A distance saving approach to the K-means problem for massive data

Marco Capó
Basque Center for Applied Mathematics
mcapo@bcamath.org

Aritz Pérez
Basque Center for Applied Mathematics
aperez@bcamath.org

José A. Lozano
University of the Basque Country UPV/EHU
ja.lozano@ehu.eus

Abstract

In spite of its dependency on the initial settings and the large number of distance computations that it may require, the K-means algorithm remains one of the most popular clustering methods. In this work, we propose an approximation to the solution of the K-means problem that controls the trade-off between the number of distances computed and the quality of the obtained solution. Our approach recursively applies a weighted version of the K-means algorithm over a local representation of the dataset: At each iteration, a thinner partition of the dataset is constructed, by dividing the subsets located at the boundaries of the clustering obtained in the previous iteration. Since all the computations are performed over a small number of representatives, this approach can drastically reduce the number of distance computations especially for problems with a massive amount of data points. Experimental results indicate that our method outperforms well-known approaches, such as the *K-means++* and the *minibatch K-means*, in terms of the relation between number of computations and the quality of the approximation.

Introduction

Clustering techniques have been widely used in different areas, such as artificial intelligence and machine learning. This process determines the intrinsic grouping of an unlabeled dataset by dividing it into a predefined number of disjoint subsets, called clusters. Even when there exist several clustering methods [11], the K-means algorithm remains one of the most popular and widely studied approach [18]. Given a set of data points $D = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ in $\mathbb{R}^d$ and an integer K, the **K-means problem** is to determine the set of K centroids $C = \{\mathbf{c}_1, \dots, \mathbf{c}_K\}$ in $\mathbb{R}^d$ that minimizes the squared-error distortion: $E(C) = \sum_{k=1}^{K} \sum_{\mathbf{x} \in G_k(C)} \|\mathbf{x} - \mathbf{c}_k\|^2$, where $G_k(C) = \{\mathbf{x} \in D \mid \|\mathbf{x} - \mathbf{c}_k\|^2 < \|\mathbf{x} - \mathbf{c}_l\|^2 \text{ for all } l \neq k\}$. This problem is known to be **NP-hard** [1], therefore its solution is commonly approximated by means of heuristics. Among these, one of the most popular is Lloyd's algorithm [13], which is often called the **K-means algorithm**. Given an initial set of centroids (**initialization step**), the K-means algorithm firstly assigns each instance to its closest centroid (**assignment step**), afterwards, considering such an assignment, the set of centroids is updated as the center of mass of each cluster (**update step**). This process is executed until a certain stopping criterion is verified.

As the assignment step is the most time consuming phase of the K-means algorithm, $O(nKd)$, the main goal of this work is to propose a novel algorithm that controls the trade-off between the number of distances computed and the quality the obtained solution. In general, our proposal could be seen as an improvement to the grid based RPKM proposed in [4]. The main idea in [4] is to apply a weighted version of the K-means over the representatives of a sequence of thinner partitions of the dataset (induced by equally sized grids), $\mathcal{P}_1, \dots, \mathcal{P}_m$. For each partition, a set of representatives is obtained by computing the center of mass of each of its subsets. From on iteration to the next, the algorithm is reinitialized using the optimal set of centroids obtained for the previous partition. Unfortunately, in this case, the size of the partition grows exponentially with respect to the dimensionality of the problem, making it not scalable with respect to this factor.

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

In this work, we propose the **boundary K-means** which constructs a sequence of thinner partitions in a more clever way than RPKM. Our proposal is based on the notion of cluster boundary: Given a partition and a set of centroids, the cluster boundary consists of the subsets near the boundaries of the Voronoi tessellation induced by the set of centroids. The idea is that, by dividing these subsets, we can obtain a finer grained representation of the areas that have higher chances of changing their cluster membership, while also reducing the number of representatives. This is, we plan to focus the computational resources in the areas that could have more impact in the solution of K-means.

The rest of this article is organized as follows: In Section 2, we describe the algorithm, in Section 3, we present a set of experiments in which we analyze the effect of different factors, such as the size of the dataset and its dimension over the performance of our algorithm. Additionally, we compare these results with those obtained by the K-means++ and the minibatch K-means. Finally, in Section 4, we define the next steps and possible improvements to our current work.

Contribution

We propose an iterative approximation for the K-means problem that is based on a sequence of thinner partitions of the data at the **cluster boundaries**. We refer to this algorithm as **boundary K-means** (**BKM**). Our proposal attempts to control the amount of representatives, while also placing them in strategical areas of the space in order not to lose quality in the approximation.

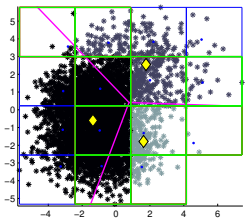

Figure 1: BKM cluster boundary.

The intuition behind this method is to position most of the representatives in the regions where the cluster assignment is harder to determine. These regions correspond to the blocks that are placed around the boundaries of the Voronoi tessellation induced by the set of centroids (purple lines in Fig.1). In order to estimate the location of this region, we will define the concept of **cluster boundary**. The cluster boundary is an area composed by all the blocks of the partition that have as neighbors other blocks with different cluster assignments (green blocks in Fig.1). Overall, BKM seeks to make such local representation more characteristic of the entire dataset by reducing the possible cluster misassignments as we use more resources in those areas where this is more likely to happen. In order to formally define the cluster boundary, we introduce the following definitions:

Definition 1 (Neighborhood of a block) *Given a grid partition of the dataset D, induced by the set of blocks $\mathcal{B}_\mathcal{P} = \{B_1, \ldots, B_t\}$, we define the neighborhood of $B_i \in \mathcal{B}_\mathcal{P}$ as follows:*

$$\mathcal{N}(B_i) = \{B_j \in \mathcal{B}_\mathcal{P} \ : \ B_i \text{ and } B_j \text{ share a } d-1 \text{ dimensional face}\}$$

The selection of this kind of neighborhood has different advantages: i) It allows to characterize in a simple manner most of the blocks (and their corresponding subsets) that can be crossed by a Voronoi boundary, see Fig.1. ii) This type of neighborhood fully covers the boundaries of the block, hence any possible cluster misassignment in the block might be warned by such a $d-1$ dimensional neighbor. It should be highlighted that, by using an appropriate data structure for representing the partition, this neighborhood allows us to identify the cluster boundaries with a computational complexity linear with respect to the number of representatives and the dimension, i.e., the number of $d-1$ dimensional faces of a hyperrectangle increases linearly with respect to d.

Definition 2 (Cluster Boundary) *Given a partition of the dataset D induced by the set of blocks $\mathcal{B}_\mathcal{P}$ and a set of centroids C, we define the cluster boundary of the partition $\mathcal{P}$ with respect to C as:*

$$\mathcal{F}_\mathcal{P}^C = \{B_i \in \mathcal{B}_\mathcal{P} \ : \ \exists\, B_j \in \mathcal{N}(B_i), \text{ where } B_i \in G_a(C) \text{ and } \ B_j \in G_b(C) \text{ with } a \neq b\}$$

As previously mentioned, the blocks in the cluster boundary are used to generate the following thinner partition of the dataset. Observe that the construction of such a thinner partition, as well as of its neighborhood, can be done from the neighborhood of the previous partition, since all the changes occur in the blocks located at the cluster boundary. Moreover, this process can be efficiently implemented through the use of tree data structures similar to the quadtrees [7], and by taking advantage of the symmetry of the neighborhood.

BKM Algorithm

In this section, we formally present the BKM algorithm. This algorithm mainly consists of constructing a sequence of thinner partitions through the cluster boundary corresponding to the previous BKM iteration. Afterwards, a weighted version of the Lloyd's algorithm (WL, see [4]) is applied over the associated set of representatives. In order to determine the boundaries of the partition and, as

the initialization of WL, from one iteration to the next the preceding set of centroids is used. The pseudocode of BKM can be seen in Algorithm 1.

Algorithm 1: BKM Algorithm

Input: Dataset D, number of clusters K, maximum number of iterations m.
Initial grid based partition $\mathcal{P}_1$, satisfying $|\mathcal{P}_1| \geq K$. Set $i = 1$.
Output: Set of centroids C_i.
while *not Stopping Criterion* **do**
  if $i > 1$ **then**
    Step 1 Determine the cluster boundary of $\mathcal{P}_{i-1}$ for the current approximation C_{i-1}, $\mathcal{F}_{\mathcal{P}_{i-1}}^{C_{i-1}}$.
    Step 2 Construct a thinner patition $\mathcal{P}_i$, by partitioning the blocks in $\mathcal{F}_{\mathcal{P}_{i-1}}^{C_{i-1}}$ and compute the set of weights and representatives of $\mathcal{P}_i$.
  end
  Step 3 Update the set of centroids: $C_i = WL(\{\overline{S}\}_{S\in\mathcal{P}_i}, \{|S|\}_{S\in\mathcal{P}_i}, K, C_{i-1})$. Set $i = i + 1$
end

The first step consists of determining the blocks of the previous partition, $\mathcal{P}_{i-1}$, that have $d-1$-dimensional neighbors with different cluster assignments, such blocks are then partitioned in the middle point of each coordinate to construct a thinner partition, $\mathcal{P}_i$ (Step 2). In Step 3, we update the set of centroids by applying WL using the set of representatives and weights determined at the previous step, we take as initialization the approximation for the previous iteration, C_{i-1}. As for the initialization of BKM, we propose two initialization strategies that are the natural adaptations of Forgy's initialization and K-means++ selecting K representatives with a probability proportional to their respective weights.

Brief example BKM

We consider a set of 10000 points from a mixture of three $2D$ Gaussians. We have computed, as a reference, the solution using K-means++. After ten runs, K-means++ has obtained, on average, an error of 11393.45 and it has computed 642000 distances. In Fig.2 and Table 1, we show the evolution of the BKM algorithm up to six iterations, 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.

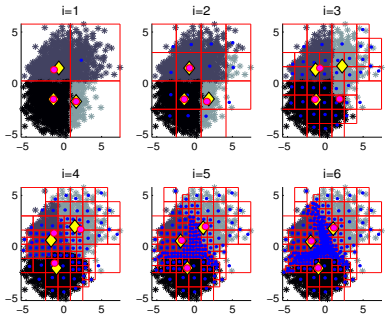

Figure 2: BKM cluster boundary.

i	**Dist Comp**	$\lvert\mathcal{P}_i\rvert$	$E(C_i)$
1	24	4	14050.06
4	2481	89	11501.16
6	8253	307	11390.05

Table 1: BKM iteration results.

From Table 1, we can observe that, even at the fourth BKM iteration, which in this case uses 89 representatives, we have a fairly good approximation of the average best solution found by the K-means++ algorithm for the entire 10000 points. Moreover, BKM computed 0.39% and 1.29% 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 BKM, the associated cost function converges to the best solution obtained by the K-means++. In Fig.2, we can clearly see the intuition behind this method: Transform a random initial set of centroids into a competitive one by using small groups of representatives, most of which are strategically accumulated around the Voronoi boundaries.

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 BKM algorithm. For the purposes of the experimental analysis, we compare the performance of the BKM against K-means++, minibatch K-means and the RPKM algorithm on artificial and real datasets [1]. As stopping criterion for the BKM, we just set a maximum number of iterations, m. Moreover, in this section, we refer to the results obtained after the m-th iteration of BKM and RPKM by **BKM** m and **RPKM** m, respectively. Finally, we denote the solution obtained using MB, with a batch size $b \in \{100, 500, 1000\}$ [2],

[1] Results on real datasets can be found at `https://bitbucket.org/BKM_info/bkm_realdata/src/`
[2] In equivalent experimentations similar batch sizes were used [16].

by **MB** b. The artificial datasets have been generated as a d-dimensional mixture of K Gaussians with an overlap of 5% (approximately). In particular, we set $K \in \{3, 9\}$, $d \in \{3, 9\}$ and $n \in \{10000, 100000, 1000000\}$. For each setting, we have generated 50 replicates of the dataset.

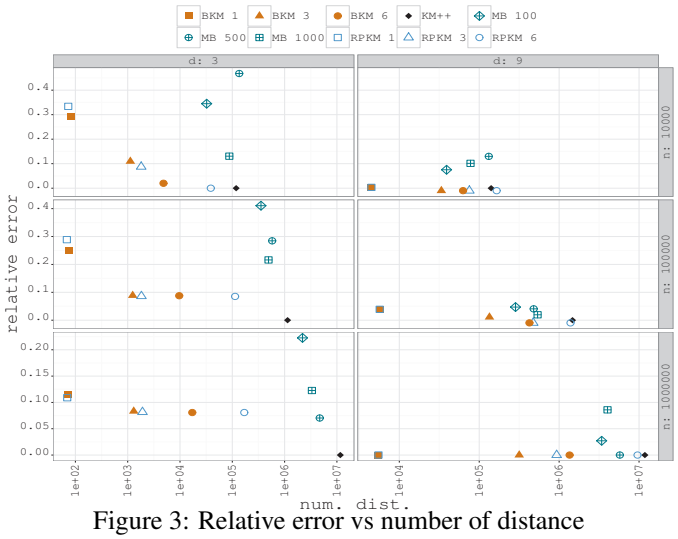

Figure 3: Relative error vs number of distance computations (Artificial dataset case).

In Fig.3, we can observe the trade-off between the number of distances computed and the relative squared-error distortion with respect to KM++ for the different methods. At first glance, we can see that, at the earlier iterations of both BKM and RPKM, there is a drastic reduction in the number of distances computed with respect to the other methods. For the largest number of instances, $n = 1000000$, such a reduction can be of 5 orders of magnitude with respect to KM++ and 4 orders with respect to MB. As we increase the number of instances, the cloud of points associated to KM++ and MB separates from that associated to BKM and RPKM. Clearly, for larger number of instances, KM++ and MB require a much larger number of distance computations in order to achieve a solution of similar or lower quality than the one obtained by BKM.

In the results shown in Fig. 3, we have initialized BKM and RPKM using the weighted version of the Forgy's procedure. We would like to mention that, as the number of iterations of BKM increases, its error tends to the one associated to the K-means procedure initialized with the same set of centroids. Commonly, at an intermediate iteration (in this case, third iteration), the quality of the approximation of BKM had a relative error with respect to KM++ lower than 10% and the number of distances that BKM computes is up to 4 orders of magnitude lower. On the other hand, MB seemed to have the worst trade-off between number of distance computations and quality of the solution. As can be seen in Fig.3, MB usually obtains solutions of quality comparable to those of BKM at the first iteration, while requiring a much larger number of distance computations that oscillated between 1 and 5 orders of magnitude with respect to BKM 1. BKM and RPKM obtains solutions of similar quality. However, as the number of iterations increases, BKM requires to compute fewer distances than RPKM (almost 10 times fewer for $n = 10^6$ and $m = 6$). This effect seems to increase as we consider larger datasets. Clearly, the representatives placed in the cluster boundary provide meaningful information for approaching the K-means problem.

Final Comments

In this work, we present an alternative to the K-means algorithm applicable to massive data problems called boundary K-means (BKM). The intuition behind this approach is to reduce the number of distance computations by applying recursively a weighted version of the K-means algorithm over a reduced number of representatives. In order to describe as accurately as possible the full dataset, most of these representatives are strategically placed in those regions where the correct cluster assignment is harder to determine, i.e., around the boundary of the Voronoi cells, rather than generating an exponential (with respect to the dimension) number of representatives in areas that are unlikely to change their current cluster assignment.

In the experimental section, BKM was compared to K-means++, minibatch K-means and the grid based RPKM. In this analysis, we observed a dramatic reduction in the number of distance computations with respect to all of them, while still obtaining a competitive approximation. Since BKM attempts 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 dataset. Furthermore, at the earliest iterations of BKM, the size of the dataset did not have a relevant impact on the number of distance computations for the associated weighted K-means problem. Thus, the number of computations, especially for massive data applications, can be greatly reduced. An additional benefit of this approach is that different acceleration techniques for the K-means algorithm,

such as [5, 6], could be applied at each iteration of BKM, this will then allow a greater reduction of distance computations. Finally, since the proposed algorithm can be easily parallelized, we also plan to implement it on a parallel framework such as *Apache Spark*.

References

[1] Aloise D., Deshpande A., Hansen P., Popat P.: NP-hardness of Euclidean sum-of-squares clustering. Machine Learning, 75, $245 - 249$ (2009).

[2] Arthur D., Vassilvitskii S.: k-means++: the advantages of careful seeding. In: Proceedings of the 18^{th} annual ACM-SIAM Symp. on Disc. Alg, pp. $1027 - 1035$ (2007).

[3] Davidson I., Satyanarayana A. : Speeding up k-means clustering by bootstrap averaging. In: IEEE data mining workshop on clustering large data sets (2003).

[4] Capó M., Pérez A., Lozano J.A.: An efficient approximation to the K-means clustering for Massive Data. Knowledge-Based Systems (2016).

[5] Drake, J., Hamerly, G.: Accelerated k-means with adaptive distance bounds. In 5th NIPS workshop on optimization for machine learning (2012).

[6] Elkan, C.: Using the triangle inequality to accelerate k-means. In ICML, 3, $147 - 153$ (2003).

[7] Finkel R., Bentley L.: Quad trees a data structure for retrieval on composite keys. Acta informatica 4.1, $1 - 9$ (1974).

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

[9] Har-Peled S., Mazumdar S.: On coresets for k-means and k-median clustering. In: Proceedings of the 36^{th} annual ACM Symp. on Theory of computing, pp. $291 - 300$ (2004).

[10] Hung M., Wu J., Chang J., Yang D.: An Efficient k-Means Clustering Algorithm Using Simple Partitioning, Jour. of Info. Sci. and Eng., 21, $1157 - 1177$ (2005).

[11] Kaufman L., Rousseeuw P.: Clustering by means of medoids. North-Holland (1987).

[12] Kollios G., Gunopulos D., Koudas N., Berchtold S.: Efficient biased sampling for approximate clustering and outlier detection in large data sets. IEEE Trans. Knowledge Data Eng. $15(5)$, $1170 - 1187$ (2003).

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

[14] Peña J.M., Lozano J.A., Larrañaga P.: An empirical comparison of four initialization methods for the k-means algorithm. Pattern Recognition Letters, $20(10)$, $1027 - 1040$ (1999).

[15] Redmond S., Heneghan C.: A method for initialising the K-means clustering algorithm using kd-trees, Journal Pattern Recognition Letters, $28(8)$, $965 - 973$ (2007).

[16] Sculley D.: Web-scale k-means clustering, In Proceedings of the 19th international conference on World wide web, $1177 - 1178$ (2010).

[17] Vattani A.: K-means requires exponentially many iterations even in the plane, Discrete Computional Geometry, $45(4)$, $596 - 616$ (2011).

[18] Wu X., Kumar V., Ross J., Ghosh J., Yang Q., Motoda H., McLachlan J., Ng A., Liu B., Yu P., Zhou Z., Steinbach M., Hand D., Steinberg D.: Top 10 algorithms in data mining. Knowl. Inf. Syst., 14, $1 - 37$ (2007).