An efficient K-means algorithm for Massive Data

An extensive revision

Eric Cañas Tarrasón

Course: 2019/2020

Contents

1 Introduction

The present document reproduces the work done by *Doctor* ***Marco Capó*** *et al.* in his work: *An efficient K-means algorithm for Massive Data* [1], which was a pre-release of his complete work later published into his **PhD Thesis** [2].

The proposed algorithm is a modification of **K-means** [3], which is probably the most well known clustering algorithm used in **Unsupervised Learning**. This modification proposes a way to improve the computational cost of clustering when **working with massive data** coming from large datasets, which is one of the most usual problems derived from the new Deep Learning tendencies.

This document will be divided in three main parts. Firstly, in section 2, the algorithm will be **deeply explained**. In section 3 the algorithm will be **tested** over *Artificial* and *Real Datasets*, reproducing and extending the experimentation done by *Marco Capó* in his original work. In order to make this analysis, an efficient **python implementation** of the algorithm has been developed and attached to this document. This implementation follows the conventions of the ***scikit-learn*** **library** with the purpose of make it familiar with the users. Finally, section 4 is reserved for the conclusions of the work.

2 The algorithm

The proposed algorithm is based on *K-means*. It have the purpose of modifying it for facilitating the fast result extraction when working with large datasets, at the minimal possible cost of worsen the clustering results. This approach focus their performance contribution on the **reduction of distance operation computed**, given that they are considered as the most expensive operations performed by *K-means*.

In order to give a deep explanation about how the algorithm works this section will be divided in two parts. Firstly, the basic *K-means* algorithm will be briefly explained in first subsection. Later, subsection 2.2 will explain how the proposed algorithm, named as *Recursive Partition based K-Means (RPKM)*, modify the original *K-Means* approach.

2.1 K-Means

The defined algorithm part from the base of the **weighted *K-means*** problem. This problem is solved by the weighted Lloyd's algorithm as is shown in pseudocode 1:

Algorithm 1 Weighted Lloyd

```
Data: Set of Points $X$. Set of Weigths $W$ for each Point in $X$. Set of initial Centroids $C_0$. Max iterations $M$.
Result: Set of new Centroids $C_i$. Set of labels $G_i$ indicating the nearest Centroid $C_i^c$ of each point in $X$
$G_0 \leftarrow (C_0, X)$ Assignment Step // Label each Point in $X$ as his nearest Centroid in $C_0$
i=1
while (not StoppingCondition) and ($i < M$) do
    $C_i \leftarrow (G_{i-1}, X, W)$ Update Centroids Step // Redefine each Centroid $C_{i-1}^c$ (in the set $C_{i-1}$) as the weighted mean of the points in $X$ labeled as $C_{i-1}^c$ in $G_{i-1}$
    $G_i \leftarrow (C_i, X)$ Assignment Step // Label each Point in $X$ as his nearest Centroid in $C_i$
    i = i+1
end
return $C_i$, $G_i$
```

This algorithm is based on two main Steps:

1. **Update Centroids Step:** Where K *Centroids* (K points who represents a bigger set of points) are **set as the weighted mean of the points which they are representing**. The weight of these points are given by an **initial set of weight** correspondencies. Since the update of a *Centroid* have a recursive definition, it must be decided how to set it in time 0. For the current case it will be decided randomly (**Forgy's initialization**[4]). However, this initialization will be concerning the following subsection.

2. **Assignment Step:** For each point in X a label will be assigned. This label will correspond to **its nearest *Centroid***. Following this assignment the set of points with the label c will conform the *Cluster* c, represented by the *Centroid* C^c.

These two steps will be executed iterative until a ***Stopping Condition*** is reached or a quantity of iterations reached.

Stopping Condition is the way how the algorithm detects the convergence. The most commonly rule used as *Stopping Condition* is a **threshold** about how much the *Centroids* have moved between two consecutive iterations ($\zeta(||C_{i-1} - C_i||^2)$).

Figure 1 shows graphically how this method works.

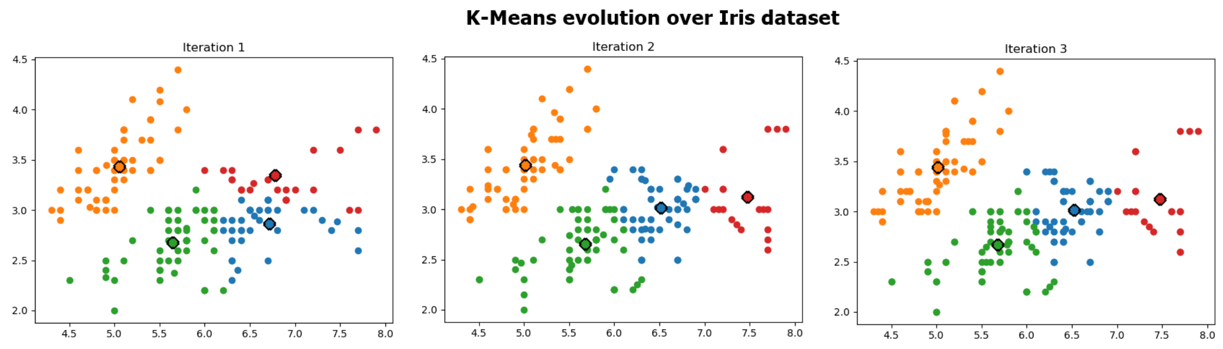

Figure 1: Three first iterations of the Weighted Lloyd algorithm running over Iris dataset.

Given this definition, the complexity of the Weighted Lloyd algorithm in the worst case is **O(KNxD)**. Where K is the number of *Centroids* required, M the maximum number of iterations, x the amount of samples (rows) in the dataset X and D the dimensions (columns) of this dataset X.

2.2 Recursive Partition based K-Means

Aiming to reduce the distance computations calculated by the algorithm, *Recursive Partition based K-means* (*RPKM*) proposes to recursively compute the *Weighted Lloyd* algorithm through **iterative sub-partitions of the space**. It is extremely useful when the algorithm works with massive datasets since it allows to obtain **recursive approximations** of the solution with less computation than a single full dataset iteration.

Pseudocode 2 shows how this algorithm works.

Algorithm 2 Recursive Partition based K-Means

Data: Set of *Points* X. Max iterations M. Number of Clusters to extract K.
Result: Set of *Centroids* C_i. Set of labels G_i indicating the nearest *Centroid* C_i^c of each point in X.

```
BoundingBox ← Hypercube circumscribing the space defined by $X$
// Divide the BoundingBox in $2^1$ equal sized sub-spaces at each dimension
$Partitions_0$ ← PartitionateSpace(BoundingBox, $2^1$)
// For those subdivisions with at least a point inside, take the mean of its points as
   representative, and its number of points as a weight.
($Representatives_0$, $W_0$) ← $TakeRepresentatives$(Partitions$_0$, $X$)
$C_0$ ← Select $K$ Random Points From $Representatives_0$
i=1
while (not StoppingCondition) and ($i < M$) do
    // Max of $M$ iterations of the Weighted K-Means over the actual partitions (code 1).
    ($C_i$, $G_i$) ← WeightedLloyd(X=$Partitions_{i-1}$, W=$W_{i-1}$, $C_0$=$C_{i-1}$, N=N)
    $Partitions_i$ ← PartitionateSpace(BoundingBox, $2^{i+1}$)
    ($Representatives_i$, $W_i$) ← $TakeRepresentatives$(Partitions$_i$, $X$)
    i = i+1
end
return $C_{i-1}$, $G_{i-1}$
```

Most usual *StoppingCondition* used is the same explained in subsection above ($\zeta(||C_{i-1} - C_i||^2) < TH$).

As explained in pseudo-code 2 the algorithm can be summarized as follow:

1. **Divide the space** in 2^d subsections (where d is the number of dimensions of the search space).
2. For each subsection generated **decide a prototype which represents the points** within it (representative), and an **associated weight** with the amount of points that it describes.
3. **Decide** K **initial clusters** randomly (*Floyd Initialization*). They will be the *Actual Clusters* at iteration 0.
4. Using these representatives as points, its weights, and the *Actual Clusters* as initial *Clusters* (C_0), **execute the *Weighted Lloyd* algorithm**. This algorithm will return the new *Actual Clusters*.
5. **Until the stop condition is satisfied** or max iterations has been executed, divide each subsection in 2^d additional subsections and return to the step 4.
6. **Return** the latest *Actual Clusters* obtained and the correspondence of each label in X (G).

It is important to remark that, with this definition, the algorithm only allows to be executed with a number of *Initial Clusters* K, lower than 2^d. For the sake of simplicity, it was defined like this on code 2, however, for cases where $K > 2^d$ a **bigger initial sub-partition** should be executed, generating at least K divisions. For this cases, step 1 must divide the space in 2^{d^m}, being m the nearest integer logarithm of K in base 2^d (For example: for $d = 2$ and $K = 6$ the space should be divided in $2^{2^2} = 16$ sub-partitions instead of $2^2 = 4$).

Figure 2 shows how the algorithm would be executed during 4 iterations when $K = 8$.

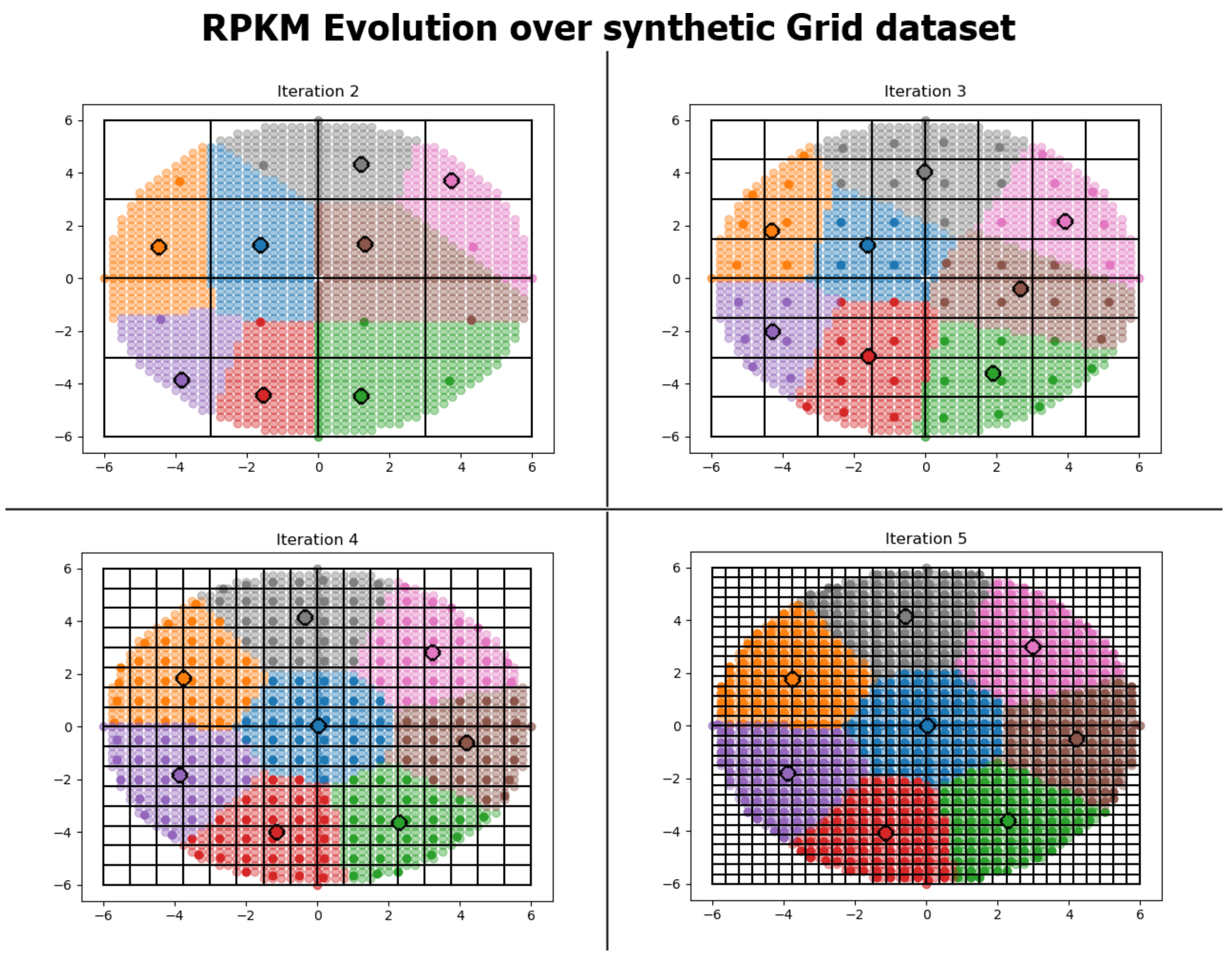

Figure 2: Four first iterations of the *RPKM* algorithm (code 2) running over a dummy synthetic dataset. Points outlined in black shows the *Actual Centroids*. Boxes show each one of the actual *Partitions* ($2^{d^{it}}$). Opaque points inside each box show each *Representative*. The rest of lighten points shows each point at the original dataset X.

If this figure 2 (*RPKM*) is compared with figure 1 (*Weighted K-Means*), can be seen how the most relevant differences are the grid partitions. While figure 1 only uses opaque points ($X_i = dataset$), figure 2 only make computations with one opaque point by *Partition* ($X_i = R_i$), its **representative**.

3 Experimentation

As *Marco Capó* did in his work [1], the purpose of this section is to **analyze the efficiency** of the algorithm. It will be analyzed, not only in terms of the quantity of **distance computations** and the **quality** of the results obtained, as *Marco Capó* did, but also in terms of **total computational time**, for taking in account all the implications of the algorithm. For this purpose, the algorithm will be **compared with two of the most commonly used algorithms** of its field: *MiniBatch K-Means* (*MB*) and *K-Means++*.

For this analysis, results will be tested over the same *Artificial* and *Real Datasets* which *Marco Capó* proposed and varying the following parameters:

- General Parameters:
  - **N**: Size of the dataset.
  - **d**: Dimensions of the instances.
  - **K**: Number of clusters.
- Specific Parameters:
- **M**: Max iterations for the *RPKM* algorithm (For this analysis: $M \in \{1..6\}$).
- **b**: Batch size of the *MiniBatch K-Means* algorithm (For this analysis: $b \in \{100, 500, 1000\}$).

The quality of the results will be measured over the *Standard Error* metric shown on equation 1:

$$p(m) = \frac{E_m^* - E_m}{E_m^*} \tag{1}$$

Where E_m is the error of the *RPKM* algorithm at iteration m and E_m^* the error of *K-Means* when is initialized with the *Centroids* of this m^{th} iteration of *RPKM.* Since this *Error Measure* is the one which *K-Means* optimize, it is insurable that $p(m) \leq 0$.

Each experiment will be repeated 10 times for computing it mean result, in order to avoid unusual casual values.

3.1 Artificial Datasets

This analysis is the easy job for the algorithm. In this case we will generate datasets based on a mixture of K Gaussians, aiming to generate K well defined clusters. This Gaussians, as paper proposed will have an overlapping lower than 5%. Datasets generated will cover the following space:

- **Distance Computations Performed:** $K \in \{3, 9\}$, $d \in \{2, 4, 8\}$, $N \in \{$1000, 1770, 3160, 5620, 10000, 17700, 31600, 56200, 100000, 177000, 316000, 562000, 1000000, 1770000, 3160000$\}$. For N, it differs from the original proposed parameter set ($N \in \{1000, 10000, 100000, 1000000\}$), since **three (log scale) middle points has been introduced** for each proposed value in order to generate richer results. This modification, also makes the top limit **closer** to the top limit proposed for the *Real Datasets.*
- **Quality of the results:** One more time, same space than *Distance Computations* performed, except for the N. Since each N value will be represented by a line on the plot, N space covered has no middle points ($N \in \{1000, 10000, 100000, 1000000, 3160000\}$) for making the results more readable.

3.1.1 Distance Computations Performance

Figure 3 shows the amount of ***Distance Computations*** performed by *K-Means++*, *MiniBatch K-means* and *RPKM* algorithm under the same conditions.

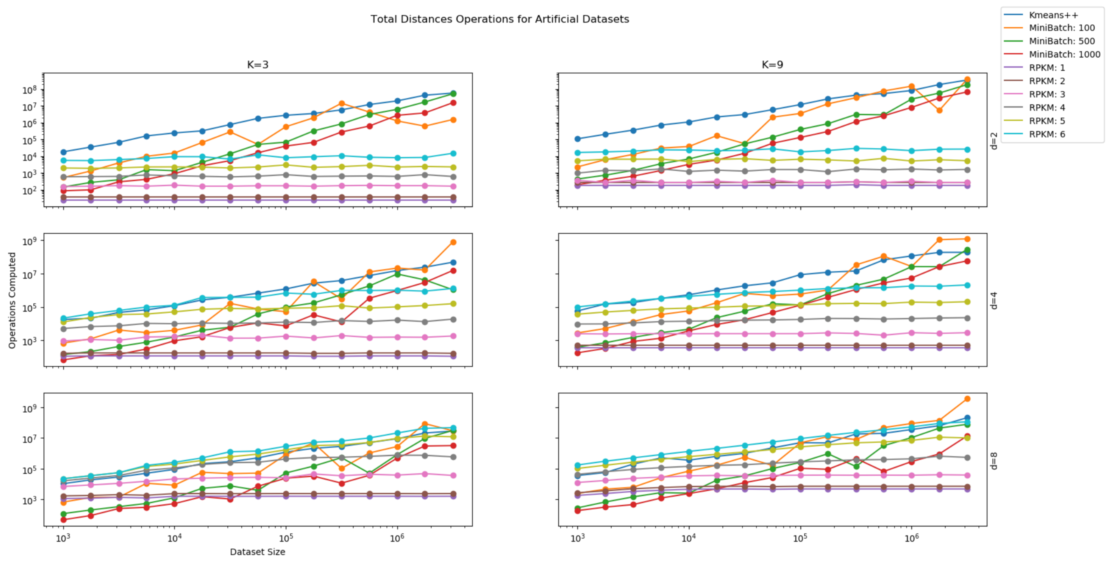

Figure 3: Distance Operations performed by *K-Means++*, *MiniBatchK-means* ($BatchSize \in \{100, 500, 1000\}$) and *RPKM* ($M \in \{1..6\}$). Graph is presented in logarithmic scale, showing differences up to 6 magnitude orders between *RPKM* and *K-Means++*.

As it shows, there is an important remarkable property of the *Recursive Partition based K-Means* algorithm. While *MiniBatch K-Means* reduce its computation as larger is the *Batch Size*, the **complexity order is maintained** with respects to *K-Means++*. In opposition,the complexity order (in terms of *distance computations*) of *RPKM* do not depend on the size of the dataset. This quality makes the algorithm ideal when dealing with its objective: executions with massive data. Since it allows to make recursive approximations to the result with a linear *Distance Computations* complexity.

3.1.1.1 Results are really as good as they sound?

On the original paper there is not any kind of experimentation about **time performance**. On figure 3 has been demonstrated how *RPKM* accomplish its purpose maintaining linear the *Distance Operations* cost when dataset grows up. However, this simplification is not completely appropriate for real cases, when the computation is perceived as the total time spent by the algorithm.

In order to reduce the *Distance Computations* executed by the algorithm, *RPKM* adds to *K-Means* some previous steps with are an **additional computational workload**. In spite of not being *Distance Computations*, these previous steps which computes partitions of the space as well as representatives and weights do **depends on** N, and it would be unfair for the rest of the algorithms to not have them in account.

Figure 4 shows the equivalencies of figure 3 in terms of total computational time.

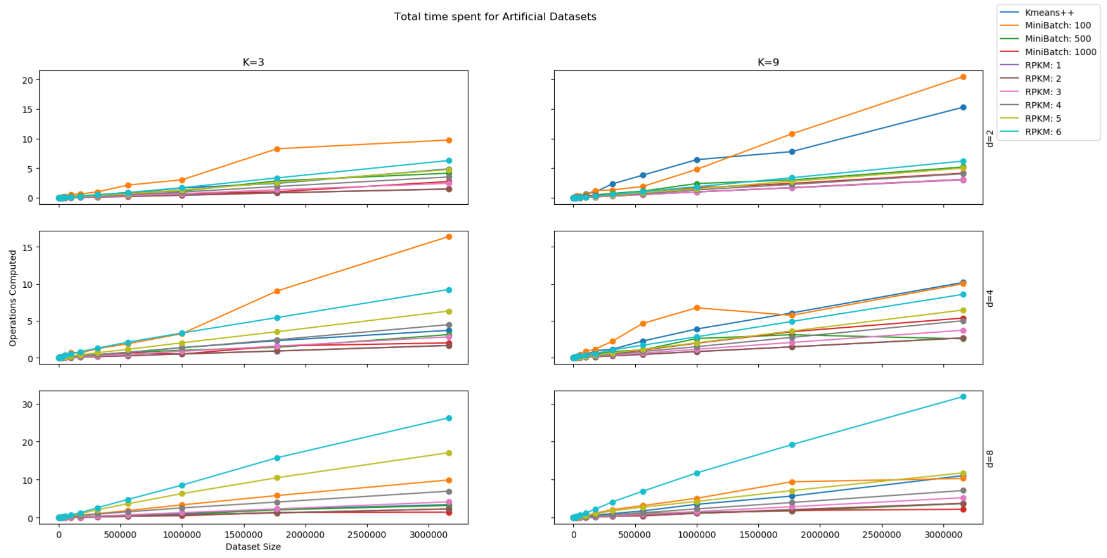

Figure 4: Total computational time consumed by *K-Means++*, *MiniBatchK-means* ($BatchSize \in \{100, 500, 1000\}$) and *RPKM* ($M \in \{1..6\}$) running over an *Artificial Dataset*. Since there are neither constant times nor different orders of magnitude here, like on figure 3, in order to appreciate better the performance of each algorithm, plots have been represented in a linear scale.

As is seen, now, *RPKM* is **not as clear winner as before**. Although it still winning in low dimensional cases, when the dimensionality grows, the algorithm starts to increase its computational time abruptly. The reason for this fact is clear: For usual *K-means* methods, having an additional dimension only supposes to add an additional factor to the distance computation. However, as *RPKM* divides the space in 2^d representatives each time step, it **increases exponentially** the number of points which the algorithm have in account.

For exemplifying it lets suppose a simple case where we are using a dataset with *10 millions* of points: When we run *K-means++*, it will have to deal with *10 millions* of points each iteration, independently from the dimensions of the dataset. However, if we are using *RPKM:4* (which is not the higher which was tested) we will have the space divided in $2^{2^4} = 65,536$ points when dimensions are 2. But, when dimensions are 9, there are $2^{9^4} = 68,719,476,736$ partitions of the space, which will mean to **equally use**, with a high probability, *10 millions* of representatives, but with the **additional overhead** of selecting which is the partition and the weight of each one.

Some **computational optimizations** have been added to this step, in order to be as efficient as possible in those cases, however, **without modifying the original algorithm**, it is completely impossible to deal easily with it for most **difficult cases**. For solving these cases the best solution would be to apply one of the following **restrictions**:

1. **Stop the partitioning** of the space when we detect that we have **as representatives as original points**. This solution would preserve the results expected from the original algorithm but would be **pointless** when we have two identical instances within the dataset.
2. **Stop the partitioning** of the dataset when detecting a **threshold condition**. This optimization would **change the results** expected from the original algorithm, however, as *RPKM* wants to obtain approximated solutions it could be **acceptable**. Two proposals for threshold conditions are:
   (a) When we have **more partitions than the amount of initial points** ($2^{d^n} \geq |InitialPoints|$). This solution would be the best in terms of computation, however it could lead to really bad

approximations when the **range of data of the dataset is extremely large** (for example datasets where the lower value is 10^{-10} and the higher 10^{10}) and there are two or more clusters which are really close (being located within the same partition).

(b) When **the most populated partition have less than** M **points**. This condition is a relaxed version of the one proposed in point 1. It reduces the possible problematic as much as desired but includes an additional parameter which should be tuned.

All these proposes **would solve the computational problems** which we can observe on figure 4. However, there are out of the original *RPKM* proposal[1].

3.1.2 Quality of the results

RPKM proposes an algorithm where the **quality** of the results obtained is **dependant on the amount of iterations** executed (which is a parameter defined a priori). It provokes that, in spite of considerably reducing the required computation, the quality of the results could be damaged when few iterations are executed.

Figure 5 shows the reduction of the *Standard Error* of the results as more *RPKM* iterations are performed.

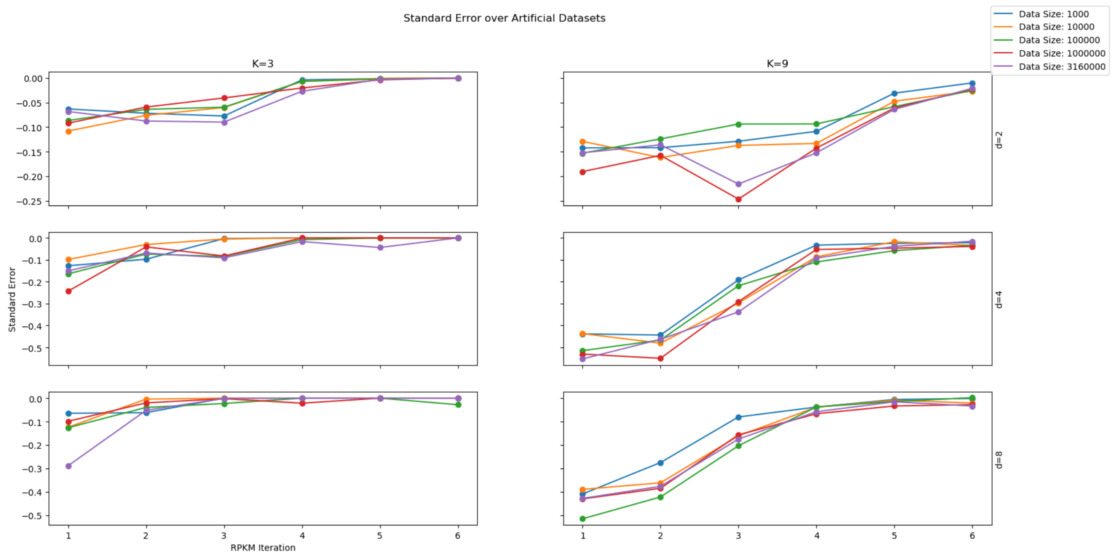

Figure 5: Evolution of the Standard Error as more iterations of the *RPKM* algorithm are performed.

Figure 5 depicts how there is a point around the 5^{th}, iteration where the *Standard Error* turns as **close to 0.** that could be considered marginal (it usually rounds 10^{-5}). This point is usually delayed until the 6^{th} iteration when K increases as can be seen on the right side of the graph.

3.2 Real Datasets

For this analysis, the *gas sensor array under dynamic gas mixtures dataset* [5] has been the one selected. It contains a total of **4,178,504 instances and 19 attributes**. Random sampling this dataset, a set of sub-datasets has been created covering the following space:

- **Distance Computations Performed:** $K \in \{3, 9\}$, $d \in \{2, 4, 8\}$, $N \in \{1000, 1770, 3160, 5620, 10000, 17700, 31600, 56200, 100000, 177000, 316000, 562000, 1000000, 1770000, 3160000\}$. This parameter subspace differs from the one proposed by its original work in the N values selected (*Marco Capó* proposed $\{4000, 12000, 40000, 120000, 400000, 1200000, 4000000\}$), this modification has been made in order to make the results equivalent and directly comparable with the ones extracted for the *Artificial Datasets* (on section 3.1).

- **Quality of the results:** One more time, and for make it comparable, same space than *Distance Computations Performed*, except for the N. The space covered by N is $N \in \{1000, 10000, 100000, 1000000, , 3160000\}$, in order to make the results easily readable.

3.2.1 Distance Computations Performance

Figure 6 shows the amount of Distance Computations Performed by *K-Means++*, *MiniBatch K-means* and *RPKM* algorithm under the same conditions, now in a Real Dataset experiment.

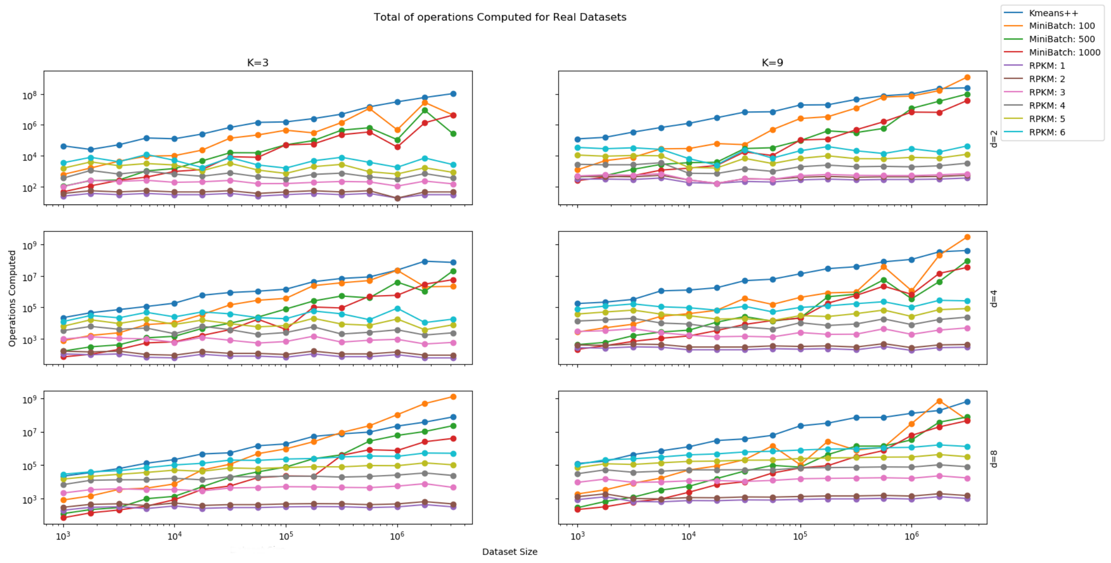

Figure 6: Distance Operations performed by *K-Means++*, *MiniBatchK-means* (*BatchSize* $\in \{100, 500, 1000\}$) and *RPKM* ($M \in \{1..6\}$) over the real datasets sampled from *gas sensor array under dynamic gas mixtures dataset* [5]. Graph is presented in logarithmic scale, showing differences up to 6 magnitude orders between *RPKM* and *K-Means++*.

Conclusions drawn from this experiment are **equivalent** to the ones commented on section 3.1.1 over *Artificial Datasets*.

3.2.1.1 Results are really as good as they sound?

One more time, perform an experimentation only having in account *Distance Computations* and avoiding the previous steps introduced by *RPKM* would be unfair with respect to *K-Means++* and *MiniBatch K-Means*. For this reason figure 7 shows the equivalencies of figure 6 in terms of **total computational time**:

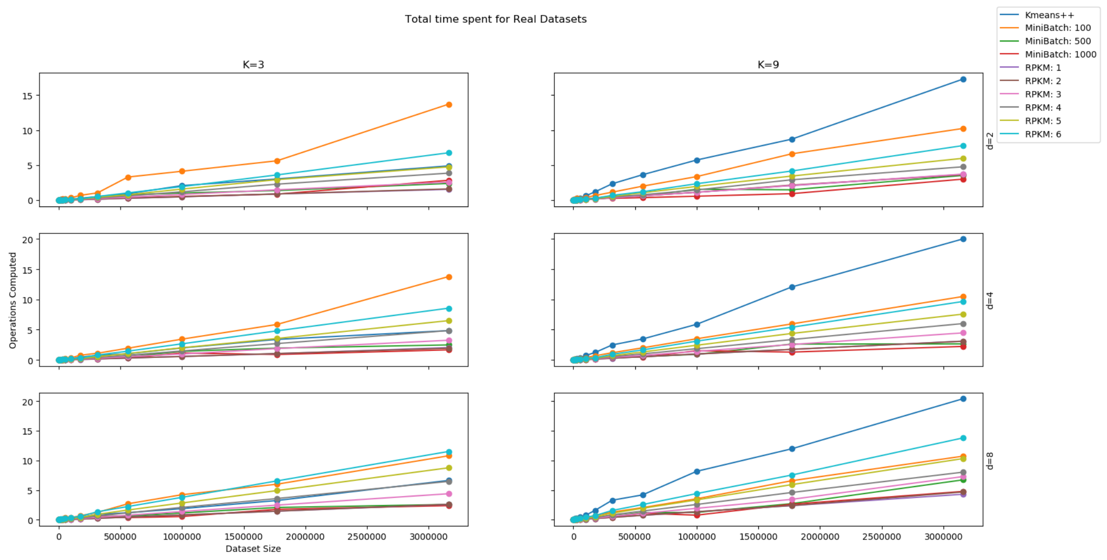

Figure 7: Total computational time consumed by *K-Means++*, *MiniBatchK-means* ($BatchSize \in \{100, 500, 1000\}$) and *RPKM* ($M \in \{1..6\}$) running over a *Real Dataset*. Since here there are no constant times, like on figure 6, in order to appreciate better the performance of each algorithm and the dimensionality problem that each algorithm entails, plots have been not represented in a logarithmic scale.

Here, results are a bit different from the ones obtained by *Artificial Datasets*. Although the **tendency still being the same** in both cases (*RPKM* worsen its results more than the rest as dimensionality increases), in this case, *Kmeans++* have obtained worse results in $K = 9$ cases than *RPKM:6*. This could be due to the following reasons:

1. *Kmeans++* is **not restricted** to a finite number of iterations. Supposing the same dataset, while the computational time of *RPKM* is specially driven by its parameterization, in *K-means++* computation is more dependant of how difficult is to find the clusters and converge. It produces to its computational time to be more **dependant on the dataset**.
2. In some cases, the causes which **negatively** affects to *K-means++* could affect **positively** to *RPKM*. When the dataset contains a big mass of **close points pertaining to different clusters**, it is more difficult for *K-means++* to separate them precisely. However, as *RPKM* divides the space in **equally spaced partitions**, those close points will compose an unique representative, decreasing the number of computations for the algorithm. This simplification, could lead to less precise results in terms of centroid boundaries.

In spite of this difference, conclusions about time are very **similar than before**. The benefits which *RPKM* shows measured in terms of *distance operations* are **questioned when the analysis is performed having in account all the factors** that the algorithm implies. In a completely fair comparison, *RPKM* still presenting some benefits, specially when we restrict it to a low number of iterations or working with low dimensionality, but results are not as incredible as a *distance computations comparison* claims, since the computational overhead that the algorithm introduces for partitioning the space can have a high **non negligible cost**.

3.2.2 Quality of the results

Alike when it was experimented with *Artificial Datasets*, it is relevant to observe if there are differences on the evolution of the results quality when experiments are performed over *Real Datasets*.

Figure 8 shows the reduction of the *Standard Error* of the results as more *RPKM* iterations are performed.

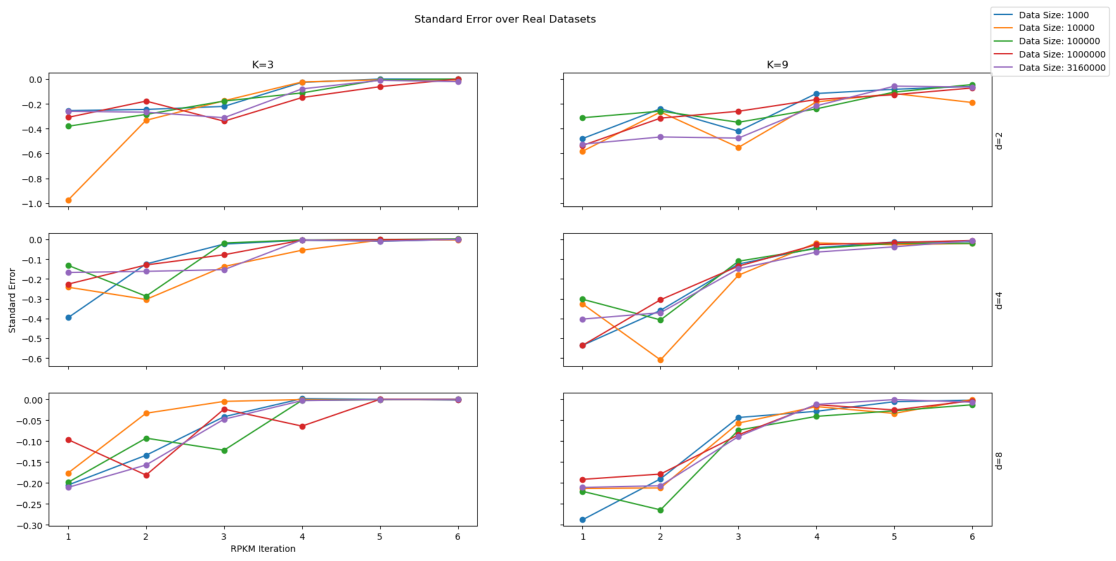

Figure 8: Evolution of the Standard Error as more iterations of the *RPKM* algorithm are performed.

Fit shows how there is, one more time, a point around the 5^{th} iteration where the *Standard Error*, in the most of cases, turns as **close to 0.**. Two dimensional $K = 9$ case, shows slightly higher *Standard Error* scores than the mean, reinforcing the previous hypothesis about how, when considering high Ks, the dataset is composed by close clusters which needs a higher precision (more partitions or more dimensions) for being precisely splitted by *RPKM*. In spite of these particularities, results could be considered **equivalent** to the ones obtained on subsection 3.1.2.

3.3 Experimentation Conclusions

Experimentation over *Real* and *Artificial datasets* have drawn **similar results**. In both cases algorithm presents a good trade-off between its results quality and its *Distance Computations* performed. While, practically in all cases, executing 6 iterations of the *RPKM* is enough for obtaining **results as goods as *K-Means++***, it always requires between **4 and 5 orders of magnitude** less *Distance Computations*. Comparisons between figures 3-5 and 6-8 shows how this great trade-off turns more and **more worth as bigger is the amount of data** used, since the quality of the results is not remarkable damaged when data is increased, while the gap of *distances computed* between *RPKM* and the rest of algorithms turns higher and higher.

However, it is important to remark that the results of the original work were biased by **unfair comparisons** between the algorithms, that favoured *RPKM* results. It was not fair to compare the algorithm only through *Distance Computations* metrics. The reason is that, while algorithms like *K-Means++* focus all its computation in *Distance Computations*, *RPKM* **diversifies its computation**. It provokes that, in order to reduce the amount of *distance operations* computed, it adds a **previous step** with higher computation that depends on the size of the dataset: the *partition step*. Moreover, this *partition step* increases its requirements exponentially when the dimensionality of the dataset increases (figures 4 and 7).

It has been **partially solved** through the implementation, which have in account these cases for solving them in a more efficient way (at the cost of slightly **penalizing the simple cases**), however, for the most complex cases, the complexity that this step acquires still being important. In order to solve this problematic, making the algorithm more robust to these cases, a set of possible **modifications of the algorithm have been proposed** on section 3.1.1.1.

4 Conclusions

In this work, *RPKM* algorithm, designed by *Marco Capó et. al* [1], has been **implemented and analyzed**. This implementation can be found attached to this work, and has been coded in an efficient and scalable way, following the standards of the *scikit-learn* library. The analysis have **repeated and extended the original experimentation proposal**, not only verifying its validity but also **founding some weak points and biases** which it did not include explicitly. Concretely, we have found how the *Distance Computations* analysis hided a computational cost that was not negligible, and which was necessary to have in account for having fair comparisons. For this reason, some **optimizations** have been implemented and **modifications** have been proposed, in order to **make this additional cost as low as possible**.

In general terms, conclusions drawn about this algorithm are the following ones:

- *RPKM* algorithm presents a quite well solution for obtaining **iterative approximations** for the clustering problem, in a reasonable time, when using massive data. In spite of its exponential grow when it needs to deal with high dimensionalities, it offers **way of parameterizing the relationship between computational cost and precision of the result**, through the number of time steps.
- In extreme cases of enormous datasets where *K-means++* could be **unable to complete a single iteration** in a reasonable time, *RPKM* implements a way of completing its **firsts approximations**. For example: If we suppose a case were the original dataset is two dimensional, with 10^{20} data points, *K-means++* would be unable to complete a single iteration. However, as **partitioning cost is lower than the distances computation cost**, it could complete up to its 5^{th} step approximation using no more than 4.3^{10} data points, in its heaviest iteration. Since the 5^{th} iteration of *RPKM* usually have an ***Standard Error* close to 0.** with respect to the *K-means++* output, the results obtained will be **quite similar**, but being generated in a **reasonable computational time**.
- In cases where the **dimesionality of the dataset** is considerably high, it is important to **configure the *RPKM* steps** r in a way that $2^{d^r} < Threshold$. Otherwise, since as the number of partitions grows exponentially with the dimensions, it could rapidly make the algorithm **slower** than *K-means++*.

So it is possible to conclude that, when **having these characteristics in account**, *RPKM* presents a **pretty good solution for dealing with massive datasets**, obtaining well approximations to the solution in a **reasonable time**.

References

[1] Marco Capó, Aritz Pérez and José A. Lozano. An efficient K-means algorithm for Massive Data. May, 2016.

Available at: `https://arxiv.org/pdf/1605.02989.pdf`

[2] Marco Capó, Aritz Pérez and José A. Lozano. K-means for Massive Data. PhD Thesis. April, 2019.

Available at: `https://addi.ehu.es/bitstream/handle/10810/33066/TESIS_CAPO_RANGEL_MARCO%20VINICIO.pdf?sequence=1&isAllowed=y`

[3] J. MacQueen. Some methods for the classification and analysis of Multivariate Observations. Proc. Fifth Berkeley Symp. on Math. Statist. and Prob., Vol. 1 (Univ. of Calif. Press, 1967), 281–297.

Available at: `https://projecteuclid.org/download/pdf_1/euclid.bsmsp/1200512992`

[4] E. Forgy, Cluster Analysis of Multivariate Data: Efficiency vs. Interpretability of Classification, Biometrics 21 (1965) 768

[5] Jordi Fonollosa, Gas sensor array under dynamic gas mixtures Data Set.

Available at: `http://archive.ics.uci.edu/ml/datasets/Gas+sensor+array+under+dynamic+gas+mixtures`