Unsupervisd and Reinforcement Learning
Unsupervised Machine Learning Course Work:
Recursive Partition based K-means

García García, Alba María

Barcelona, Master in Artificial Intelligence

1^{st} June 2020

Contents

1 Introduction

The **purpose** of this course work is to **implement a novel Unsupervised Learning algorithm** proposed by a paper chosen by the student amongst the available at https://www.cs.upc.edu/~bejar/URL/courseworkB.html. This implementation must be carried out in *Python 3*, following the API conventions used by the library `scikit-learn` and must be as efficient as possible. The last is important because it is also required to tailor an experimentation which compares the implemented algorithm against well accepted methods using artificial and real datasets in order to test its expected properties. The paper chosen for this work is "An efficient K-means algorithm for Massive Data" by M. Capó, A. Pérez and J. A. Lozano [1]. It presents the ***recursive partition based K-means* algorithm**, an algorithm based on the original *K-means* method but introducing the concept of *partition* in order to reduce the number of distance computations with the aim of obtaining a more efficient implementation.

The **structure of this document** is formed by four main sections: "Motivation", "The *Recursive Partition based K-means* algorithm, "Experimentation" and "Conclusions". The first introduces the problems of current Unsupervised Learning algorithms when applied to *big data*, the motivation behind the development of this new algorithm; the second explains in detail the proposed algorithm and comments the improvements added to the original implementation for the sake of efficiency; the third shows the results obtained by the experimentation proposed in the paper along with an analysis of the results; and the last section mentions the conclusions drawn from the experimentation.

2 Motivation

In recent years, we have witnessed an **exponential increase of the amount of data available** thanks to the spread of the digital storage; reaching to what we could call the *Information Age*. These massive data are interesting for a wide variety of fields because they can provide *information* when analyzed. However, this analysis cannot be performed manually –it is impracticable– and has to be done automatically by means of Machine Learning methods. **One of the most relevant analysis performed to data is *clustering***, which consists in grouping different instances of the data in disjoint sets according to their *intrinsic* characteristics.

The most popular algorithm for this task is ***K-means*** [2], which is an iterative process that selects randomly K centroids from the instances of the dataset where is applied to (*Forgy's initialization* [3]) so that each instance belongs to the cluster defined by the closest centroid, minimizing the error function (Formula 1). An iterative stage known as *Lloyd's algorithm* follows this initialization stage, which, in each iteration, changes the position of every centroid to the center of mass of the instances that form the cluster it defines (update step) and then re-assigns the clusters to the instances as explained before (assignement step). This iterative process ends when the changes in the centroids in two consecutive iterations are so minimal that the decrease in the error function is below a given *threshold* or when a maximum number of iterations is reached.

$$E(C) = \sum_{x \in D} \min ||x - c_k||^2, \quad k = 1, ..., K \tag{1}$$

Notes on Formula 1.

- $D = \{x_1, ..., x_n\} \in \mathbb{R}^d$ define the set of instances of the dataset, embedded in a real space with dimensionality d.
- $C = \{c_1, ..., c_K\} \in \mathbb{R}^d$ define the set of K centroids, also embedded in a real space with dimensionality d.

Despite its high popularity, ***K-means* has well-known drawbacks** named the strong impact of the initialization in the final set of centroids and its not so low complexity –of O(nKd)– due to the number of distance computations d performed when assigning a cluster to every instance; being the latter of special relevance for massive datasets. The **initialization issue** can be addressed using several well-known alternatives: the simplest is to repeat the experiment a number of times and retain the one with the lowest error (not recommended when applied to massive datasets because the performance of the algorithm depends of the number of instances n) and the most effective to date, ***K-means++*** [4], which chooses the initial centroids based on a probability distribution proportional to the distance with respect to the previously selected centroids.

Several approaches have been proposed as **alternatives to the *Lloyd's algorithm*** with the purpose of reducing its time complexity without a high cost in cluster quality. The most widespread proposal of these characteristics is ***mini batch K-means*** [5], which uses only a small set of randomly chosen instances of fixed size at each iteration instead of the full dataset. The proposed algorithm by this paper, ***recursive partition based K-means*** –henceforth named *RPKM*–, can also be included in this category, as it also aims to reduce the number of distance computations generating competitive clusters but using the concept of *partition* rather than the concept of *batch*.

3 The *Recursive Partition based K-means* algorithm

The paper "An efficient K-means algorithm for Massive Data" presents an **alternative to the classic *clustering* algorithm *K-means*: *RPKM***. This section details the method itself according to the documentation provided by the paper in "Explanation of the algorithm" and then comments how it has been implemented for this course work, focusing on the design decisions intended for a better optimization of the source code in "Implementation approach".

3.1 Explanation of the algorithm

The idea behind *RPKM* is to **solve the *K* partition problem**, this is, to divide the space defined by the given dataset into K regions or *clusters*, **dividing recursively this space in a *thinner* partition** and obtaining for each partition a new set of *clusters* with their corresponding centroids applying a weighted version of *Lloyd's algorithm* to the *representatives* of the partition rather than to every instance of the dataset and thus reducing the number of distance computations. This partitioning process is repeated iteratively until some stopping criterion is met.

A ***partition P*** of the dataset D (Formula 2) is defined as a **nonempty set of subsets of instances** of the dataset D so that the disjoint union of these subsets form the entire dataset. Each subset $S \in P$ is characterized by two elements: its ***weight*** and its ***representative***. The *weight* is computed as the cardinality of the subset $|S|$, this is, the number of instances it contains, and the *representative* is equal to the center of mass of its elements, computed as the mean of the instances in the subset[1].

$$P = \{S_1, ..., S_t\}, \quad P \neq \varnothing, \quad \bigsqcup_{i=1}^{t} S_i = D \tag{2}$$

As we explained before, *RPKM* generates a sequence of *partitions* $P_1, ..., P_m$ of the given dataset, one per iteration, so that every new partition is *thinner* than the previous. Given two partitions of the dataset, P and P', we say that ***P' is thinner than P if every subset S ∈ P can be written as the union of subsets of P'***. Consequently, at every iteration we have more *representatives*, increasing the computational cost of every iterative step due to the greater number of distance computations required. However, the algorithm is expected to converge at a low m.

[1] Formula $\overline{P} = \frac{\sum_{x \in P} x}{|P|}$ for this specific notation.

The inclusion of **the partitioning strategy adds a computational overhead** to *RPKM* when compared to the original *K-means*, which does not need this *preprocessing* of the dataset. The authors estimate the complexity of the computation of the complete partition sequence to be of $O(d(n + \sum_{i=2}^{m} |P_i|))$ provided that the set of *representatives* and *weights* of each partition are computed *backwards*, this is, from P_m to P_1. Therefore, this forces the method to obtain all the partitions at the beginning altough some of them may be unsued due to a fast convergence.

The **data structure** to store these partitions proposed by the paper is the ***d-dimensional generalization of a quadtree*** (Figure 1). A *quadtree* is a tree that divides 2D spaces recursively into four equal-sized regions, containing the root node all the area and the rest of the nodes ¼ of its parent's area. Thus, a d-dimensional generalization of this tree can be understood in this context as a tree whose root node contains the entire dataset D and the rest of the nodes $1/2^d$ of its parent's space, thus understanding a partition as the union of the nodes in the same level. We can observe that in both cases the *thinner* constraint is preserved.

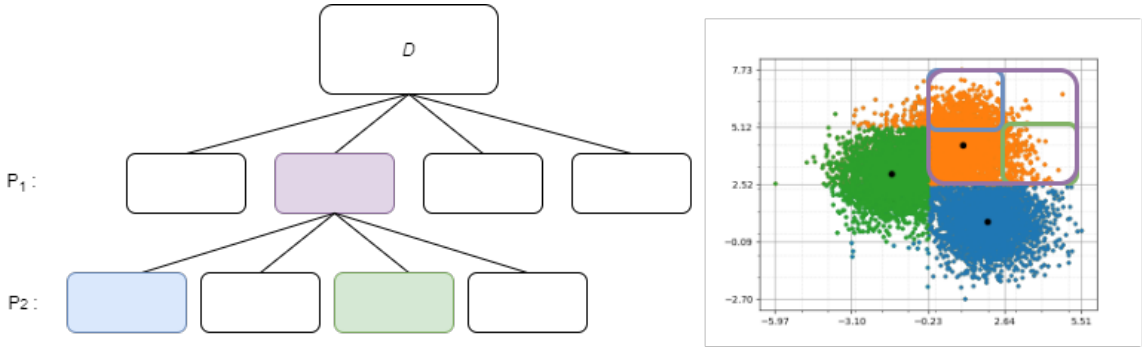


Figure 1: Example of a regular *quadtree* when applied to the partitioning process defined by RPKM for a two dimensional dataset.

As explained at the beginning of this section, *RPKM* obtains for each partition P a new set of clusters G with their corresponding centroids C applying the ***Weighted Lloyd's algorithm*** (Algorithm 1) to the *representatives* of the partition P. This algorithm is an adaptation of the classic *Lloyd's algorithm* used by *K-means* so that it can be applied to weighted points, the *representatives*.

The ***Lloyd's algorithm*** is a process which performs **two operations** iteratively: the **assignment of data points to clusters** and the **update of the position of the cluster's centroids**. The criterion used to assign a certain data point x to a cluster G_k is the distance of x to the cluster's centroid c_k according to the L2 norm so that this distance is minimized (Formula 3) and therefore the error function (Formula 1) is minimized too. This assignment step appears twice in the algorithm: one for obtaining the initial set of clusters G_0 given the initial set of centroids C_0 and another for getting the remaining set of clusters G_r obtained in each iteration. On the other hand, the update of the centroids has similarities with the computation of the *representatives* seen before, as they are also defined as the center of mass of the data points within their corresponding cluster.

$$\text{argmin} \, ||x - c_k||^2, \quad k = 1, ..., K \tag{3}$$

Hence, which are the **changes applied to the *Lloyd's algorithm* so that it can work with weighted data points**? The assignment step remains unchanged if we consider the *representatives* as regular data points. However, the update step changes so that the center of mass also takes into account the weight of each *representative* $\overline{S}$ (Formula 4).

$$c_k = \overline{G_k} = \frac{\sum_{\overline{S} \in G_k} |\overline{S}| \cdot \overline{S}}{|G_k|}, \quad k = 1, ..., K \tag{4}$$

Algorithm 1: *Weighted Lloyd*

Data: Set of *representatives* $\{\overline{S}\}_{S\in P}$ and *weights* $\{|S|\}_{S\in P}$ for the partition P. Number of clusters K and initial set of centroids C_0.
Result: Set of centroids C_r and set of clusters G_r.
Step 0: Initial assignment.
$G_0 \longleftarrow C_0$; r = 0;
while *stopping criterion is not met* **do**
  r = r + 1;
  Step 1: Update step.
  $C_r \longleftarrow G_{r-1}$;
  Step 2: Assignment step.
  $G_r \longleftarrow C_r$;
end
Return C_r and G_r.

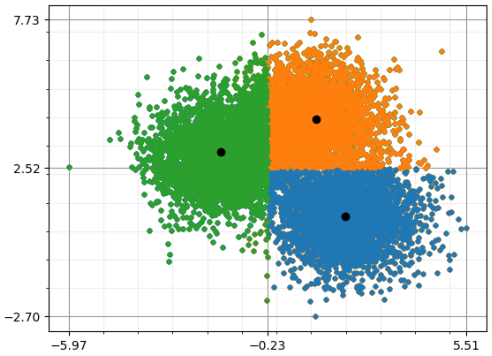
The **time complexity** of this weighted version of the *Lloyd's algorithm* is significantly lower than the original supposing that $|P| \ll n$, since both perform the same operations but over different data points: *representatives* instead of instances. Thus, we have that the complexity of *Weighted Lloyd* changes its definition from $O(nKd)$ to $O(|P|Kd)$.

We can find the **formalization of RPKM** in Algorithm 2. Thereby, *RPKM* is given as input a dataset D, the number of *clusters* K to group that data and the number of maximum iterations m; and returns the set of centroids C_r, from which we can induce the set of *clusters*. As explained before, the first step consists on computing the full sequence of partitions *backwards*. Then, at each iteration, *Weighted Lloyd* is called to compute the set of centroids C_i for the *representatives* of the i^{th} partition in the sequence, P_i, taking as reference the centroids got in the previous iteration, C_{i-1}. The proposed centroid initialization and stopping criterion by the authors is the same as *K-means*, *Forgy's* (random choice of the *representatives*) and maximum allowed centroid's displacement in consecutive iterations given a *threshold* or maximum number of iterations m, respectively. The time complexity of *RPKM*, $O(\max(d(n+\sum_{i=2}^{m}|P_i|), |P_m|Kd))$, depends on which of the two steps shown in Algorithm 2 is more costly (either the partitioning of the dataset or the iterative computation of the centroids). However, a fast analysis seem to indicate that a high value of K is enough for the iterative process to be more costly than the computation of the partitions.

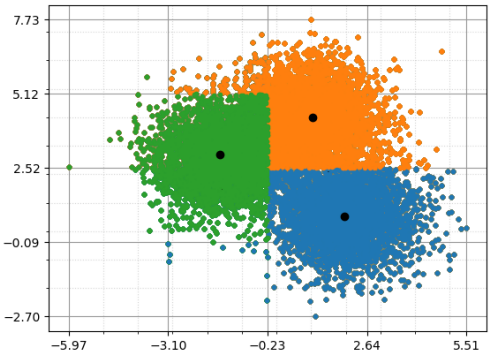
Algorithm 2: *RPKM*

Data: Dataset D, number of clusters K, maximum number of iterations m.
Result: Set of centroids approximation C_i.
Step 1: Compute the set of *weights* and *representatives* of the sequence of thinner partitions, $P_1, ..., P_m$, backwards; i = 0;
while *stopping criterion is not met* **do**
  i = i + 1;
  Step 2: Update the centroid's set approximation, $C_i = \{c_j^i\}_{j=1}^K$:
  $C_i = WeightedLloyd(\{\overline{S}\}_{S\in P}, \{|S|\}_{S\in P}, K, C_{i-1})$;
end
Return C_i.

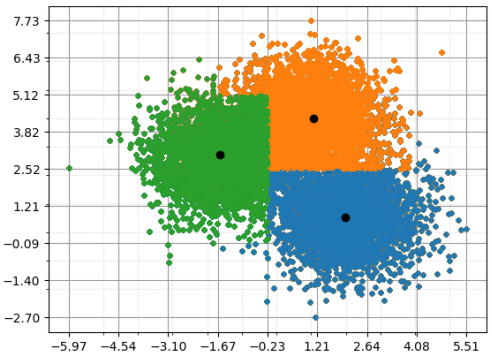
Therefore, ***RPKM* is** nothing more than **a *K-means* performed on different stages, where the information about the distribution of the data is more precise each next stage thus allowing a better centroid placement**. Conceptually, it is the same idea applied in stochastic gradient descent optimization methods with adaptative learning rate, where the first steps find the broader area of the function where the error is minimized and later on look for a local minimum more exhaustively in the area of interest (see the evolution of the error function in Figure 2).



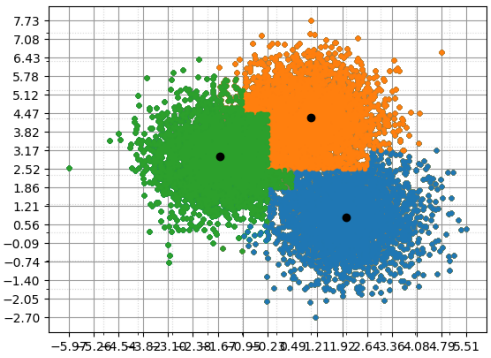
(a) i=1, $E(C_1) = 12,012.23$



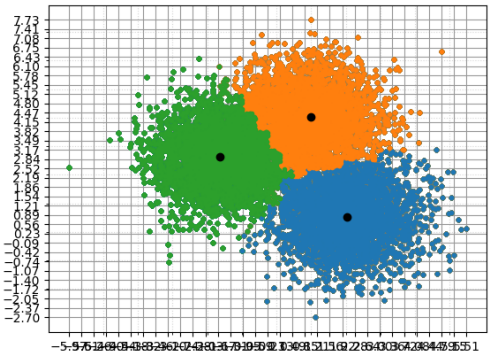
(b) i=2, $E(C_2) = 11,989.16$



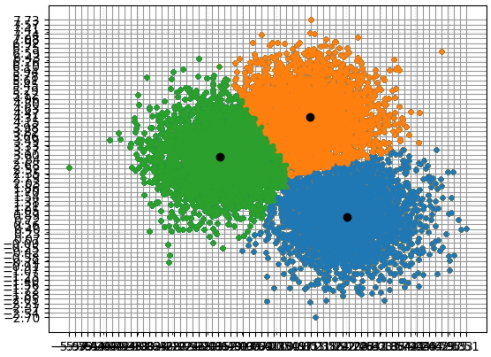
(c) i=3, $E(C_3) = 11,958.81$



(d) i=4, $E(C_4) = 11,871.43$



(e) i=5, $E(C_5) = 11,824.23$



(f) i=6, $E(C_6) = 11,816.74$

Figure 2: *Clustering* obtained after each iteration i of the *RPKM* algorithm for an artificial dataset of n=10,000 generated from a mixture of three Gaussian distributions making the area occupied by each subset visible. Reference error (*K-means++*) E = 11,813.51.

3.2 Implementation approach

The **source code for *RPKM*** can be found in the file **`RPKM.py`**. This file contains the object *RPKM*, which includes all the main functions provided by the algorithms of the library `scikit-learn`: `fit()`, `predict()` and `fit_predict()`. Apart from the **parameters** of the original *RPKM* algorithm, some **others were added**: a maximum number of iterations for *Weighted Lloyd* (Algorithm 1), *max_ iter*, so that it can stop when the maximum centroid displacement for two consecutive iterations is under the threshold *tol* (which is also another parameter) or when this number of iterations is reached, following the criterion proposed by the authors for Algorithm 2; an optional seed for computing the initial set of centroids C_0, *random_ state*, which is always computed as proposed in *Forgy's initialization* as recommended in the paper; and a verbosity mode.

The authors proposed a recursive data structure, the *quadtree*, to store all the partitions that could be needed during the execution of the algorithm –a total of m–. The problem of this data structure apart from the well-known poorly efficient memory usage of any recursive method, is that its size grows exponentially in terms of m, its *depth*, since each node has 2^d children in order to assure that all the possible divisions of the space are taken into account even though many of them may be empty. Furthermore, some of the partitions of the deeper levels may never be used because *RPKM* converged earlier than reaching the maximum number of iterations. Thereby, in order **to overcome a probable memory overflow** issue, **a non-recursive approach was implemented**: at each iteration i, the partition P_i is computed as a **one-dimensional array of size n** which contains at each position the coordinates (in form of d-dimensional array) of the partition subset it belongs to. Later on, these d-dimensional arrays are transformed to unique indexes for an easier handling of the data structure. The main advantages of this design is that it ignores all the empty subsets of the dataset, which are never employed in *Weighted Lloyd*, and that it computes only the partitions that are actually used. Moreover, the obtaining of the *representatives* and *weights* of each partition subset has also been optimized splitting those partitions with only one data point from those with more than one, so that the *representative* and *weight* of the first can be obtained straightforward without any computation.

An issue never addressed by the authors is **what happens when the number of clusters K is greater than the number of *representatives***. In order to solve this issue a function called `compute_min_partition()` was developed. This function computes the minimum theoretical partition index allowed for a given dataset D and a number of clusters K. However, empty subsets can appear and this minimum theoretical partition index may not be enough, so it is mandatory to iteratively partition the dataset until the number of *representatives* is actually greater or equal than the number of clusters K. The idea is that m partitions are still evaluated at most but with a shift in the index when needed. The resulting algorithm with these changes is shown in Algorithm 3.

Algorithm 3: Implemented *RPKM*

Data: Dataset D, number of clusters K, maximum number of iterations m.
Result: Set of centroids approximation C_i.
Step 1: Compute the minimum theoretical partition index i for the dataset D and the number of clusters K. Then, partition D until *representatives* $\geq K$ if needed;
i = minimum partition index;
while *stopping criterion is not met* **do**
  i = i + 1;
  Step 2: Compute the set of *weights* and *representatives* of the partition P_i;
  Step 3: Update the centroid's set approximation, $C_i = \{c_j^i\}_{j=1}^K$:
  $C_i = WeightedLloyd(\{\overline{S}\}_{S\in P}, \{|S|\}_{S\in P}, K, C_{i-1})$;
end
Return C_i.

4 Experimentation

The **experimentation** required for this course work is to compare experimentally the implemented algorithm to similar well-accepted methods using artificial and real datasets. Since the experimentation followed in the paper has all the aforementioned characteristics, it **has been completely reproduced** with the only exceptions that the number of repetitions of each experiment has been reduced to 5 and that the number of instances tested in the real dataset has been the same as in the artificial in order to make both experimentations comparable. In all the experiments performed, *RPKM* have been compared against itself or against the `scikit-learn` implementations of *K-means++* and *mini batch K-means*; setting the same number of maximum iterations with *early stopping* for all of them so that the comparison is fair. The only parameters tested have been the size and dimension of the dataset, n and d, respectively; and the number of clusters K. Regarding the algorithm-specific parameters, *RPKM*'s maximum number of iterations m have been always set to 1, 2, 3, 4, 5 and 6 ignoring the *convergence stopping criterion* so that all the iterations specified are actually computed and the batch sizes of *mini batch K-means* have been 100, 500 and 1000.

The following subsections, "Artificial datasets" and "Real dataset", contain an explanation of the dataset(s) chosen along with the combination of parameters tested for each one of them and their experimentation pipeline. After that, they provide an analysis of the resulting plots.

4.1 Artificial datasets

All the artificial datasets have been generated by means of the `scikit-learn` function `make_blobs()` as a ***d*-dimensional mixture of *K* Gaussian distributions with *n* instances**. The component overlapping lower than 5% mentioned in the paper was not included because the aforementioned function has not any parameter for that purpose. The tested parameter values for this batch of experiments have been: $K \in \{3, 9\}$, $d \in \{2, 4, 8\}$ and $n \in \{1000, 10000, 100000, 1000000\}$.

The **pipeline** of this experimentation is focused on **testing the fundamental aspects that motivated the development of *RPKM*** (Section 2): the reduction of distance computations d of the algorithm in order to improve the time complexity of *Weighted Lloyd* –$O(|P|Kd)$– and the impact of this new approximation to the K partition problem in the cluster quality. Thus, the following subsections analyze the aforementioned aspects and conclude with the relation observed between both metrics: "Distance computations evaluation", "Quality of the approximation evaluation" and "Efficiency-quality trade-off evaluation", respectively.

4.1.1 Distance computations evaluation

This first experiment tests the **influence of the dataset size**, n, **in the number of distance computations** performed by *K-means++* (denoted as *KM++*), *mini batch K-means* (denoted as *MB* <batch size>) and *RPKM* (denoted as *RPKM* <*m*>) for different values of the number of clusters, K, and the dimensionality of the data, d (Figure 3).

This experiment shows that ***RPKM* beats *K-means++*** (and therefore *mini batch K-means* also) **in distance computations when the dimensionality of the data *d* and the number of partition index *m* are high enough**. This is especially worrisome in some extreme cases such as *RPKM 6* with K=9 and d=8, where the order of magnitude of distance computations performed is even greater than *K-means++*. However, this does not happen for lower values of m under the same circumstances, where the number of computations remains constant independently of the dataset size; so we could infer that this problem comes directly from the repetitive computation of *Weighted Lloyd* over partitions with a larger number of *representatives*, since their number increases exponentially ($\sim 2^d$) and this is critical in datasets with a large dimensionality.

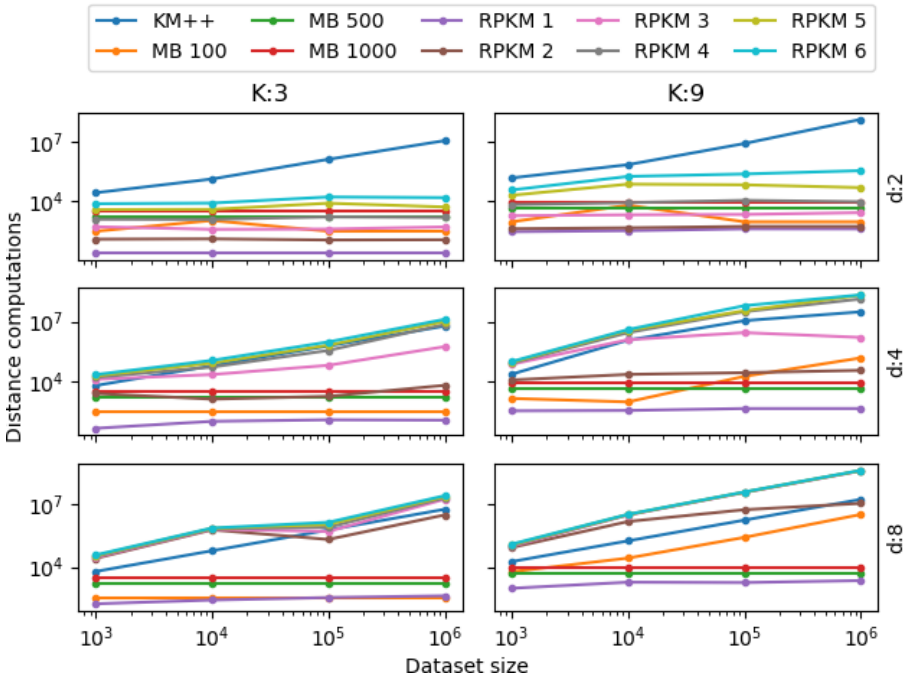


Figure 3: Influence of the dataset size in the number of distance computations performed by *K-means++*, *mini batch K-means* and *RPKM* for different values of K and d in artificial datasets.

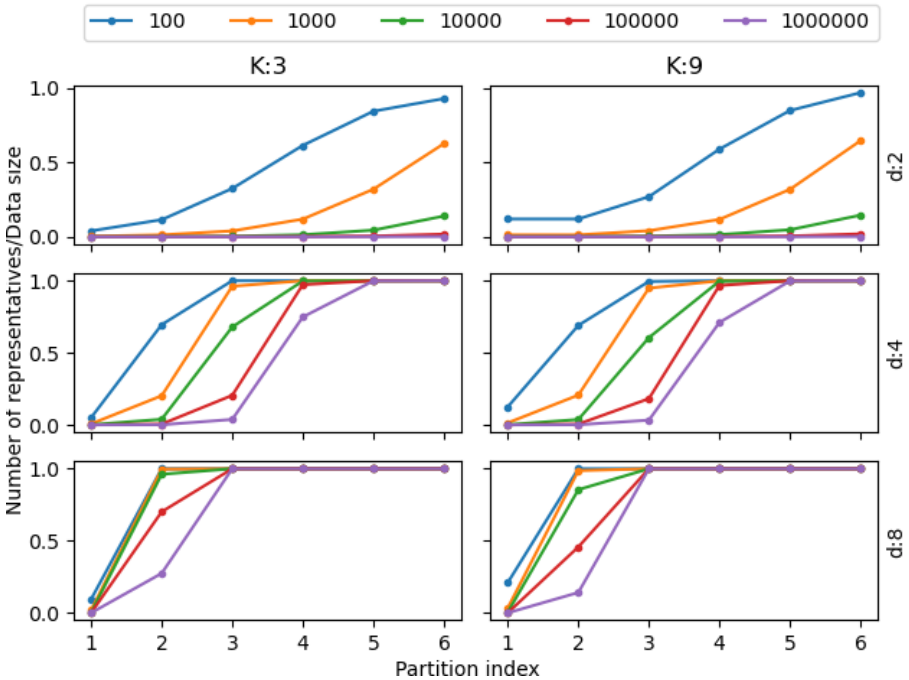


Figure 4: Influence of the number of iterations of *RPKM* in the ratio of number of *representatives* per data size for different values of K and d in artificial datasets.

We can prove this reasoning empirically in Figure 4, where we can see that the main factors of the increase in the number of *representatives* are the dimensionality of the data and the number of partitions computed, as the number of clusters do not change dramatically any curve. This is because K makes the number of distance computations increase linearly but m and d exponentially –the order of *representatives* in the last partition is of $O(2^{m \cdot d})$–. This explains why ***RPKM* can even beat the number of distance computations of *K-means++***, since it can **easily reach the situation where $|P|=n$ in high dimensionality data** and thus repeat the original *Lloyd's algorithm* a number of times until the maximum number of iterations is computed (i.e. As we can see in Figure 3, *RPKM 2* surpass the distance computations of *K-Means++* using data with d=8 and $n < 1{,}000{,}000$ because max(*representatives*) ~65,000; but not when n=1,000,000 as $|P| < n$). Thus, *early stopping* using the *convergence* criterion is key in this algorithm.

4.1.2 Quality of the approximation evaluation

This second experiment tests the **influence of the partition index**, m, **in the *standard error* of *RPKM* over *K-means*** (Formula 5), which measures the percentage of error of *RPKM* with respect to *K-means* under the same circumstances, performed by different data sizes n for different values of the number of clusters, K, and the dimensionality of the data, d (Figure 5).

$$\rho(m) = \frac{E^*_m - E_m}{E_m} \tag{5}$$

Notes on Formula 5.

- E_m define the error obtained by *RPKM* at the m^{th} step for some dataset D.
- E^*_m define the error obtained by *K-means* over the dataset D, taking as initial centroids the obtained by *RPKM* at the m^{th} step.

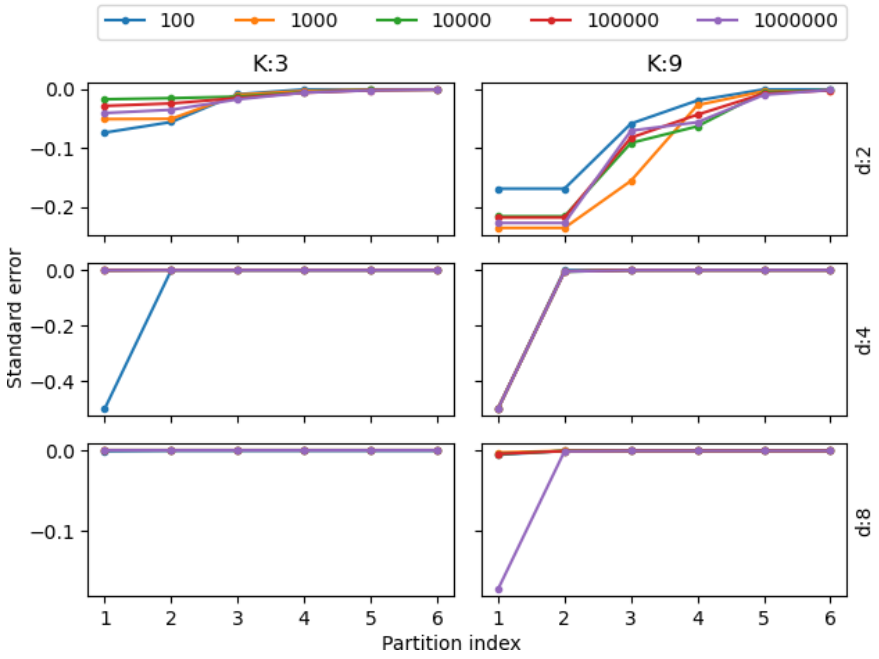


Figure 5: Influence of the partition index in the standard error of *RPKM* over *K-means* performed by different data sizes for different values of K and d in artificial datasets.

As commented before, the main issue of *RPKM* is that it reaches easily the situation where $|P|{=}n$; especially when the dimensionality of the data is large, as the number of *representatives* increases exponentially in terms of d, but also of m. Thus, we would expect the algorithm to converge at low m in order to be at least a competitive option for *massive* datasets, its target. Figure 5 shows that a greater dimensionality of the data makes the algorithm converge faster (reaching the same error as *K-means*) but a greater data size and K slowdown the process. The first is understandable, since the number of *representatives* increase exponentially with respect to the dimensionality of the data (Figure 4); allowing a more precise representation of the dataset since the first partitions and still meeting $|P| \ll n$. The second also makes sense, as the number of possible clusterings increase with both the amount of data points and clusters (provided that $K \ll n$). If we inspect the most demanding situation tested ($K{=}9$ and $d{=}8$), we can see in Figure 5 that *RPKM* converges at $m{=}1$ except for $n{=}1{,}000{,}000$; which converges at $m{=}2$. Furthermore, Figure 3 shows that *RPKM 1* is way efficient than *K-means++* for any n and that *RPKM 2* needs the same order of distance computations as *K-means++* for $n{=}1{,}000{,}000$. Thus, we could affirm that ***RPKM* is a competitive alternative for *small* datasets but there are not conclusive results yet for *massive* datasets**.

4.1.3 Efficiency-quality trade-off evaluation

This third experiment tests the **relation of the distance computations to the error obtained** by *K-means++* (denoted as *KM++*), *mini batch K-means* (denoted as *MB* <batch size>) and *RPKM* (denoted as *RPKM* <*m*>) for different values of the number of clusters, K, and the dimensionality of the data, d (Figure 6).

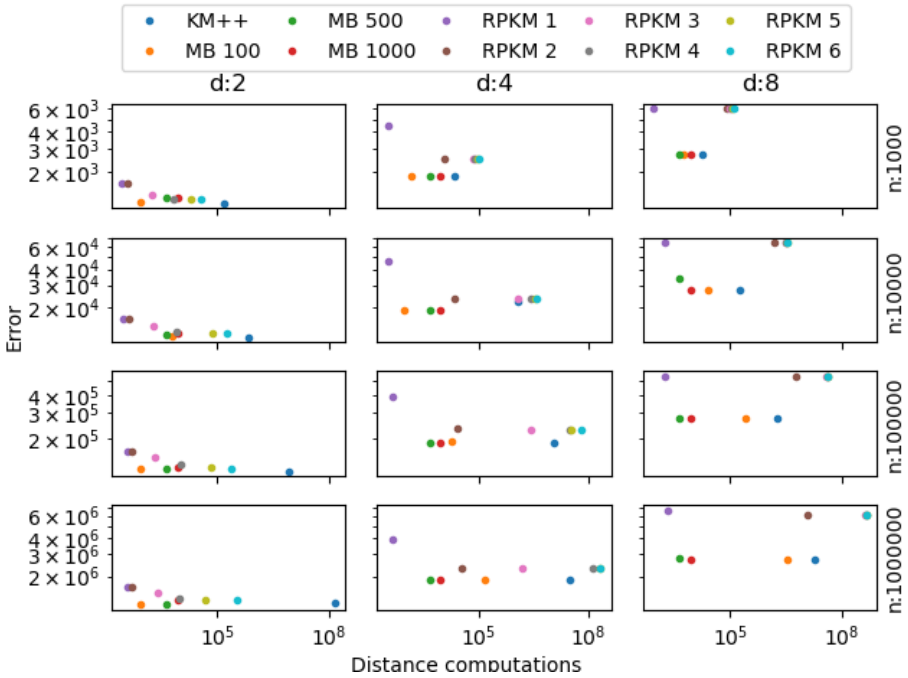


Figure 6: Relation of the distance computations to the error obtained by *K-means++*, *mini batch K-means* and *RPKM* for different values of data size and dimensionality and $K{=}9$ in artificial datasets.

Figure 6 shows that *RPKM* is able to get close to the error obtained by *K-Means++* in datasets of very low dimensionality such as d=2 computing distances in orders of magnitude less, especially when the data size increases. However, as the dimensionality of the data grows, the gap between the best error obtained by *RPKM* and *K-means++* increases. This could be caused by the well-known ***curse of dimensionality***, which states that increasing the number of dimensions of the data space linearly requires an exponential amount of data so that these data remains significant enough to solve the problem. Therefore, **partitioning the space and thus reducing the amount of data points available seem to hinder the clustering task in terms of quality**. This does not happen in *mini batch K-means* because a fixed amount of data points are selected randomly in each iteration and thus all of them will appear with the same probability when the method has finished.

4.2 Real dataset

The real data chosen for this experimentation comes from the ***Gas sensor array under dynamic gas mixtures dataset*** [6], a time-series that measure the reaction of 16 chemical sensors exposed to two different gas mixtures (Ethylene and Methane and Ethylene and CO) at varying concentration levels. These signals were adquired continuosly for about 12 hours without interruption. This dataset consists of **4,178,504 instances and 19 attributes**, one for the relative time (in seconds) of the measurement, another two for the gas mixture concentration (in ppm) and the remaining 16 for the sensor readings. The tested parameter values for this batch of experiments have been the same as in the artificial datasets: $K \in \{3, 9\}$, $d \in \{2, 4, 8\}$ and $n \in \{1000, 10000, 100000, 1000000\}$ –selected by means of a random sampling of the data–.

The **pipeline** of this experimentation is **the same as artificial datasets'** (Section 4.1) except for the removal of the standard error analysis, which did not provide conclusive results because the random sampling of the real dataset was not deterministic and thus it could not be retrieved to use it later in *K-means*. Thereby, the most relevant aspects are tested again (distance computations and cluster quality) but over a different data distribution which is expected to entail a greater complexity than Gaussian's since it comes from a real source.

4.2.1 Distance computations evaluation

This first experiment tests the **influence of the dataset size**, n, **in the number of distance computations** performed by *K-means++* (denoted as *KM++*), *mini batch K-means* (denoted as *MB* <batch size>) and *RPKM* (denoted as *RPKM* $<m>$) for different values of the number of clusters, K, and the dimensionality of the data, d (Figure 7).

Differently to the artificial datasets, ***RPKM* does not beat *K-means++* in distance computations** even for a large number of dimensions d and partition index m in this real dataset; it rather obtains the same order of computations in these cases. This means that **this specific data distribution is not as spread as Gaussian's** and that its points are grouped in reduced areas of the space so that a much more *thinner* partition is needed to reach the situation where $|P|$=n. This condensed distribution also allows the appearance of more executions of *RPKM* with a constant number of distance computations independently of the data size (Figure 7). However, we cannot expect this favorable situation to happen in all real data sources.

The effect of this distribution can also be observed in Figure 8, where the increase of the amount of representatives keeps being exponential with respect to the dimensionality of the data –as it cannot be avoided–, but in a slower pace than artificial datasets (Figure 4).

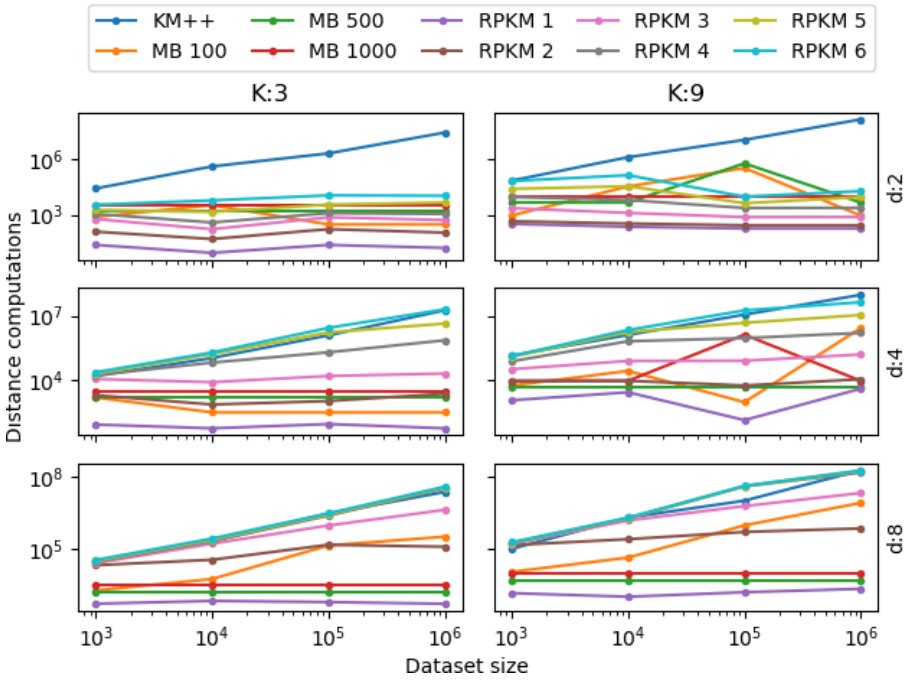


Figure 7: Influence of the dataset size in the number of distance computations performed by *K-means++*, *mini batch K-means* and *RPKM* for different values of K and d in a real datasets.

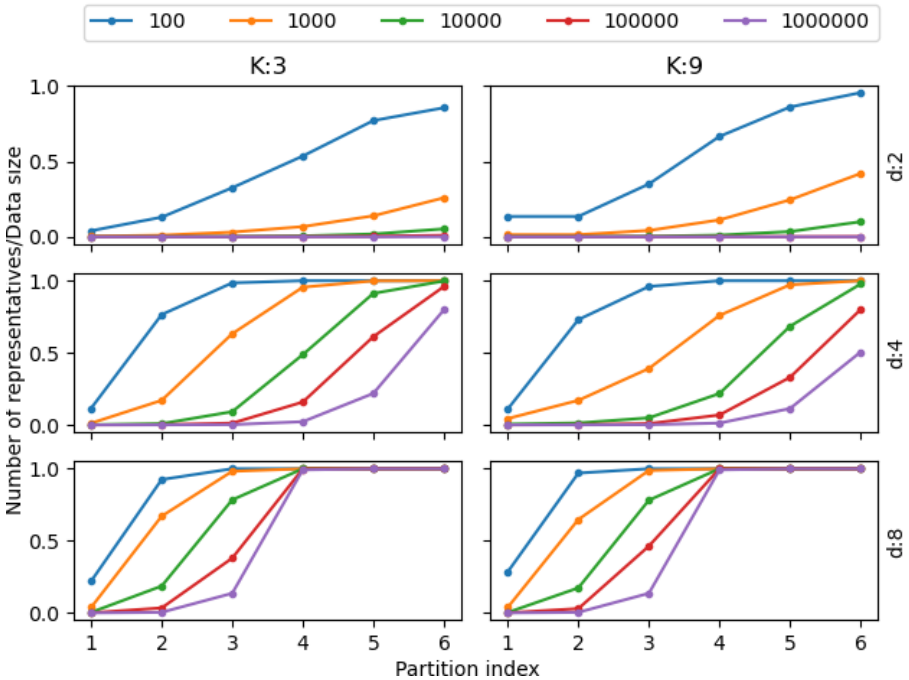


Figure 8: Influence of the number of iterations of *RPKM* in the ratio of number of *representatives* per data size for different values of K and d in a real dataset.

4.2.2 Efficiency-quality trade-off evaluation

This second experiment tests the **relation of the distance computations to the error obtained** by *K-means++* (denoted as *KM++*), *mini batch K-means* (denoted as *MB* <batch size>) and *RPKM* (denoted as *RPKM* <*m*>) for different values of the number of clusters, K, and the dimensionality of the data, d (Figure 9).

As opposed to artificial datasets (Figure 6), where we faced the effect of the *curse of dimensionality*, this time *RPKM* has been able to approximate to *K-means++* and *mini-batch K-means* in terms of performance for all the sizes and dimensions of the data tested except for d=2 and $n > 10{,}000$ (an example of low-dimensionality data and large size, which is the case where *RPKM* has more difficulty to converge). Again, it seems that the already clustered distribution of the dataset has helped the algorithm, since **reducing the number of data points in this case does not affect that much the information about the real distribution of the data**.

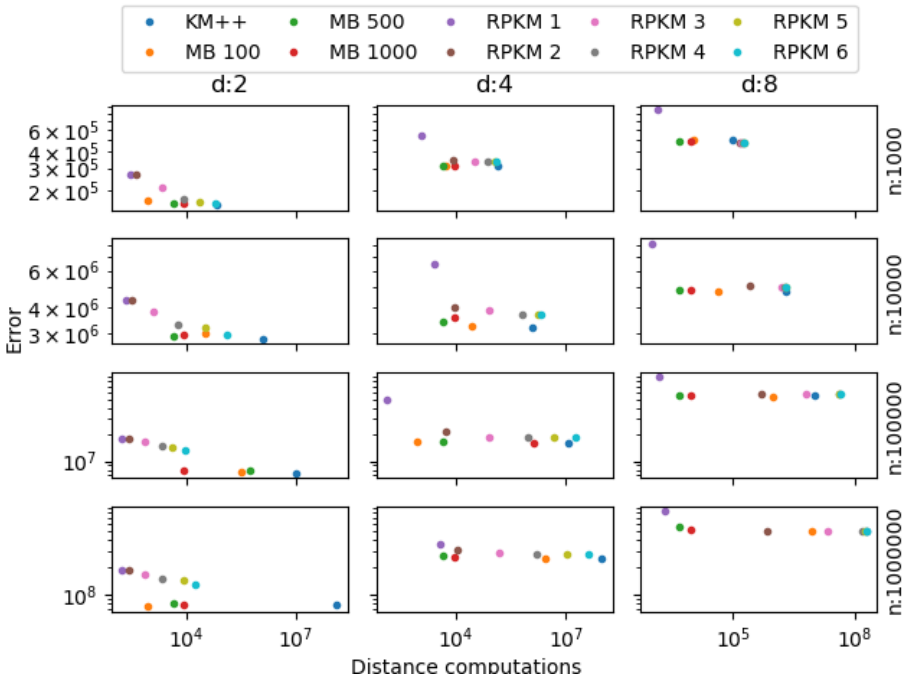


Figure 9: Relation of the distance computations to the error obtained by *K-means++*, *mini batch K-means* and *RPKM* for different values of data size and dimensionality and K=9 in a real dataset.

5 Conclusions

The **goal of *RPKM* is to reinvent the widespread *K-means* algorithm in such a way that it is still efficient for *massive datasets***. The authors' analysis of the complexity concluded that the main problem of *K-means* was the amount of distance computations performed during the *Lloyd's algorithm* and then, they proposed the idea of *partition* as a solution. A *partition* is a division of the space where the data to be analyzed is located so that each region contains an unique weighted data point called *representative* thus simplifying the original distribution of the data. Since earlier *partitions* may not provide enough information to obtain good clusters, *RPKM* keeps dividing recursively this space into *thinner* partitions until convergence. Is this approach good enough for its purpose taking into account the experimental results?

The **main problem of *RPKM* is that the number of *representatives***, which are key for an efficient performance (the time complexity of *Weighted Lloyd* is $O(|P|Kd)$), **increase exponentially** –in a greater or lower extent depending of how scatter the data is– in terms of the dimensionality of the data d and the number of recursive partitions computed m in an order of $O(2^{m \cdot d})$. Even though a large dimensionality cannot be avoided, we could expect the algorithm to converge at low m. The good news are that the convergence based on the centroids' displacement is faster as the dimension of the data increases, but the bad news are that it is slower for a larger data size. *Massive* datasets are by definition datasets that comprises a huge amount of data and they do not always have a large dimensionality. Moreover, it has been proven that a greater dimensionality may compromise the quality of the clustering much more than in analogous methods such as *mini batch K-means* because the same reduced amount of data points are used for the centroids approximation throughout the entire algorithm (*curse of dimensionality*). Therefore, it seems that **the proposal of *RPKM*** is valid both in time complexity and quality terms, but only if certain assumptions are met (*RPKM* has severe problems with datasets of low dimensionality and large size, as seen in the real dataset tested (Figure 9)), so it **is not a good solution for its purpose on a *general basis***.

As a final note, we cannot forget that in some situations the **computation of the *partition* sequence may even be more costly than *Weighted Lloyd*** (the time complexity of *RPKM* is $O(\max(d(n + \sum_{i=2}^{m} |P_i|),\ |P_m|Kd)))$, as commented by the authors; but **there are no experiments nor mention about this issue in the experimentation**, since the experimentation is completely focused on the amount of distance computations of *Weighted Lloyd* when testing the time performance.

References

[1] Capó, M., Pérez A., Lozano J. A. (2016). "An efficient K-means algorithm for Massive Data". *arXiv* e-print: 1801.02949. Link: https://arxiv.org/abs/1801.02949.

[2] Lloyd S. P. (1982) "Least Squares Quantization in PCM". *IEEE Trans. Information Theory*, 28: 129 - 137

[3] Forgy, E. (1965) "Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications". *Biometrics*, 21: 768 - 769 .

[4] Bahmani B., Moseley B., Vattani A., Kumar R., Vassilvitskii S. (2012) "Scalable K-means++". *Proceedings of the VLDB Endowment.*

[5] Sculley D. (2010) "Web-scale K-means clustering". *Proceedings of the 19th international conference on World wide web*, 1177 - 1178.

[6] Fonollosa J., Sheik S., Huerta R., Marco, S. (2015) "Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring". *Sensors and Actuators B: Chemical*, 215: 618 - 629.