define* $C_{OPT}^{\mathcal{P}} = \underset{C \subset \mathbb{R}^d, |C|=K}{\arg\min} E_{\mathcal{P}}(C)$ *and* $C_{OPT} = \underset{C \subset \mathbb{R}^d, |C|=K}{\arg\min} E_X(C)$*, then*

$$E_X(C_{OPT}^{\mathcal{P}}) \le E_X(C_{OPT}) + (n - |\mathcal{P}|) \cdot l^2,$$

where $l = \max_{B \in \mathcal{B}} l_B$*.*

Proof. Using Eq. 9 in Theorem 4, we know that $|E_X(C) - E_{\mathcal{P}}(C)| \le \sum_{P \in \mathcal{P}} \sum_{\mathbf{x} \in P} \|\mathbf{x} - \mathbf{c}_{\overline{P}}\|^2 - \|\mathbf{x} - \mathbf{c}_{\mathbf{x}}\|^2 + \|\mathbf{x} - \overline{P}\|^2$.

Observe that, for a certain instance $\mathbf{x} \in P$, where $\epsilon_{C,X}(B) = \max\{0, 2 \cdot l_B - \delta_P(C)\} = 0$, $\|\mathbf{x} - \mathbf{c}_{\overline{P}}\|^2 - \|\mathbf{x} - \mathbf{c}_{\mathbf{x}}\|^2 = 0$, as $\mathbf{c}_{\mathbf{x}} = \mathbf{c}_{\overline{P}}$ by Theorem 2. On the other hand, if $\epsilon_{C,X}(B) > 0$, we have the following inequalities:

$$\begin{aligned} \|\mathbf{x} - \mathbf{c}_{\overline{P}}\| - \|\mathbf{x} - \mathbf{c}_{\mathbf{x}}\| &\le 2 \cdot \|\mathbf{x} - \overline{P}\| - (\|\overline{P} - \mathbf{c}_{\mathbf{x}}\| - \|\overline{P} - \mathbf{c}_{\overline{P}}\|) \\ &\le \epsilon_{C,X}(B) \end{aligned}$$

$$\begin{aligned} \|\mathbf{x} - \mathbf{c}_{\overline{P}}\| + \|\mathbf{x} - \mathbf{c}_{\mathbf{x}}\| &\le 2 \cdot \|\mathbf{x} - \overline{P}\| + \|\overline{P} - \mathbf{c}_{\mathbf{x}}\| + \|\overline{P} - \mathbf{c}_{\overline{P}}\| \\ &< 2 \cdot l_B + (2 \cdot l_B + \|\overline{P} - \mathbf{c}_{\overline{P}}\|) \\ &+ \|\overline{P} - \mathbf{c}_{\overline{P}}\| \\ &= 2 \cdot (2 \cdot l_B + \|\overline{P} - \mathbf{c}_{\overline{P}}\|) \end{aligned}$$

Using both inequalities, we have $\|\mathbf{x} - \mathbf{c}_{\overline{P}}\|^2 - \|\mathbf{x} - \mathbf{c}_{\mathbf{x}}\|^2 \le 2 \cdot \epsilon_{C,X}(B) \cdot (2 \cdot l_B + \|\overline{P} - \mathbf{c}_{\overline{P}}\|)$. On the other hand, observe that $\sum_{\mathbf{x} \in P} \|\mathbf{x} - \overline{P}\|^2 = \frac{1}{|P|} \cdot \sum_{\mathbf{x}, \mathbf{y} \in P} \|\mathbf{x} - \mathbf{y}\|^2 \le \frac{1}{|P|} \cdot \frac{|P| \cdot (|P|-1)}{2} \cdot l_B^2 = \frac{|P|-1}{2} \cdot l_B^2$.

Furthermore, if the partition is well assigned, then $\epsilon_{C,X}(B) = 0$ for all $B \in \mathcal{B}$ and so,

$$\begin{aligned} E_X(C_{OPT}^{\mathcal{P}}) &\le E_{\mathcal{P}}(C_{OPT}^{\mathcal{P}}) + \sum_{B \in \mathcal{B}} \frac{|P|-1}{2} \cdot l_B^2 \\ &\le E_X(C_{OPT}) + 2 \cdot \sum_{B \in \mathcal{B}} \frac{|P|-1}{2} \cdot l_B^2 \\ &\le E_X(C_{OPT}) + (n - |\mathcal{P}|) \cdot l^2 \end{aligned}$$

As we do not have access to the error for the entire dataset, $E_X(C)$, since its computation is expensive, in Algorithm 8 we propose a possible stopping criterion that bounds the displacement of the set of centroids. In the following result, we show a possible choice of this bound in a way that, if the proposed criterion is verified, then the common Lloyd's algorithm stopping criterion is also satisfied.

Theorem 12 *Given two sets of centroids $C = \{\boldsymbol{c}_k\}_{k=1}^K$ and $C' = \{\boldsymbol{c}'_k\}_{k=1}^K$, if $\|C - C'\|_\infty = \max\limits_{k=1,\dots,K} \|\boldsymbol{c}_k - \boldsymbol{c}'_k\| \leq \varepsilon_w$, where $\epsilon_w = \sqrt{l^2 + \frac{\epsilon^2}{n^2}} - l$, then $|E_X(C) - E_X(C')| \leq \varepsilon$.*

Proof. Initially, we bound the following terms: $\|\mathbf{x} - \mathbf{c_x}\| + \|\mathbf{x} - \mathbf{c}'_\mathbf{x}\|$ and $|\|\mathbf{x} - \mathbf{c_x}\| - \|\mathbf{x} - \mathbf{c}'_\mathbf{x}\||$ for any $\mathbf{x} \in X$.

If we set j and t as the indexes satisfying $\mathbf{c}_j = \mathbf{c_x}$ and $\mathbf{c}'_t = \mathbf{c}'_\mathbf{x}$, then we have $\|\mathbf{x} - \mathbf{c_x}\| + \|\mathbf{x} - \mathbf{c}'_\mathbf{x}\| = \|\mathbf{x} - \mathbf{c}_j\| + \|\mathbf{x} - \mathbf{c}'_t\| \leq \|\mathbf{x} - \mathbf{c}_t\| + \|\mathbf{x} - \mathbf{c}'_t\| \leq 2 \cdot \|\mathbf{x} - \mathbf{c}'_t\| + \varepsilon_w = 2 \cdot \|\mathbf{x} - \mathbf{c}'_\mathbf{x}\| + \varepsilon_w$ (1). Analogously, applying the triangular inequality, we have $|\|\mathbf{x} - \mathbf{c_x}\| - \|\mathbf{x} - \mathbf{c}'_\mathbf{x}\|| \leq \varepsilon_w$ (2). In the following chain of inequalities, we will make use of (1) and (2):

$$
\begin{aligned}
|E_X(C) - E_X(C')| &\leq |\sum_{\mathbf{x} \in X} \|\mathbf{x} - \mathbf{c_x}\|^2 - \|\mathbf{x} - \mathbf{c}'_\mathbf{x}\|^2| \\
&\leq \sum_{\mathbf{x} \in X} |\|\mathbf{x} - \mathbf{c_x}\|^2 - \|\mathbf{x} - \mathbf{c}'_\mathbf{x}\|^2| \\
&\leq \sum_{\mathbf{x} \in X} (\|\mathbf{x} - \mathbf{c_x}\| + \|\mathbf{x} - \mathbf{c}'_\mathbf{x}\|) \cdot \\
&\quad\ |\|\mathbf{x} - \mathbf{c_x}\| - \|\mathbf{x} - \mathbf{c}'_\mathbf{x}\|| \\
&\leq \sum_{\mathbf{x} \in X} \varepsilon_w \cdot (2 \cdot \|\mathbf{x} - \mathbf{c}'_\mathbf{x}\| + \varepsilon_w) \\
&\leq n \cdot \varepsilon_w^2 + 2 \cdot n \cdot \max_{\mathbf{x} \in X} \|\mathbf{x} - \mathbf{c}'_\mathbf{x}\| \cdot \varepsilon_w \\
&\leq n \cdot \varepsilon_w^2 + 2 \cdot n \cdot l \cdot \varepsilon_w = \varepsilon
\end{aligned}
$$

In Theorem 4, we show an interesting property of the BWKM algorithm. We verify that a fixed point of the weighted Lloyd's algorithm, over a partition with only well assigned blocks, is also a fixed point of Lloyd's algorithm over the entire dataset X.

Theorem 4 *If C is a fixed point of the weighted K-means algorithm for a spatial partition $\mathcal{B}$, for which all of its blocks are well assigned, then C is a fixed point of the K-means algorithm on X.*

Proof. $C = \{\mathbf{c}_1, \dots, \mathbf{c}_K\}$ is a fixed point of the weighted K-means algorithm, on a partition $\mathcal{P}$, if and only if when applying an additional iteration of the weighted K-means algorithm on $\mathcal{P}$, the generated clusterings

$\mathcal{G}_1(\mathcal{P}), \dots, \mathcal{G}_K(\mathcal{P})$, i.e., $\mathcal{G}_i(\mathcal{P}) \coloneqq \{P \in \mathcal{P} : \mathbf{c}_i = \underset{\mathbf{c} \in C}{\arg\min} \|\overline{P} - \mathbf{c}\|\}$, satisfies $\mathbf{c}_i = \frac{\sum_{P \in \mathcal{G}_i(\mathcal{P})} |P| \cdot \overline{P}}{\sum_{P \in \mathcal{G}_i(\mathcal{P})} |P|}$ for all $i = \{1, \dots, K\}$ (1).

Since all the blocks $B \in \mathcal{B}$ are well assigned, then the clusterings of C in X, $\mathcal{G}_i(X) \coloneqq \{\mathbf{x} \in X : \mathbf{c}_i = \underset{\mathbf{c} \in C}{\arg\min} \|\mathbf{x} - \mathbf{c}\|\}$, satisfy $|\mathcal{G}_i(X)| = \sum_{P \in \mathcal{G}_i(\mathcal{P})} |P|$ (2) and $\sum_{\mathbf{x} \in \mathcal{G}_i(X)} \mathbf{x} = \sum_{P \in \mathcal{G}_i(\mathcal{P})} \sum_{\mathbf{x} \in P} \mathbf{x}$ (3). From (1), (2) and (3), we have

$$\begin{aligned} \mathbf{c}_i &= \frac{\sum\limits_{P \in \mathcal{G}_i(\mathcal{P})} |P| \cdot \overline{P}}{\sum\limits_{P \in \mathcal{G}_i(\mathcal{P})} |P|} = \frac{\sum\limits_{P \in \mathcal{G}_i(\mathcal{P})} |P| \cdot \sum\limits_{\mathbf{x} \in P} \frac{\mathbf{x}}{|P|}}{\sum\limits_{P \in \mathcal{G}_i(\mathcal{P})} |P|} \\ &= \frac{\sum\limits_{P \in \mathcal{G}_i(\mathcal{P})} \sum\limits_{\mathbf{x} \in P} \mathbf{x}}{\sum\limits_{P \in \mathcal{G}_i(\mathcal{P})} |P|} = \frac{\sum\limits_{\mathbf{x} \in \mathcal{G}_i(X)} \mathbf{x}}{|\mathcal{G}_i(X)|} \forall\, i \in 1, \dots, K, \end{aligned}$$

this is, C is a fixed point of K-means algorithm on X.

.2.2 About the grid based RPKM

In the experimental section in Chapter 3, the partition sequence used (grid based RPKM) consisted on sequentially constructing a new spatial partition by dividing each block of the previous partition into 2^d new blocks, i.e., $\mathcal{P}$ can have up to $2^{i \cdot d}$ representatives. In this section, we provide some additional results in which we compare the performance of the grid based RPKM with respect to the methods and datasets presented in Section 4.2 and $K \in \{3, 5, 10, 25, 50\}$.

As in Chapter 3, we fix the maximum number of iterations, M, as the stopping criterion for the grid based RPKM. Initially, we considered $M = 10$, however just for the *CIF* and *3RN* datasets -case i)- the grid based RPKM managed to converge before reaching the limit running time (24 hours). Moreover, for the *HPC* and *WUY* datasets -case ii)-, we obtained results for $M = 5$ and, unfortunately, for the datasets with the largests dimensionality (*GS* and *SUSY*), the grid based RPKM failed to provide any output. The obtained results are summarized in the following figures:

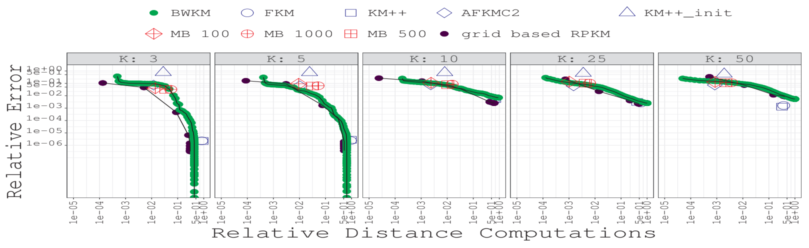

Fig. .1: Relative distance computations *vs* relative error on the *CIF* dataset.

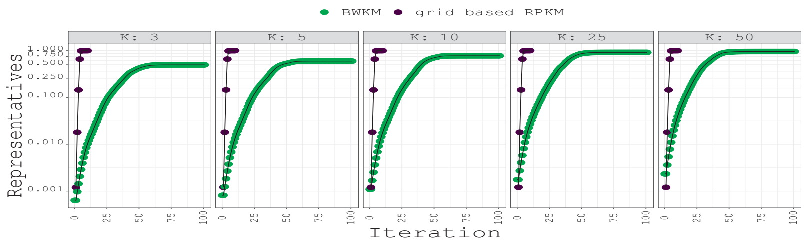

Fig. .2: Proportion representatives/instances with respect to the number of iterations on the *CIF* dataset.

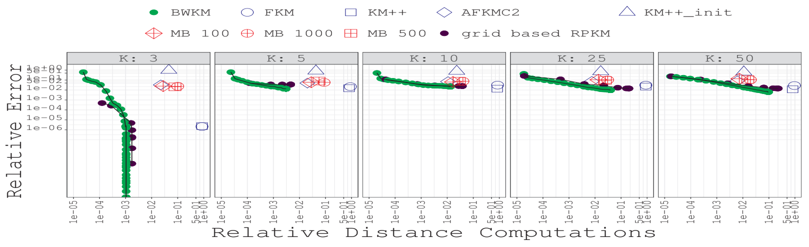

Fig. .3: Relative distance computations *vs* relative error on the *3RN* dataset.

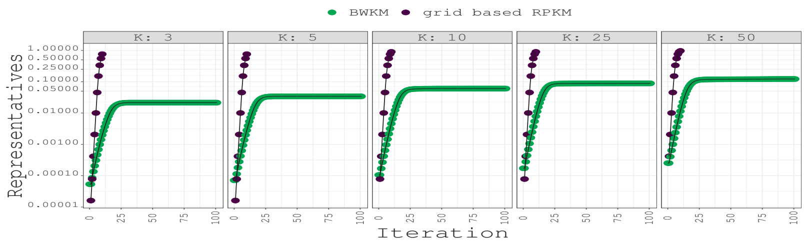

Fig. .4: Proportion representatives/instances with respect to the number of iterations on the *3RN* dataset.

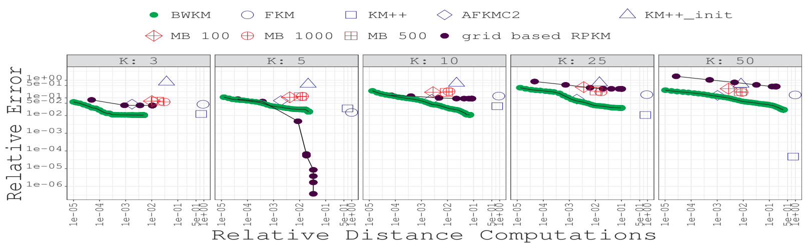

Fig. .5: Relative distance computations *vs* relative error on the *HPC* dataset.

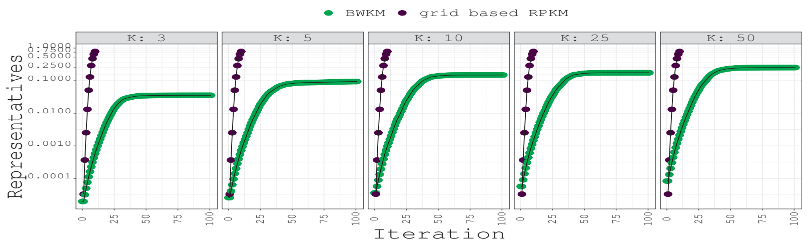

Fig. .6: Proportion representatives/instances with respect to the number of iterations on the *HPC* dataset.

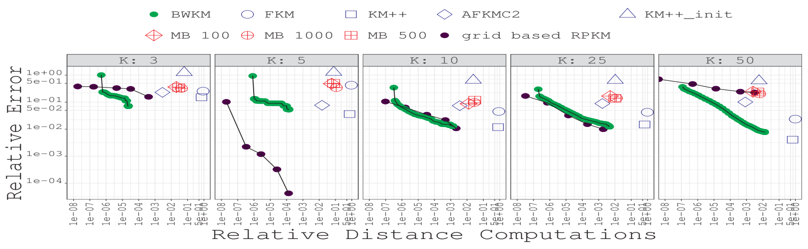

Fig. .7: Relative distance computations *vs* relative error on the *WUY* dataset.

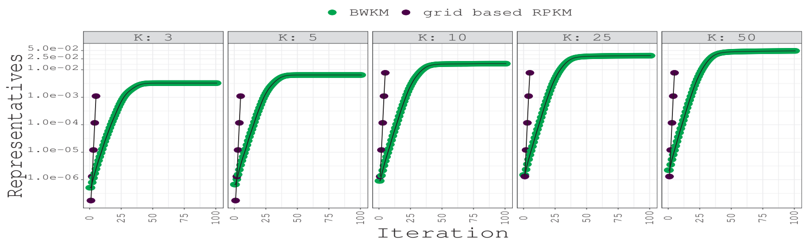

Fig. .8: Proportion representatives/instances with respect to the number of iterations on the *WUY* dataset.

Dataset	K	**BWKM**	**grid based RPKM**
CIF	3	0.498 ± 0.042	0.999 ± 0.000
	5	0.597 ± 0.021	0.999 ± 0.000
	10	0.779 ± 0.013	0.999 ± 0.000
	25	0.917 ± 0.005	0.999 ± 0.000
	50	0.958 ± 0.002	0.999 ± 0.000
3RN	3	0.021 ± 0.000	0.760 ± 0.000
	5	0.034 ± 0.001	0.760 ± 0.000
	10	0.060 ± 0.004	0.900 ± 0.000
	25	0.090 ± 0.008	0.900 ± 0.000
	50	0.123 ± 0.009	0.967 ± 0.000
HPC	3	0.038 ± 0.014	0.785 ± 0.000
	5	0.095 ± 0.014	0.785 ± 0.000
	10	0.148 ± 0.008	0.785 ± 0.000
	25	0.175 ± 0.009	0.785 ± 0.000
	50	0.253 ± 0.011	0.785 ± 0.000
WUY	3	0.003 ± 0.001	0.001 ± 0.000
	5	0.006 ± 0.000	0.001 ± 0.000
	10	0.017 ± 0.004	0.007 ± 0.000
	25	0.033 ± 0.004	0.007 ± 0.000
	50	0.049 ± 0.003	0.007 ± 0.000

Table .1: Proportion final number of representatives / instances for the different datasets and number of clusters.

In the datasets of case i), we have better view of the evolution of the number of representatives of the grid based RPKM with respect to the number of iterations. In Fig..2, Fig..4 and Table .1, we observe that the number of representatives of the grid based RPKM, after 10 iterations, is about the number of instances in both the *CIF* and *3RN* datasets, while, for the BWKM, we observe a much slower growth in the number of representatives. In particular, for the *3RN* dataset, the number of representatives, for the different number of clusters and after 100 iterations, is still under 13% the number of instances, while generating approximations of similar and/or better quality than those of the grid based RPKM. Furthermore, we observe that, for all the datasets, the number of representatives of BWKM reaches a *plateau* way before the final number of iterations, meaning that, for a small number of iterations, most of the blocks generated by BWKM are well assigned.

On the other hand, as the number of representatives for the grid based RPKM, for the datasets in case ii), is smaller than in the previous case, we observe in Fig..5 and Fig..7, that the quality of the approximation of the grid based RPKM is commonly much less competitive than the solutions obtained via BWKM: the grid based RPKM commonly has over 10% of relative error with respect to BWKM.

In Table .2, we present the proportion of cases in which BWKM generates a well assigned partition verified via the assignment function of Theorem 2. As we commented in Section 4.1.4.2, this is a sufficient condition to verify that the solution obtained via BWKM is also a fixed point of Lloyd's algorithm for the entire dataset. We observe that, specially for a low number of clusters, BWKM is very likely to converge to a local minima of the K-means problem. For instance, for *WUY* dataset and $K \in \{3, 5\}$, BWKM always generated well assigned partitions, which is quite remarkable as the number of representatives in these cases is under 0.6% of the number of instances. As expected, as the number of cluster increases, it is harder to verify such a condition, however we must remember that this is just a sufficient condition since we are using Theorem 2, rather than computing all the pairwise distances instance-centroid.

K	***CIF***	***3RN***	***HPC***	***WUY***
3	0.92	1.00	0.78	1.00
5	1.00	1.00	0.70	1.00
10	0.44	0.98	0.46	0.84
25	0.40	0.96	0.54	0.08
50	0.48	0.84	0.02	0.00

Table .2: Proportion of cases in which the spatial partition obtained by BWKM satisifes Theorem 4 verified via the misassigment function of Theorem 2.

From the results presented in this section, it is clear that BWKM alleviates the main drawback of the grid based RPKM, as it also controls the growth of the number of representatives, which, in the worst case scenario, only has a linearly growth in this case. This is an important factor, as it allows BWKM to scale better with respect to both the dimensionality and the number of iterations. Furthermore, BWKM is still a RPKM type approach, meaning that, besides the theoretical guarantees that we have developed during article and the results that just commented on, BWKM also has the guarantees of the grid based RPKM.

.3 Appendix of Chapter 5

In this section, we comment on the proofs to the theoretical results presented in Chapter 5 and an extension to the empirical results in Section 5.2.

.3.1 Proofs

As we commented, in Section 5.1, the proposed feature selection in Algorithm 9-13 is mainly based on the error bound proposed in Theorem 5. The result presented in Theorem 5 offers a simple way of quantifying the importance of a certain dimension, $j \in D_i$, in terms of its impact on the quality of the obtained clustering. This measurement consists on fixing the the corresponding entry, on each center of mass, to a given value, v_j, and estimating the increase of the error that it implies.

Theorem 7 *Given two clusters P_i and P_j and their corresponding centers of mass $\mathbf{c}_i$ and $\mathbf{c}_j$, then $f_{i,j} = \frac{|P_i|\cdot|P_j|}{|P_i|+|P_j|} \cdot \|\mathbf{c}_i - \mathbf{c}_j\|^2$.*

Proof. From now on, we refer by X_D to the dataset X on the dimensions in X and, by $X_{P_l} \subset X_D$, to the instances X_i belonging to the cluster P_l. We know that the following inequality holds

$$\sum_{\mathbf{x}\in X_{P_l}} \|\mathbf{x} - \mathbf{c}'_l\|^2 = \sum_{\mathbf{x}\in X_{P_l}} \|\mathbf{x} - \mathbf{c}_l\|^2 + |P_l| \cdot \|\mathbf{c}_l - \mathbf{c}'_l\|^2 \tag{.10}$$

and so, due to possible clustering re-assignments, from Eq..10, we deduce the following bound

$$\begin{aligned} E^{X_D}(C') &\le E^{X_D}(C) + \sum_{l=1}^{K} |P_l| \cdot \|\mathbf{c}_l - \mathbf{c}'_l\|^2 \\ &= E^{X_D}(C) + \sum_{l=1}^{K} |P_l| \cdot \sum_{j\in D\setminus S} (\mathbf{c}_{l,j} - v_j)^2 \\ &= E^{X_D}(C) + \sum_{j\in D\setminus S} t_j \end{aligned} \tag{.11}$$

In Remark 1 in Section 5.1.1, we described the following simple feature selection process, based on Theorem 5, that leads to a $1+\varepsilon$- approximation of $E^{X_{D_i}}(C_i)$:

Remark 1 *Using Theorem 5, we can minimize the size of the set of features selected, $S \subseteq D$, needed to keep a $1+\varepsilon$- approximation of C. This can be easily done by computing the set $\{t_j\}_{j=1}^{|D|}$ and sorting it increasingly, $\{t_{j:|D|}\}_{j=1}^{|D|}$. We then determine the largest index $k \in \{1,\ldots,|D|\}$ for which, $\sum_{l=1}^{k} t_{l:|D|} \le \varepsilon \cdot E^{X_D}(C_i)$ holds, i.e., $E^{X_D}(C') \le (1+\varepsilon)\cdot E^{X_D}(C)$. Therefore, the features selected, $S \subseteq D$, correspond to those dimensions associated to the last $|D|-k$ entries in $\{t_{j:|D|}\}_{j=1}^{|D|}$.* [2].

[2] In Section .3.3, we briefly present some additional practical results showing the accuracy of the proposed feature selection procedure.

Proof. The construction proposed in this remark can be directly verified from Theorem 5: Since $\{t_{j:|D|}\}_{j=1}^{|D|}$ is sorted incresingly and k is defined as the largest index for which, $\sum_{l=1}^{k} t_{l:|D|} \leq \varepsilon \cdot E^{X_S}(C)$ holds, i.e., $E^{X_D}(C') \leq (1+\varepsilon) \cdot E^{X_D}(C)$. Hence, the set of dimensions associated to the last $|D|-k$ entries in $\{t_{j:|D|}\}_{j=1}^{|D|}$ is the minimal cardinality set, generated via Theorem 5, that keeps a $1+\varepsilon$-approximation of $E^{X_D}(C)$.

In Theorem 6, we propose a bound to the clustering quality of the approximation obtained via Remark 1. Such a bound depends on the predefined $\varepsilon > 0$, the approximation ratio λ of algorithm $\mathcal{A}$ (which could be the K-means algorithm or any coreset-type approach, for instance) and the quality constant, $\varphi = \lambda^2 \cdot \frac{E_{K=1}^{X_S,opt}}{\sum_{i=1}^{t} E^{X_{S_i}}(C_i)}$, which measures the ratio between the 1-means optimal error and sum of the K-means error on the selected variables, for each party of dimensions.

Theorem 6 *Given a data set X, a constant $\varepsilon > 0$ and a partition of the dimensions $\{1,\ldots,d\}$ into t disjoint groups $\{D_1,\ldots,D_t\}$, if the set of features selected $S_i \subseteq D_i$ is obtained via Remark 1, for all $i \in \{1,\ldots,t\}$, then the output of a λ-approximate K-means algorithm (algorithm $\mathcal{A}$) on $S = \bigcup_{i=1}^{t} S_i$, C^*, satisfies*

$$E^X(C^*) \leq \varphi \cdot (1+\varepsilon) \cdot E^X(C_{opt}) \tag{5.3}$$

where $C_{opt} = \underset{C\subseteq\mathbb{R}^d,|C|=K}{\arg\min} E^X(C)$, $\varphi = \lambda^2 \cdot \frac{E_{K=1}^{X_S,opt}}{\sum_{i=1}^{t} E^{X_{S_i}}(C_i)}$ and, $E_{K=1}^{X_S,opt}$, is the optimal value of the 1-means error on X_S.

Proof. We first apply algorithm $\mathcal{A}$ on each D_i, which generates a set of centroids, C_i, satisfying

$$\begin{aligned} E^{X_{D_i}}(C_i') &\leq (1+\varepsilon) \cdot E^{X_{D_i}}(C_i) \\ &\leq \lambda \cdot (1+\varepsilon) \cdot E^{X_{D_i}}(C_{opt}) \ \forall\, i \in \{1,\ldots,t\} \end{aligned} \tag{.12}$$

and so, if we add all the elements in Eq..12, we get

$$\sum_{i=1}^{t} E^{X_{D_i}}(C_i') \leq \lambda \cdot (1+\varepsilon) \cdot E^X(C_{opt}). \tag{.13}$$

Furthermore, observe that

$$\sum_{i=1}^{t} E^{X_{D_i}}(C_i') = \underbrace{\sum_{i=1}^{t} E^{X_{S_i}}(C_i')}_{\text{Non-fixed error}} + \underbrace{\sum_{i=1}^{t} E_{K=1}^{X_{D_i\setminus S_i},opt}}_{\text{Fixed error}} \tag{.14}$$

From now on, we focus on bounding the "Non-fixed error" (T). We first apply algorithm $\mathcal{A}$ on $S = \bigcup_{i=1}^{t} S_i$ and obtain a set of K centroids, C^*:

$$\begin{aligned} E^{X_S}(C^*) &\leq \lambda \cdot E^{X_S}(C'_i) \\ &= \lambda \cdot [E^{X_{S_i}}(C'_i) + E^{X_{S \setminus S_i}}(C'_i)] \quad \forall\, i \in \{1, \ldots, t\}, \end{aligned}$$

and so,

$$E^{X_S}(C^*) \leq \frac{\lambda}{t} \cdot [T + \sum_{i=1}^{t} E^{X_{S \setminus S_i}}(C'_i)]. \tag{.15}$$

If we denote $L = \sum_{i=1}^{t} E^{X_{S \setminus S_i}}(C'_i)$, we would like to determine an upper bound to c such that $L = c \cdot T$. This bound is of relevance since, considering Eq..13 and Eq..15, we know that

$$\begin{aligned} E^{X}(C^*) \leq{}& \frac{\lambda^2}{t} \cdot (1+\varepsilon) \cdot (1+c) \cdot E^{X}(C_{opt}) \\ &- \frac{\lambda \cdot (1+c) - t}{t} \cdot \sum_{i=1}^{t} E_{K=1}^{X_{D_i \setminus S_i}, opt}. \end{aligned} \tag{.16}$$

We focus now on the factor $c = \frac{\sum_{i=1}^{t} E^{X_{S \setminus S_i}}(C'_i)}{\sum_{i=1}^{t} E^{X_{S_i}}(C'_i)} = \sum_{i=1}^{t} \frac{\sum_{j \neq i} E^{X_{S_j}}(C'_i)}{E^{X_{S_i}}(C'_i)} \leq \frac{(t-1) \cdot E_{K=1}^{X_D, opt}}{\sum_{i=1}^{t} E^{X_{S_i}}(C'_i)}$.

First of all, observe that, as $E^{X_{S_i}}(C'_i) \leq E_{K=1}^{X_{S_i}, opt}$, then

$$c = \frac{(t-1) \cdot E_{K=1}^{X_S, opt}}{\sum_{i=1}^{t} E^{X_{S_i}}(C'_i)} \geq \frac{(t-1) \cdot E_{K=1}^{X_S, opt}}{\sum_{i=1}^{t} E_{K=1}^{X_{S_i}, opt}} = t - 1 \tag{.17}$$

In other words, the second factor in Eq..16, will always be non-negative. Furthermore,

$$\frac{(\sum_{i=1}^{t} E^{X_{S_i}}(C'_i)) + (t-1) \cdot E_{K=1}^{X_S, opt}}{\sum_{i=1}^{t} E^{X_{S_i}}(C'_i)} \leq \frac{t \cdot E_{K=1}^{X_S, opt}}{\sum_{i=1}^{t} E^{X_{S_i}}(C'_i)} \tag{.18}$$

Hence, using Eq..19 and Eq..20, we can bound Eq..16 using information that we know in advanced, as follows

$$\begin{aligned} E^X(C^*) &\leq \lambda^2 \cdot (1+\varepsilon) \cdot \frac{E^{X_S,opt}_{K=1}}{\sum_{i=1}^{t} E^{X_{S_i}}(C'_i)} \cdot E^X(C_{opt}) \\ &= \lambda^2 \cdot (1+\varepsilon) \cdot \frac{E^{X_S,opt}_{K=1}}{\sum_{i=1}^{t} E^{X_{S_i}}(C_i)} \cdot E^X(C_{opt}) \end{aligned}$$

The following result is a corollary of Theorem 6 and its used, in Algorithm 13, as as criterion for selecting the variables after `Local approximation step`.

Corollary 2 *Given a set of positive constants $\{\varepsilon_1, \ldots, \varepsilon_t\}$, then if the set of features selected $S_i \subseteq D_i$ is obtained via Remark 1, with $\varepsilon = \varepsilon_i$, for all $i \in \{1, \ldots, t\}$, then the output of a λ-approximate K-means algorithm (algorithm $\mathcal{A}$) on $S = \bigcup_{i=1}^{t} S_i$, C^*, satisfies*

$$E^X(C^*) \leq \varphi \cdot (1 + \max_{i \in \{1,\ldots,t\}} \varepsilon_i) \cdot E^X(C_{opt}) \tag{5.4}$$

Proof. From Remark 1, we know the following inequality chain holds, for all $i \in \{1, \ldots, t\}$,

$$E^{X_{D_i}}(C'_i) \leq (1+\varepsilon_i) \cdot E^{X_{D_i}}(C_i) \leq (1 + \max_{j \in \{1,\ldots,t\}} \varepsilon_j) \cdot E^{X_{D_i}}(C_i).$$

Hence, the result can be deduced from Theorem 6.

.3.2 Feature Selection of KMR

As we commented in Section 5.1.1, if we define the lists $\mathcal{E}_i = \{\sum_{l=1}^{|D_i|-d_i} \frac{t^i_{l:|D_i|}}{E^{X_{D_i}}(C_i)}\}_{d_i=0}^{|D_i|-1}$, for all $i \in \{1, \ldots, t\}$, we can solve Problem 1 by selecting the m largest entries in $\{\mathcal{E}_1, \ldots, \mathcal{E}_t\}$. However, as the lists $\mathcal{E}_i$, for all $i \in \{1, \ldots, t\}$, are previously sorted in Remark 1, Problem 1 can be easily solved using the following $\mathcal{O}(t)$ time heuristic:

Algorithm 13: Feature Selection

```
Input: $\mathcal{E}_i^j = \sum_{l=1}^{|D_i|-j} \frac{t^i_{l:|D_i|}}{E^{X_{D_i}}(C_i)}$, for all $j \in \{0,\dots,|D_i|-1\}$ and
$i \in \{1,\dots,t\}$.
Output: Set of $m$ features selected, $D \subseteq \{1,\dots,d\}$.

- Take a set of non-negative integers $\{d_1,\dots,d_t\}$, satisfying $\sum_{i=1}^{t} d_i = m$.
- Set $\mathcal{E}_i^{|D_i|} = 0$ and $\mathcal{E}_i^{-1} \to \infty$, for all $i \in \{1,\dots,t\}$.
while $\max_{i\in\{1,\dots,t\}} \mathcal{E}_i^{d_i} > \min_{i\in\{1,\dots,t\}} \mathcal{E}_i^{d_i-1}$ do
    - Set $d_j = d_j + 1$ , where $j = \arg\max_{i\in\{1,\dots,t\}} \mathcal{E}_i^{d_i}$.
    - Set $d_l = d_l - 1$, where $l = \arg\min_{i\in\{1,\dots,t\}} \mathcal{E}_i^{d_i-1}$.
end
- For all $i \in \{1,\dots,t\}$, set $S_i$ as the dimensions associated to the last $d_i$
  entries of $\mathcal{E}_i$.
Return $S = \bigcup_{i=1}^{t} S_i$.
```

The heuristic proposed in Algorithm 13 is fairly simple as it just needs to update the maximum of $\{\mathcal{E}_i^{d_i}\}_{i=1}^t$ and the minimum of $\{\mathcal{E}_i^{d_i-1}\}_{i=1}^t$, which implies, at most, a $\mathcal{O}(t)$ time cost. Since the arrays, $\{\mathcal{E}_i^{d_i}\}_{d_i=0}^{|D_i|}$, are all sorted for $i \in \{1,\dots,t\}$, then when the condition $\max_{i\in\{1,\dots,t\}} \mathcal{E}_i^{d_i} \le \min_{i\in\{1,\dots,t\}} \mathcal{E}_i^{d_i-1}$, is satisfied, $\max_{i\in\{1,\dots,t\}} \mathcal{E}_i^{d_i}$ is minimized, i.e., Problem 1 is solved.

.3.3 Some practical results using Remark 1

In this section, we provide some additional practical results using the feature selection strategy proposed in Remark 1. In particular, we consider all 16 data sets used in Section 5.2 and apply the feature selection approach discussed in Remark 1 to obtain a $1+\varepsilon$-approximation of the solution obtained via KM++, for $\varepsilon \in \{0.01, 0.05, 0.10, 0.50, 1.01\}$.

As we commented in Section 5.1.1, Remark 1 is primarily based on the variable importance score proposed in Theorem 5. Such a score is a bound to the error increase that would take place, if all the centers of mass obtained by algorithm $\mathcal{A}$ are fixed on a predefined subset of dimensions. This bound does not take into consideration possible clusters re-assignements which notoriously fastens the selection procedure, as there is no need to compute an additional cluster re-assignment step, which would be $\mathcal{O}(n{\cdot}K{\cdot}m)$, and, more importantly, the effect of each variable is additive, meaning that the corresponding error increase of each dimension is independent of the subset of fixed dimensions considered, i.e., we do not need to evaluate all the possible combinations of fixed dimensions separately.

In Fig. .9, we show the obtained epsilon (which must be $\leq \varepsilon$) obtained by Remark 1. Afterwards, we present the error (difference) between, the epsilon predicted by Remark 1, and the epsilon obtained, for the same features selected by Remark 1, computing all clustering re-assignements, see Fig. .10. Finally, in Fig. .11, we observe the number of features discarded (fixed) to reach the $1 + \varepsilon$-approximation.

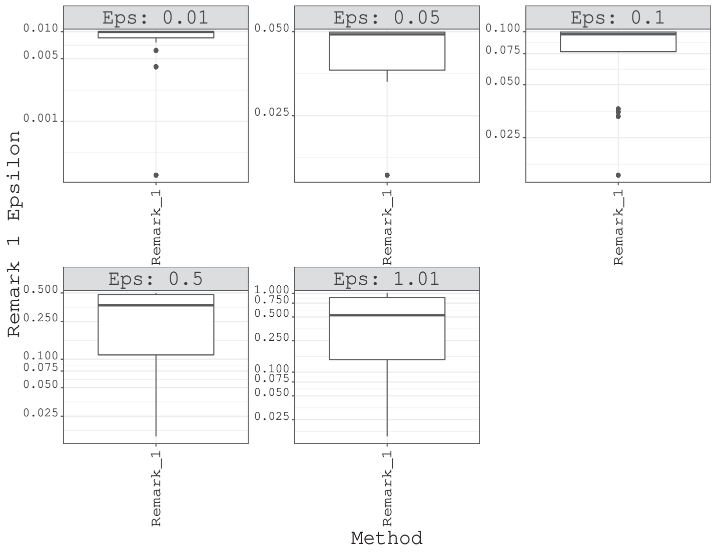

Fig. .9: Obtained epsilon after applying Remark 1.

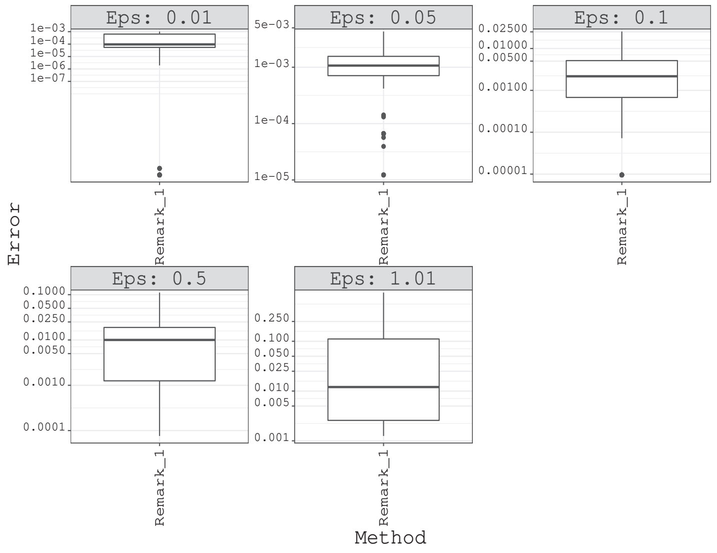

Fig. .10: Error obtained after selecting the last variable for which Remark 1 achieves the $1 + \varepsilon$-approximation.

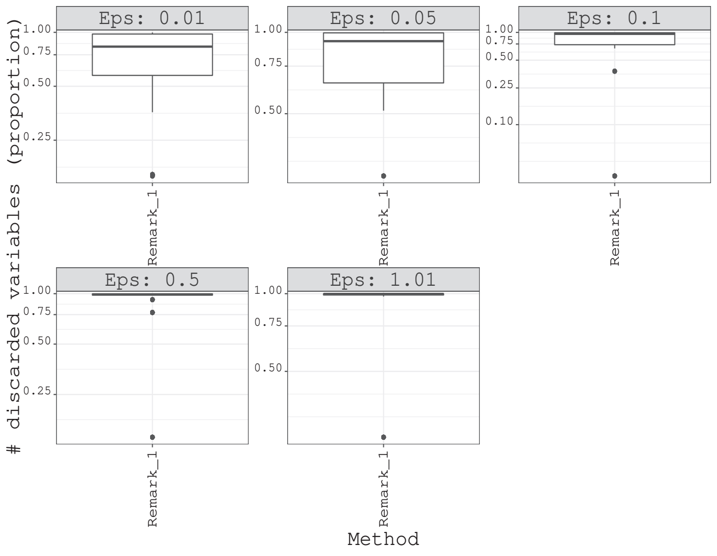

Fig. .11: Proportion of variables discarded by Remark 1 for reaching the $1+\varepsilon$-approximation.

Table .3: Average, over all data sets, for the results presented in Fig..9-.11.

ε	Remark 1 Epsilon	Error	Discarded Variables (proportion)
0.01	8.03×10^{-3}	1.76×10^{-4}	0.70
0.05	4.09×10^{-2}	8.85×10^{-4}	0.79
0.10	8.26×10^{-2}	3.18×10^{-3}	0.83
0.50	3.02×10^{-1}	1.53×10^{-2}	0.93
1.01	5.13×10^{-1}	5.33×10^{-2}	0.96

According to Fig..9, we observe that, as the value of ε decreases, the epsilon obtained by Remark 1 is closer to it. This observation is partially related to the fact that, as we increase ε, and, therefore, discard more variables, we will mostly remain with those variables with the highest error increments, reason for which adding an extra variable to those features that are already discarded, may easily exceed the $1+\varepsilon$- bound. In this scenario (largest values of ε), we observe that the error of the approximation provided by Theorem 5 loses more accuracy, as in this case we are fixing more variables and, therfore, adding more error to the bound. However, we must point out that the error obtained is still neglegible with respect to the ε chosen: In Fig..10 and Tab..3, we observe that, in large majority of the cases, such an error is under 5% of the actual value of ε.

Regardless of the value selected for ε, Remark 1 is always able to discard a large amount of the variables, while generating a $1+\varepsilon$-approximation of the best solution obtained via KM++. In particular, for $\varepsilon = 0.01$, Remark 1, eliminated, in average, 70% of the variables to reach the expected quality and, already, for $\varepsilon = 0.50$, over 90% of the variables are fixed and the error bound is satisfied.

.4 Appendix of Chapter 6

In this section, we comment on the proofs to the theoretical results presented in Chapter 6 and an extension to the empirical results in Section 6.2.

.4.1 Proofs

In this section, we present the proofs to Theorem 7 and Theorem 8 presented in Section 6.1.

Theorem 7 *Given two clusters P_i and P_j and their corresponding centers of mass $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$, then $f_{i,j} = \frac{|P_i|\cdot|P_j|}{|P_i|+|P_j|} \cdot \|\boldsymbol{c}_i - \boldsymbol{c}_j\|^2$.*

Proof. We know that for all $\mathbf{c} \in \mathbb{R}^d$,

$$\sum_{\mathbf{x}\in P_i} \|\mathbf{x}-\mathbf{c}\|^2 = \sum_{\mathbf{x}\in P_i} \|\mathbf{x}-\mathbf{c}_i\|^2 + |P_i|\cdot\|\mathbf{c}-\mathbf{c}_i\|^2 \tag{.19}$$

$$\sum_{\mathbf{x}\in P_j} \|\mathbf{x}-\mathbf{c}\|^2 = \sum_{\mathbf{x}\in P_j} \|\mathbf{x}-\mathbf{c}_j\|^2 + |P_j|\cdot\|\mathbf{c}-\mathbf{c}_j\|^2, \tag{.20}$$

In particular, for $\mathbf{c} = \frac{|P_i|\cdot\mathbf{c}_i+|P_j|\cdot\mathbf{c}_j}{|P_i|+|P_j|}$, we have $\|\mathbf{c}-\mathbf{c}_i\|^2 = \frac{|P_j|^2}{(|P_i|+|P_j|)^2}\cdot\|\mathbf{c}_i-\mathbf{c}_j\|^2$ and $\|\mathbf{c}-\mathbf{c}_j\|^2 = \frac{|P_i|^2}{(|P_i|+|P_j|)^2}\cdot\|\mathbf{c}_i-\mathbf{c}_j\|^2$, therefore adding both equations .19 and .20:

$$\sum_{\mathbf{x}\in P_{i,j}} \|\mathbf{x}-\frac{|P_i|\cdot\mathbf{c}_i+|P_j|\cdot\mathbf{c}_j}{|P_i|+|P_j|}\|^2 = \sum_{\mathbf{x}\in P_i}\|\mathbf{x}-\mathbf{c}_i\|^2 + \sum_{\mathbf{x}\in P_j}\|\mathbf{x}-\mathbf{c}_j\|^2 + \frac{|P_i|\cdot|P_j|}{|P_i|+|P_j|}\cdot\|\mathbf{c}_i-\mathbf{c}_j\|^2$$

Hence, $f_{i,j} = \frac{|P_i|\cdot|P_j|}{|P_i|+|P_j|}\cdot\|\mathbf{c}_i-\mathbf{c}_j\|^2$.

Theorem 8 *Given a Lloyd's algorithm fixed point, C, if the SMK-means algorithm splits a cluster P_s, taking the corresponding 2-means initialization via D^2-`sampling`[3] and $\min_{l\neq k\neq s} \frac{|P_l|\cdot|P_k|}{|P_l|+|P_k|}\cdot\|\mathbf{c}_l-\mathbf{c}_k\|^2 \leq \frac{E_{P_s}(\{\mathbf{c}_s\})}{|P_s|}$, then the re-initialization C' satisfies $E_X(C') \leq E_X(C)$, on average.*

Proof. In terms of `SplitStep`, it is not simple to estimate a priori g_s. However, we can find an upper bound to this term by analyzing the reduction of the error of P_s after the initialization of the 2-means on P_s.

In the statement, we propose to take the first centroid ($\mathbf{c}_s^1$) of the initialization as the center of mass of P_s, $\mathbf{c}_s$, and the second ($\mathbf{c}_s^2$) selected at random via D^2-sampling.

We now can take the following bound $E_{P_s}(\{\mathbf{c}_s,\mathbf{c}_s^2\}) \leq E_{P_s}(\{\mathbf{c}_s\}) - \|\mathbf{c}_s-\mathbf{c}_s^2\|^2$ (in the worst case, at least $\mathbf{c}_s^2$ is the closest instance to itself). Moreover,

[3] For the sake of simplicity, in Theorem 8, we consider a variation of the 2-means++ initialization that consists of taking the initial centroid as the center of mass of the cluster, $\mathbf{c}_s$, and, as the second centroid, $\mathbf{x}\in C_s$ with probability $\Pr(\mathbf{x}) \propto \|\mathbf{x}-\mathbf{c}_s\|^2$ (D^2-sampling).

$$\begin{aligned}
\mathbf{E}(E_{P_s}(\{\mathbf{c}_s,\mathbf{c}_s^2\})) &= \sum_{\mathbf{c}_s^2\in P_s} \frac{\|\mathbf{c}_s^2-\mathbf{c}_s\|^2}{\sum\limits_{\mathbf{x}\in P_s}\|\mathbf{x}-\mathbf{c}_s\|^2}\cdot E_{P_s}(\{\mathbf{c}_s,\mathbf{c}_s^2\})\\
&\le \sum_{\mathbf{c}_s^2\in P_s} \frac{\|\mathbf{c}_s^2-\mathbf{c}_s\|^2}{E_{P_s}(\{\mathbf{c}_s\})}\cdot (E_{P_s}(\{\mathbf{c}_s\})-\|\mathbf{c}_s-\mathbf{c}_s^2\|^2)\\
&= E_{P_s}(\{\mathbf{c}_s\}) - \frac{1}{E_{P_s}(\{\mathbf{c}_s\})}\cdot\sum_{\mathbf{c}_s^2\in P_s}\|\mathbf{c}_s^2-\mathbf{c}_s\|^4 \qquad (.21)
\end{aligned}$$

This is $g_s = \frac{1}{E_{P_s}(\{\mathbf{c}_s\})}\cdot\sum\limits_{\mathbf{c}_s^2\in P_s}\|\mathbf{c}_s^2-\mathbf{c}_s\|^4$, using the power-mean inequality [4], we have

$$\begin{aligned}
g_s &= \frac{1}{E_{P_s}(\{\mathbf{c}_s\})}\cdot\sum_{\mathbf{c}_s^2\in P_s}\|\mathbf{c}_s^2-\mathbf{c}_s\|^4\\
&\ge \frac{1}{E_{P_s}(\{\mathbf{c}_s\})}\cdot\frac{1}{|P_s|}\cdot(\sum_{\mathbf{c}_s^2\in P_s}\|\mathbf{c}_s^2-\mathbf{c}_s\|^2)^2 = \frac{E_{P_s}(\{\mathbf{c}_s\})}{|P_s|}
\end{aligned}$$

Furthermore, for `MergeStep`, we can establish the following upper bound

$$\min_{l\neq k}\frac{|P_l'|\cdot|P_k'|}{|P_l'|+|P_k'|}\cdot\|\mathbf{c}_l'-\mathbf{c}_k'\|^2 \le \min_{l\neq k\neq s}\frac{|P_l|\cdot|P_k|}{|P_l|+|P_k|}\cdot\|\mathbf{c}_l-\mathbf{c}_k\|^2$$

Then,

$$\begin{aligned}
\mathbf{E}(E_X(C')) \le E_X(C) &+ \min_{l\neq k\neq s}\frac{|P_l|\cdot|P_k|}{|P_l|+|P_k|}\cdot\|\mathbf{c}_l-\mathbf{c}_k\|^2\\
&- \frac{E_{P_s}(\{\mathbf{c}_s\})}{|P_s|},
\end{aligned}$$

and so, if $\min\limits_{l\neq k\neq s}\frac{|P_l|\cdot|P_k|}{|P_l|+|P_k|}\cdot\|\mathbf{c}_l-\mathbf{c}_k\|^2 \le \frac{E_{P_s}(\{\mathbf{c}_s\})}{|P_s|}$, then $\mathbf{E}(E_X(C')) \le E_X(C)$.

.4.2 Experiments

In the experimental section, we analyzed the effect of the number of clusters over the performance of SMKM++, SMKMr, HKM++, HKMr, KM++ and

[4] Given any real numbers $a_1,\dots,a_n$, then $\sum\limits_{i=1}^{n} a_i^2 \ge \frac{1}{n}\cdot(\sum\limits_{i=1}^{n} a_i)^2$.

FKM on a wide variety of data sets. In this section, we show that SMKM has a competitive performance regardless of the size of the data sets and their dimensionalities. In Fig..12-.19, each point represents the average relative number of distances computed and error obtained at each iteration of all the considered methods.

As we can see .12-.19, SMKMr and SMKM++ has a similar performance regardless of the data sets considered. Both variants of SMKM systematically converged at a very fast rate to approximations of similar or better quality than those generated by HKM++, HKMr, KM++ and FKM. At this point, we would like to highlight that, as for both SMKM and HKM, the final number of re-starts may vary at different executions, the curves associated to the relative average error of both SMKM++, SMKMr, HKM++ and HKMr, in Fig..12-.19, are not necessarily monotonically descendant, as it is for each run of the experiment:

As we discussed in the experimental section, in order to better appreciate the benefits of the proposed Split-Merge process on improving the quality of the approximation, we could take a look at the performance of those algorithms that are initialized at random- SMKMr (green circles), HKMr (pink diamonds) and FKM (black squares)-. We highlight this comparison, since the remaining algorithms are initialized via a K-means++ execution, and so the obtained error in their first iterations are usually closer to the lowest error found (there is less room for improvement). In this case, we observe for instance that SMKMr only computed, on average, 9.1% (BC, $K = 10$) and 12.2% (ET, $K = 250$) of the distances of FKM to match the quality of its approximation. Analogously, SMKMr computed, on average, 3.0% (AC, $K = 250$) and 6.5% (SVA, $K = 250$) of the distances of HKMr.

In the case of the multi-start Lloyd approaches, FKM and KM++, we observe, as commented in the experimental section, that their best performance takes places when the number of clusters is small ($K = 10$): As we increase the number of clusters, it is more likely for the SMK-means algorithm and the Hartigan-based Lloyd's algorithm to reduce the error of a given local minima.

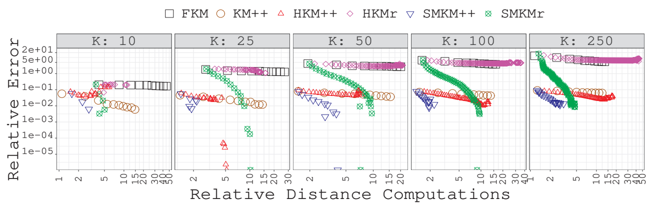

Fig. .12: Relative distance computations/error on *BC*.

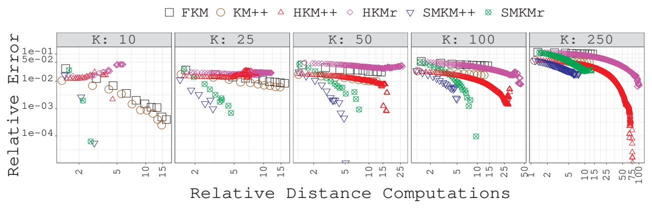

Fig. .13: Relative distance computations/error on *DIG*.

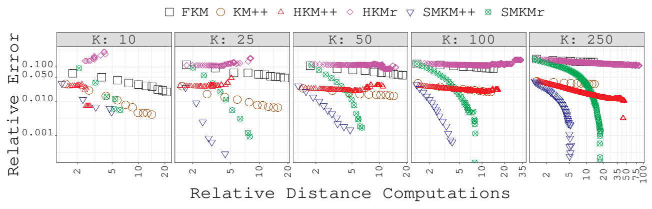

Fig. .14: Relative distance computations/error on *AC*.

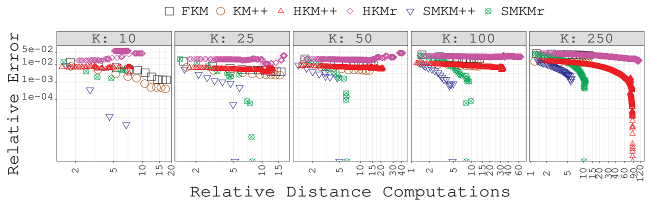

Fig. .15: Relative distance computations/error on *HAR*.

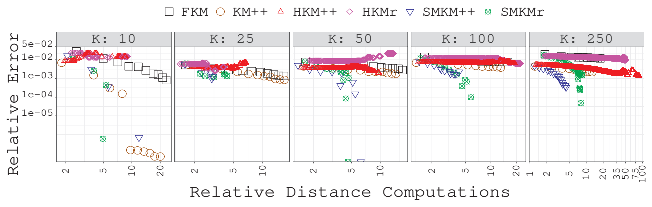

Fig. .16: Relative distance computations/error on *SVA*.

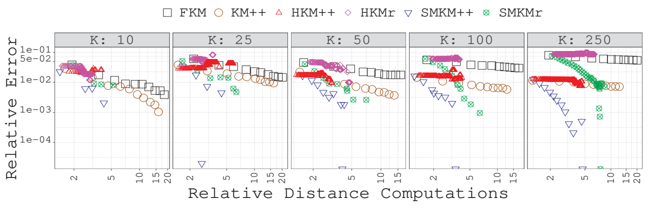

Fig. .17: Relative distance computations/error on *3RN*.

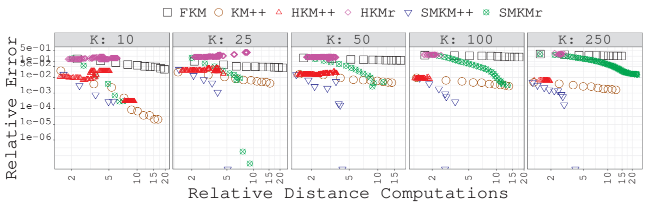

Fig. .18: Relative distance computations/error on *HP*.

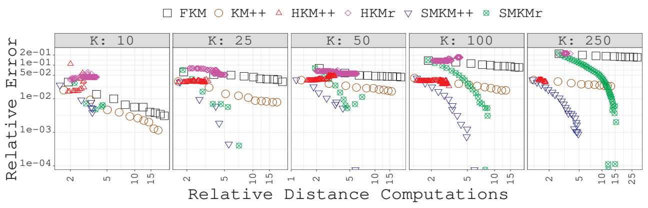

Fig. .19: Relative distance computations/error on *GS*.

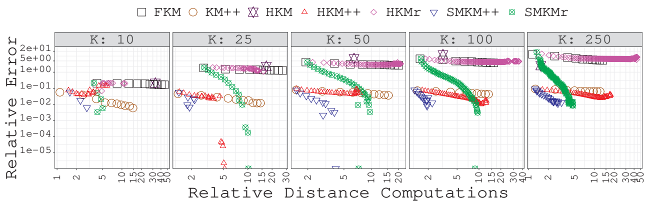

Fig. .20: Relative distance computations/error on *BC* (Hartigan-Wong *K*-means).

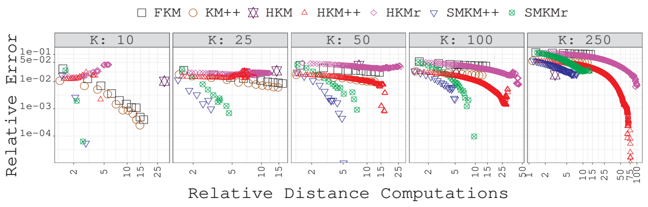

Fig. .21: Relative distance computations/error on *DIG* (Hartigan-Wong *K*-means).

.4.3 Hartigan-Wong K-means algorithm

Even when the goal of this article is the design of a re-initialization strategy for Lloyd's algorithm and its comparison to other multi-start approaches for the K-means algorithm, in this section we briefly report on the performance of the Hartigan-Wong K-means algorithm (HKM) for the same settings and data sets presented in the experimental section.

First of all, we would like to emphasize that HKMr and HKM++ refer to the multi-start K-means algorithms (initialized via Forgy's approach and K-means++, respectively) that are re-initialized through Hartigan's heuristic. On the other hand, HKM consists of applying Hartigan's heuristic recursively, without running Lloyd's algorithm, until the heuristic is not able to reduce the K-means error. Furthermore, we want to highlight that, due to the large computational time required by HKM, we had to set a time limit of 24 hours for each repetition of the experiment.

Table .4: Average relative error of HKM for the different data sets and number of clusters.

Data Set	K=10	K=25	K=50	K=100	K=250
BC	0.21	2.28	6.88	11.89	2.50
DIG	0.01	0.02	0.04	0.02	0.02
AC	0.40	0.27	0.32	0.53	0.54
HAR	0.06	0.12	0.16	0.28	0.18
SVA	0.01	0.68	1.46	2.05	2.97
3RN	3.00	7.40	15.21	26.88	61.90
HP	6.10	12.96	19.92	29.58	49.65
ET	3.83	8.10	14.40	26.66	66.17

Unfortunately, for such a time limit, HKM only managed to converge, for all the repetitions and number of clusters, in two data sets: *BC* and *DIG*, see Fig.20-.21. In both cases, we observe that HKM had a similar behavior to that of FKM in terms of quality, but computed a larger amount of distances. However, for $K = 250$, HKM showed a more competitive performance, leading to an average relative error of 0.02 in *DIG*.

As can be seen in Table .4, for the other data sets, HKM failed to generate competitive approximations, leading to average relative errors of order 10^2 in the *3RN*, *HP* and *ET*.

References

[1] M. Jordan, "Committee on the analysis of massive data, committee on applied and theoretical statistics, board on mathematical sciences and their applications, division on engineering and physical sciences, council, nr, 2013. frontiers in massive data analysis," *Frontiers in Massive Data Analysis*.

[2] R. Jacobson, "2.5 quintillion bytes of data created every day. How does CPG & Retail manage it?." `https://www.ibm.com/blogs/insights-on-business/consumer-products/2-5-quintillion-bytes-of-data-created-every-day-how-does-cpg-retail-manage-it/`, 2013.

[3] B. Marr, "How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read." `https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#60a0f4d60ba9`, 2018.

[4] A. Sinha and P. K. Jana, "A novel k-means based clustering algorithm for big data," in *Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on*, pp. 1875–1879, IEEE, 2016.

[5] X. Cai, F. Nie, and H. Huang, "Multi-view k-means clustering on big data.," in *IJCAI*, pp. 2598–2604, 2013.

[6] D. Feldman, M. Schmidt, and C. Sohler, "Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering," in *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1434–1453, Society for Industrial and Applied Mathematics, 2013.

[7] M. Chen, S. A. Ludwig, and K. Li, "Clustering in big data," in *Big Data Management and Processing*, pp. 333–346, Chapman and Hall/CRC, 2017.

[8] 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*, vol. 2, no. 3, pp. 267–279, 2014.

[9] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM computing surveys*, vol. 31, no. 3, pp. 264–323, 1999.

[10] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[11] P.-N. Tan *et al.*, *Introduction to data mining.* Pearson Education India, 2007.

[12] A. K. Jain and R. C. Dubes, *Algorithms for clustering data.* Prentice-Hall, Inc., 1988.

[13] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[14] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[15] P. Berkhin *et al.*, "A survey of clustering data mining techniques.," *Grouping Multidimensional Data*, vol. 25, p. 71, 2006.

[16] J.-P. W. Kappmeier, D. R. Schmidt, and M. Schmidt, "Solving k-means on high-dimensional big data," in *International Symposium on Experimental Algorithms*, pp. 259–270, Springer, 2015.

[17] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, *et al.*, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2008.

[18] R. T. Ng and J. Han, "Efficient and effective clustering methods for spatial data mining," in *Proceedings of VLDB*, pp. 144–155, 1994.

[19] R. T. Ng and J. Han, "Clarans: A method for clustering objects for spatial data mining," *IEEE transactions on knowledge and data engineering*, vol. 14, no. 5, pp. 1003–1016, 2002.

[20] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert systems with applications*, vol. 36, no. 2, pp. 3336–3341, 2009.

[21] W. Zhao, H. Ma, and Q. He, "Parallel k-means clustering based on mapreduce," in *IEEE International Conference on Cloud Computing*, pp. 674–679, 2009.

[22] O. Bachem, M. Lucic, and A. Krause, "Scalable and distributed clustering via lightweight coresets," *arXiv preprint arXiv:1702.08248*, 2017.

[23] M.-F. F. Balcan, S. Ehrlich, and Y. Liang, "Distributed k-means and k-median clustering on general topologies," in *Advances in Neural Information Processing Systems*, pp. 1995–2003, 2013.

[24] C. Ding and X. He, "K-means clustering via principal component analysis," in *Proceedings of the twenty-first international conference on Machine learning*, p. 29, ACM, 2004.

[25] J. Drake and G. Hamerly, “Accelerated k-means with adaptive distance bounds,” in *5th NIPS Workshop on Optimization for Machine Learning*, pp. 42–53, 2012.

[26] C. Elkan, “Using the triangle inequality to accelerate k-means,” in *Proceedings of the 20th International Conference on Machine Learning*, pp. 147–153, 2003.

[27] D. Feldman, M. Monemizadeh, and C. Sohler, “A ptas for k-means clustering based on weak coresets,” in *Proceedings of the twenty-third annual symposium on Computational geometry*, pp. 11–18, 2007.

[28] G. Hamerly, “Making k-means even faster,” in *Proceedings of the 2010 SIAM International Conference on Data Mining*, pp. 130–140, 2010.

[29] M. Lucic, O. Bachem, and A. Krause, “Strong coresets for hard and soft bregman clustering with applications to exponential family mixtures,” in *Artificial Intelligence and Statistics*, pp. 1–9, 2016.

[30] D. Sculley, “Web-scale k-means clustering,” in *Proceedings of the 19th International conference on World Wide Web*, pp. 1177–1178, 2010.

[31] H. Steinhaus, “Sur la division des corp materiels en parties,” *Bull. Acad. Polon. Sci*, vol. 1, no. 804, p. 801, 1956.

[32] J. Lücke and D. Forster, “k-means is a variational em approximation of gaussian mixture models,” *arXiv preprint arXiv:1704.04812*, 2017.

[33] R. Ostrovsky, Y. Rabani, L. J. Schulman, and C. Swamy, “The effectiveness of lloyd-type methods for the k-means problem,” in *Foundations of Computer Science, 2006. FOCS’06. 47th Annual IEEE Symposium on*, pp. 165–176, IEEE, 2006.

[34] S. Äyrämö and T. Kärkkäinen, “Introduction to partitioning-based clustering methods with a robust example,” *Reports of the Department of Mathematical Information Technology. Series C, Software engineering and computational intelligence 1/2006*, 2006.

[35] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, “Np-hardness of Euclidean sum-of-squares clustering,” *Machine Learning*, vol. 75, no. 2, pp. 245–248, 2009.

[36] M. Mahajan, P. Nimbhorkar, and K. Varadarajan, “The planar k-means problem is np-hard,” in *International Workshop on Algorithms and Computation*, pp. 274–285, 2009.

[37] 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 (methodological)*, pp. 1–38, 1977.

[38] Y. Linde, A. Buzo, and R. Gray, “An algorithm for vector quantizer design,” *IEEE Transactions on communications*, vol. 28, no. 1, pp. 84–95, 1980.

[39] J. MacQueen *et al.*, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.

[40] J. Max, "Quantizing for minimum distortion," *IRE Transactions on Information Theory*, vol. 6, no. 1, pp. 7–12, 1960.

[41] L. Bottou and Y. Bengio, "Convergence properties of the k-means algorithms," in *Advances in Neural Information Processing Systems*, pp. 585–592, 1995.

[42] C. D. Manning, P. Raghavan, and H. Schütze, "Evaluation in information retrieval," *Introduction to Information Retrieval*, pp. 151–175, 2008.

[43] O. Bachem, M. Lucic, H. Hassani, and A. Krause, "Fast and provably good seedings for k-means," in *Advances in Neural Information Processing Systems*, pp. 55–63, 2016.

[44] J. M. Peña, J. A. Lozano, and P. Larranaga, "An empirical comparison of four initialization methods for the k-means algorithm," *Pattern Recognition Letters*, vol. 20, no. 10, pp. 1027–1040, 1999.

[45] S. J. Redmond and C. Heneghan, "A method for initialising the k-means clustering algorithm using kd-trees," *Pattern Recognition Letters*, vol. 28, no. 8, pp. 965–973, 2007.

[46] D. Steinley and M. J. Brusco, "Initializing k-means batch clustering: A critical evaluation of several techniques," *Journal of Classification*, vol. 24, no. 1, pp. 99–121, 2007.

[47] A. Vattani, "K-means requires exponentially many iterations even in the plane," *Discrete & Computational Geometry*, vol. 45, no. 4, pp. 596–616, 2011.

[48] E. W. Forgy, "Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications," *Biometrics*, vol. 21, pp. 768–769, 1965.

[49] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the 18th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1027–1035, 2007.

[50] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *Proceedings of the VLDB Endowment*, vol. 5, no. 7, pp. 622–633, 2012.

[51] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[52] M. Telgarsky and A. Vattani, "Hartigans method: k-means clustering without voronoi," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 820–827, 2010.

[53] F. Nielsen and R. Nock, "Further heuristics for k-means: The merge-and-split heuristic and the (k, l)-means," *arXiv preprint arXiv:1406.6314*, 2014.

[54] N. Slonim, E. Aharoni, and K. Crammer, "Hartigan's k-means versus lloyd's k-means-is it time for a change?," in *IJCAI*, pp. 1677–1684, 2013.

[55] L. Bottou and Y. Bengio, "Convergence properties of the k-means algorithms," in *Advances in Neural Information Processing Systems*, pp. 585–592, 1995.

[56] P. S. Bradley and U. M. Fayyad, "Refining initial points for k-means clustering.," in *Proceedings of the 15th International Conference on Machine Learning*, vol. 98, pp. 91–99, 1998.

[57] I. Davidson and A. Satyanarayana, "Speeding up k-means clustering by bootstrap averaging," in *IEEE Data Mining Workshop on Clustering Large Data Sets*, 2003.

[58] J. Newling and F. Fleuret, "Nested mini-batch k-means," in *Advances in Neural Information Processing Systems*, pp. 1352–1360, 2016.

[59] S. Har-Peled and S. Mazumdar, "On coresets for k-means and k-median clustering," in *Proceedings of the 36th ACM Symposium on Theory of Computing*, pp. 291–300, 2004.

[60] A. Kumar, Y. Sabharwal, and S. Sen, "A simple linear time $(1 + \varepsilon)$-approximation algorithm for k-means clustering in any dimensions," in *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pp. 454–462, 2004.

[61] J. Matoušek, "On approximate geometric k-clustering," *Discrete & Computational Geometry*, vol. 24, no. 1, pp. 61–84, 2000.

[62] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "A local search approximation algorithm for k-means clustering," in *Proceedings of the 18th annual Symposium on Computational Geometry*, pp. 10–18, 2002.

[63] I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror, "Result analysis of the nips 2003 feature selection challenge," in *Advances in neural information processing systems*, pp. 545–552, 2005.

[64] C. Boutsidis, P. Drineas, and M. W. Mahoney, "Unsupervised feature selection for the k-means clustering problem," in *Advances in Neural Information Processing Systems*, pp. 153–161, 2009.

[65] D. Cai, C. Zhang, and X. He, "Unsupervised feature selection for multi-cluster data," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 333–342, ACM, 2010.

[66] X. He, D. Cai, and P. Niyogi, "Laplacian score for feature selection," in *Advances in neural information processing systems*, pp. 507–514, 2006.

[67] T. HO, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[68] G. Louppe and P. Geurts, "Ensembles on random patches," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 346–361, Springer, 2012.

[69] M. B. Cohen, S. Elder, C. Musco, C. Musco, and M. Persu, "Dimensionality reduction for k-means clustering and low rank approximation," in *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pp. 163–172, ACM, 2015.

[70] Y. Liang, M.-F. Balcan, and V. Kanchanapally, “Distributed pca and k-means clustering,” in *The Big Learning Workshop at NIPS*, vol. 2013, Citeseer, 2013.

[71] M. Holmes, A. Gray, and C. Isbell, “Fast svd for large-scale matrices,” in *Workshop on Efficient Machine Learning at NIPS*, vol. 58, pp. 249–252, 2007.

[72] E. Bingham and H. Mannila, “Random projection in dimensionality reduction: applications to image and text data,” in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 245–250, ACM, 2001.

[73] C. Boutsidis, A. Zouzias, and P. Drineas, “Random projections for k-means clustering,” in *Advances in Neural Information Processing Systems*, pp. 298–306, 2010.

[74] D. Arthur and S. Vassilvitskii, “How slow is the k-means method?,” in *Proceedings of the twenty-second annual symposium on Computational geometry*, pp. 144–153, ACM, 2006.

[75] R. A. Finkel and J. L. Bentley, “Quad trees a data structure for retrieval on composite keys,” *Acta informatica*, vol. 4, no. 1, pp. 1–9, 1974.

[76] V. Gandhi, J. Kang, and S. Shekhar, “Spatial databases,” tech. rep., MINNESOTA UNIV MINNEAPOLIS DEPT OF ELECTRICAL AND COMPUTER ENGINEERING, 2007.

[77] J. Fonollosa, S. Sheik, R. Huerta, and S. Marco, “Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring,” *Sensors and Actuators B: Chemical*, vol. 215, pp. 618–629, 2015.

[78] M. Capó, A. Pérez, and J. A. Lozano, “An efficient approximation to the k-means clustering for massive data,” *Knowledge-Based Systems*, vol. 117, pp. 56–69, 2017.

[79] M. Capó, A. Pérez, and J. A. Lozano, “An efficient k-means clustering algorithm for massive data,” *arXiv preprint arXiv:1801.02949*, 2018.

[80] H. Ding, Y. Liu, L. Huang, and J. Li, “K-means clustering with distributed dimensions,” in *International Conference on Machine Learning*, pp. 1339–1348, 2016.

[81] Z.-J. Bai, R. H. Chan, and F. T. Luk, “Principal component analysis for distributed data sets with updating,” in *International Workshop on Advanced Parallel Processing Technologies*, pp. 471–483, Springer, 2005.

[82] L. Hubert and P. Arabie, “Comparing partitions,” *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

[83] W. M. Rand, “Objective criteria for the evaluation of clustering methods,” *Journal of the American Statistical association*, vol. 66, no. 336, pp. 846–850, 1971.

[84] D. Pelleg, A. W. Moore, *et al.*, “X-means: Extending k-means with efficient estimation of the number of clusters.,” in *Icml*, vol. 1, pp. 727–734, 2000.

[85] G. H. Ball and D. J. Hall, “Isodata, a novel method of data analysis and pattern classification,” tech. rep., Stanford research inst Menlo Park CA, 1965.

[86] I. S. Dhillon, Y. Guan, and J. Kogan, “Iterative clustering of high dimensional text data augmented by local search,” in *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pp. 131–138, IEEE, 2002.

[87] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, *et al.*, “Mllib: Machine learning in apache spark,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[88] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, *et al.*, “Apache spark: a unified engine for big data processing,” *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[89] J. Fan, R. Samworth, and Y. Wu, “Ultrahigh dimensional feature selection: beyond the linear model,” *Journal of machine learning research*, vol. 10, no. Sep, pp. 2013–2038, 2009.

[90] S. Greenhill and S. Venkatesh, “Distributed query processing for mobile surveillance,” in *Proceedings of the 15th ACM international conference on Multimedia*, pp. 413–422, ACM, 2007.

[91] S. Mitra, M. Agrawal, A. Yadav, N. Carlsson, D. Eager, and A. Mahanti, “Characterizing web-based video sharing workloads,” *ACM Transactions on the Web (TWEB)*, vol. 5, no. 2, p. 8, 2011.

[92] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[93] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise.,” in *Kdd*, vol. 96, pp. 226–231, 1996.