What Data-Centric AI Can Do For k-means: a Faster, Robust k-means-d

Parichit Sharma [1 2] **Hasan Kurban** [2 3] **Mehmet Dalkilic** [2]

Abstract

Data-centric AI (DCAI) is an emerging paradigm that prioritizes the quality, diversity, and representation of data over model architecture and hyperparameter tuning. DCAI emphasizes upstream data operations such as cleaning, balancing, and preprocessing, rather than solely focusing on downstream model selection and optimization. This work aims to push DCAI into the model-building phase itself, observing whether benefits downstream can be as significant in a classical, well studied algorithm like k-means. We introduce data-centric k-means (or k-means-d for short). k-means-d is a novel adaptation of k-means clustering that achieves significant speedups while preserving algorithmic accuracy. The key innovation classifies data points as **high expressive (HE)**, impacting the objective function significantly, or **low expressive (LE)**, with minimal influence. By categorizing data points as HE/LE, k-means-d extracts quality signals from data to improve scalability and reduce computational overhead. Comprehensive experimental evaluation demonstrate substantial performance gains of k-means-d over existing alternatives. The novelty lies in the pioneering integration of data-centric principles within a fundamental algorithm's iterative core. By rethinking k-means through a data lens, k-means-d delivers superior efficiency without sacrificing properties like accuracy and convergence, paving the way for infusing data-centricity into other canonical algorithms.

Data-centric AI (DCAI) is a very recent, somewhat radical approach to building better AI applications. It observes that AI models, in large part, have not changed in some time. Indeed, traditional AI focuses on modestly improving model performance above all else; data, however, is continually becoming larger, and more complex, with higher throughput accompanied by new concerns of security, privacy, and ethics. Notably, the number of submissions to the first NeurIPs workshop (https://datacentricai.org/neurips21/) on data-centric AI (DCAI), was in the several hundreds. There was an overwhelming consensus that DCAI is a valid and crucial rethinking of AI. Consequently, data-centric approaches have gained attention as a means to improve the performance of AI models by enhancing the quality and representation of training data itself. However, what's largely missing is a general push of data-centric techniques into the basic framework of machine learning algorithms. To that end, a rethinking of how DCAI can be used to retool existing AI algorithms, must take place. This work introduces k-means-d, a novel data-centric adaptation of the classical k-means algorithm that achieves significant computational speedups while preserving the exactness of the original algorithm.

The work is novel in a number of ways: we are pushing DCAI which is mostly upstream in AI to downstream into the heart of the algorithm. We modify the algorithm to be data-driven by expressiveness (separating high from low expressive data) rather than the traditional full indiscriminate iteration. It should be pointed out that expressive levels of data change, often at each iteration, which means this approach is a deterministic algorithm supplemented by a data-driven heuristic. Data expressiveness is measured by how much the objective function is affected. Convergence is faster in accordance with the growing amount of low expressive (LE) data. Ultimately, what is left is a small pool of high expressive (HE) data, that is ambiguous–noise. We call our approach $k$means-data-centric (or, k-means-d in short). By dynamically identifying HE and LE points during iterations, k-means-d avoids redundant computations on LE points, significantly reducing the computational overhead. k-means-d assigns data identical to k-means, ensuring convergence to the same solution. Through comprehensive experiments on real-world and large synthetic datasets, k-means-d achieve substantial performance gains over it's counterparts. Depending on the dataset, k-means-d achieves

[1]Luddy Center for Artificial Intelligence, Indiana University, Bloomington, Indiana, USA [2]Luddy School of Informatics, Computing & Engineering, Indiana University, Bloomington, Indiana, USA [3]Texas A&M University, Qatar, Doha, Qatar. Correspondence to: Parichit Sharma <parishar@iu.edu>, Hasan Kurban <hasan.kurban@qatar.tamu.edu>.

Data-centric Machine Learning Research (DMLR): Datasets for Foundations Models Workshop, Proceedings of the 41^{st} International Conference on Machine Learning (ICML 2024), Vienna, Austria.

up to 19× reduction in distance computations and 303× speedup in runtime compared to existing faster k-means variant.

The contributions of this paper are as follows: (1) a demonstration that data-centric principles can further enhance the SOTA k-means, (2) a novel and more efficient data-centric k-means (k-means-d), (3) comprehensive experimental study to show that k-means-d outperforms contemporary alternatives in runtime, while reducing distance computations (DC), (4) a systematic integration of data-driven heuristic with the iterative framework of k-means, and (5) a formal proof showing that k-means-d converges to the same solution as k-means. The rest of the paper is organized as follows: Sec. 1 provides an overview of existing work. Sec. 2 present formal elements of this work and describe the algorithm. Experimental setup, dataset details, and results are discussed in Sec. 3. Finally, Sec. 4 discuss broader implications on data-centric AI, provide a summary and some directions for future work.

1. Related work

k-means clustering remains a cornerstone technique in machine learning due to its simplicity, efficiency, and versatility, making it highly relevant across applications Jain (2010); Ikotun et al. (2023). Diverse techniques have been devised to expedite the conventional k-means algorithm. These approaches include, methods that yield the same solution as the original k-means clustering but with enhanced computational efficiency (Pelleg & Moore, 1999; Moore, 2000; Phillips, 2002; Elkan, 2003; Hamerly, 2010; Ding et al., 2015; Kanungo et al., 2002), algorithms that achieve notably faster computation, albeit at the cost of an approximate solution diverging from the standard k-means (Fränti & Sieranoja, 2019; Dav, 2010; Philbin et al., 2007; Philbin & of Oxford, 2010; Fah). Lastly, methods focused on expediting convergence through preprocessing steps, notably centroid initialization (Bradley & Fayyad, 1998), (Bachem et al., 2016b), (Bachem et al., 2016a), (Newling & Fleuret, 2017). Good initialization i.e. k++ often leads to faster convergence, but does not ameliorate the problem of redundant DC

Various strategies for reducing distance calculations have been proposed. For instance, (Pelleg & Moore, 1999) and (Kanungo et al., 2002) employ k-d trees for data partitioning and storage, conducting distance calculations solely with the neighbor nodes of data. However, while k-d tree methods excel in lower dimensions, their performance wanes in higher dimensions (Curtin, 2017). Another widely used alternative is the use of distance bounds to select specific data for performing distance computations. Notably, this line of research has produced several variants (Elkan, 2003; Hamerly, 2010; Hamerly & Drake, 2015; Newling & Fleuret, 2016). However, a recent approach, Ball-$k$means (Xia et al., 2022), adopts a bound-free (referred here as unbounded) method wherein distance computations are confined to data within specific annulus regions, termed as active area. Ball-$k$means has demonstrated similar or better performance compared to existing k-means variants. It's worth noting that the pursuit of reducing distance computations is not exclusive to k-means. In the realm of probabilistic soft-clustering, for example, Expectation-Maximization (EM) algorithm, data expressiveness is used to measure the propensity of datum towards changing its cluster membership (Sharma et al., 2022; Kurban & Dalkilic, 2017b; Kurban et al., 2017; 2021). However, majority of improvements in this area are based on bounded/unbounded DC or model-oriented approach, whereas data is largely treated in an ad-hoc manner. To that end, we propose a novel design, which is faster, exact (results in the same solution) as k-means, and demonstrate a principled application of data-centric ideas for enhancing the venerable k-means clustering.

2. Methods

Table 1 summarize the notations used throughout the paper. The set of clusters at t^{th} iteration is represented as $\mathbf{C}^{(t)} = \{\mathbf{c}_1^{(t)}, \ldots, \mathbf{c}_k^{(t)}\}$. Clusters will be underlined to denote k-means-d-$\underline{\mathbf{C}}$. Similarly, set of cluster mean (centroids) at the t^{th} iteration is denoted by $\boldsymbol{\mu}^{(\boldsymbol{t})} = \{\mu_1^{(t)}, \ldots, \mu_k^{(t)}\}$ for k-means. Centroids will be underlined to denote k-means-d *e.g.*, $\underline{\boldsymbol{\mu}}$. Additionally, we overload the centroid as: $\mu_i(D) = D^{'} \subset D$ to indicate the data ($D^{'}$) assigned to i^{th} cluster. Subscripts are added to midpoint i.e. M_{ij}, to indicate it's the midpoint of line segment connecting centroid μ_i and μ_j.

Table 1. List of notation

Notation	Meaning	Explanation
D	Dataset	$D = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ is set of m vectors
$\mathbf{x}$	Vector	$\mathbf{x} \in \mathbb{R}^d$ is a d dimensional vector
x_i	Scalar	i^{th} element of $\mathbf{x}$
$\|\|\mathbf{x}\|\|$	Norm of $\mathbf{x}$	$\sqrt{\mathbf{x} \cdot \mathbf{x}}$
$\|\|\mathbf{x} - \mathbf{y}\|\|_2$	Distance between $\mathbf{x}$ and $\mathbf{y}$	$\sqrt{\sum_{i=1}^{d}(\mathbf{x}_i - \mathbf{y}_i)^2}$
$\overrightarrow{\mathbf{xy}}$	Vector	A vector from $\mathbf{x}$ to $\mathbf{y}$
$M(\mu_i, \mu_j)$	Coordinates of midpoint (mean) of line segment $\overline{\mu_i, \mu_j}$	$\frac{1}{2}\left(\sum_{i=1}^{d}(\mu_i + \mu_j)\right)$
k	Number of clusters	$k \in \mathbb{N}$
$\mathsf{HE}(\mathbf{x})$	Expressiveness	$\mathbf{x}$ is high expressive
$\mathsf{LE}(\mathbf{x})$	Expressiveness	$\mathbf{x}$ is low expressive
$\mathbf{c}_i$	Cluster index	i^{th} cluster
μ_i	Vector	Centroid of i^{th} cluster

Definition 2.0.1 (Radius) *The radius of a cluster is the distance between it's centroid μ and the furthest member point:*

$$r(\mathbf{c}) = \underset{||\mathbf{x}-\mu||}{\operatorname{argmax}} \{||\mathbf{x}-\mu|| \; s.t. \; ||\mathbf{x}-\mu|| \geq ||\mathbf{y}-\mu||\} \tag{1}$$

$\forall \, \mathbf{x}, \mathbf{y} \in \mu(D)$. ■

Definition 2.0.2 (Neighbors (Xia et al., 2022)) *Given centroids μ_i and μ_j; μ_j is considered neighbor of μ_i, if half of the distance between μ_i and μ_j is less than the radius of cluster $\mathbf{c}_i$. Recall that μ_i, μ_j denote centers of cluster $\mathbf{c}_i$, $\mathbf{c}_j$, respectively.*

$$\frac{1}{2} \, ||\mu_i - \mu_j|| < r(\mathbf{c}_i) \tag{2}$$

Set of all neighbors of μ_i can be represented as:

$$n(\mu_i) = \{\mu_j \mid \frac{1}{2}||\mu_i - \mu_j|| < r(\mathbf{c}_i)\} \tag{3}$$

$\forall \, \mu_i, \mu_j \in \boldsymbol{\mu} \wedge \mu_i \neq \mu_j$.

2.1. Significance of Neighbors

Lloyd's k-means computes distances between data points and all centroids. However, as noted by (Ryšavỳ & Hamerly, 2016), significant computations can be saved by only considering centroids which qualify as neighbors of a given cluster. For instance: if $\mathbf{x} \in \mu_i(D)$ then $\mathbf{x}$ is closer to the neighbors of μ_i than μ_j where $\mu_j \neq n(\mu_i)$. DC between $\mathbf{x}$ and μ_j are, therefore, not required. In Ball-$k$means, neighbors are established by comparing the inter-centroid distances with cluster radius. Additional information such as state of clusters (number of neighbors) is used from the previous iteration to evaluate if the current iteration mandates predetermination of neighborhood. Under specific situations, for example when the neighborhood size grows or when centroids remain relatively stable, this design truncates the time required for finding the neighbors. Ball-$k$means also impose an additional step to sort neighbors to select specific data points during distance computations. However, in the worst-case scenario, computational complexity remains $O(k^2)$, particularly during initial algorithmic phase marked by frequent centroid shifts. Notably, k-means-d obviates the need for sorting neighbors, instead only necessitating the tracking of the nearest centroid. The pseudo code is given in Sec. B.

2.2. Data Expressiveness

The concept of categorizing data as HE or LE was first reported in (Kurban et al., 2017) for probabilistic clustering. The work was subsequently extended to k-means in (Kurban & Dalkilic, 2017a). The authors utilize probabilities

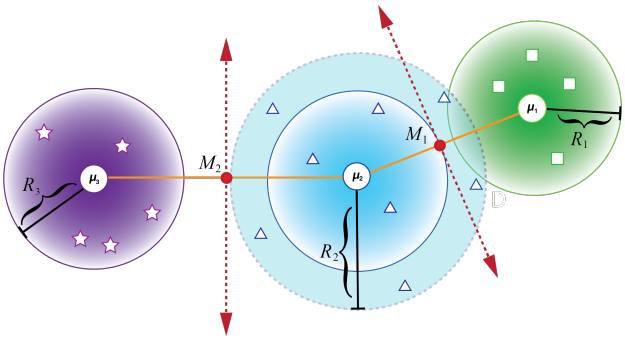

Figure 1. Pictorial representation of neighborhood: Three clusters are denoted with their centers-μ_1 (green), μ_2 (skyblue) and μ_3 (purple). R_1, R_2 and R_3 are the respective radii. M_1 and M_2 are the midpoints of line segments $\overline{\mu_1\mu_2}$ and $\overline{\mu_2\mu_3}$. μ_1 is a neighbor of μ_2 because the length of segment $\overline{\mu_2 M_1}$ is less than R_2. In case of μ_2 and μ_3, none of the clusters satisfy neighborhood criteria, neither μ_2 is neighbor of μ_3 nor μ_3 is a neighbor of μ_2.

to order the data in the nodes of heaps such that HE data is stored in leaves, facilitating quick retrieval during the execution. (Sharma et al., 2022) benchmarked this approach and replaced heaps with balanced binary trees that achieved a significant reduction in the run-time (Kurban et al., 2021). Though, the existing definitions provide a way to effectively separate data into HE and LE, they lack in dynamic treatment of data size. Specifically, (Kurban & Dalkilic, 2017a) use 50% of data in the leaves of the heap (identified as HE), while (Sharma et al., 2022) uses 30% of data as HE. In keeping with the data-centric philosophy *i.e.* data is the first class citizen rather an atomic entity, we propose a novel method of dynamically finding HE data. Particularly, k-means-d finds HE/LE points in a data-driven manner without relying on user specified threshold. This new design ensures adaptability and better scalability to larger datasets. In the following sections, we formalize the notion of LE and HE data, and explain how the native geometry of data is useful in reducing DC. We now present the essential properties of k-means-d that tie DC, and LE/HE together.

2.3. Expressiveness and DC

First we establish that low expressive data does not change its membership, hence it can be safely ignored from distance computations. Consequently, by finding LE data in each iteration-it is possible to actively divert distance computations towards HE data.

Lemma 2.0.1 (Distance and LE) *A data point $\mathbf{x} \in \mu_i(D)$ is defined as* LE*, if $||\mathbf{x} - \mu_i|| < \frac{1}{2}||\mu_i - \mu_j|| \; \exists \mu_j \in n(\mu_i)$.* LE *data point $\mathbf{x}$ will not change its membership; consequently, a distance computation is not required.*

PROOF *We show that: if* $||\mathbf{x} - \mu_i|| < \frac{1}{2}||\mu_i - \mu_j||$, *then* $||\mathbf{x} - \mu_i|| < ||\mathbf{x} - \mu_j||$, *which implies that* $\mathbf{x}$ *is closer to* μ_i; *hence will remain assigned to* μ_i.

Assume: $||\mathbf{x} - \mu_i|| < \frac{1}{2}||\mu_i - \mu_j||$: *Then*

$$2\,||\mathbf{x} - \mu_i|| < ||\mu_i - \mu_j|| \tag{4}$$

From triangle inequality:

$$||\mu_i - \mu_j|| \leq ||\mathbf{x} - \mu_i|| + ||\mathbf{x} - \mu_j|| \tag{5}$$

From Eqs. 4 and 5

$$2\,||\mathbf{x} - \mu_i|| < ||\mu_i - \mu_j|| \leq ||\mathbf{x} - \mu_i|| + ||\mathbf{x} - \mu_j|| \tag{6}$$
$$2\,||\mathbf{x} - \mu_i|| < ||\mathbf{x} - \mu_i|| + ||\mathbf{x} - \mu_j|| \tag{7}$$
$$||\mathbf{x} - \mu_i|| < ||\mathbf{x} - \mu_j|| \tag{8}$$

Therefore, for a data point $\mathbf{x} \in \mu_i(D), \exists \mu_j \in n(\mu_i)$, *if distance between* $\mathbf{x}$ *and* μ_i *is less than* $\frac{1}{2}||\mu_i - \mu_j||$, *then* $\mathbf{x}$ *is nearer to* μ_i, *and distance computation with centroids can be ignored. Note that amongst all neighbors of* μ_i, *it suffice to check if* $\mathbf{x}$ *is LE w.r.t the closest neighbor* μ_j *We note that: when* $d(\mathbf{x}, \mu_i) = \frac{1}{2}d(\mu_i, \mu_j)$ *then membership depends on how ties are broken, this is discussed in Sec. A.1 (Appendix).* ■

Lemma 2.0.2 (Distance and HE) *As a corollary of Lemma 2.0.1, a data point* $\mathbf{x} \in \mu_i(D)$, *is* HE, *if* $||\mathbf{x} - \mu_i|| > \frac{1}{2}||\mu_i - \mu_j|| \; \forall \mu_j \in n(\mu_i)$. *Such a* HE *data point may or may not change it's membership and require a distance computation.* ■

Lemma 2.0.1 and 2.0.2 facilitates the classification of data as either LE or HE solely based on distance. However, it can be further refined by considering the orientation of a data point. For example, let's examine the scenario depicted in Fig. 2. Suppose points A, B, C, and D are assigned to cluster μ_2. As per lemma 2.0.2, all four points would be labeled as HE, necessitating distance computation. Nevertheless, it is evident that A and B are already proximate to μ_2 (oriented towards μ_2) and, thus can be safely exempted from the distance calculation. Only C, D are true HE data. Interestingly, not all points that satisfy lemma 2.0.2 will change their membership. The difficulty lies in discerning between LE and HE without resorting to distance computation. In this instance, the distinction is made by observing the orientation of data in the native space. Specifically, LE points, A and B are oriented towards μ_2 (closer to μ_2) and will remain assigned to μ_2, while C, D are oriented towards μ_1 (closer to μ_1), therefore are HE due to their potential to switch membership, consequently affecting centroid computation. In essence, our objective is to exclusively calculate

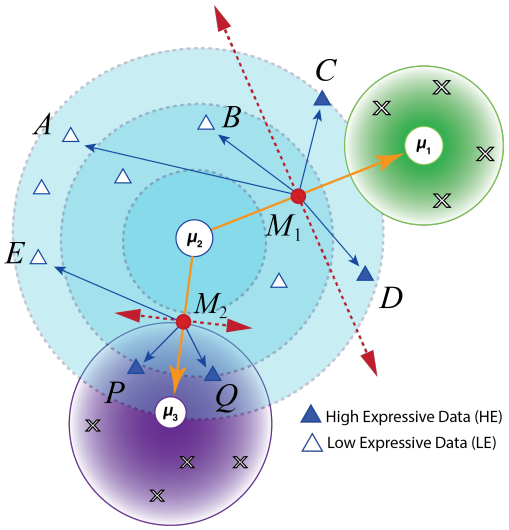

Figure 2. Three clusters are denoted with their centers-μ_1 (green), μ_2 (skyblue) and μ_3 (purple). M_1 and M_2 are the midpoints of line segments $\overline{\mu_1\mu_2}$ and $\overline{\mu_2\mu_3}$. For clusters μ_1 and μ_2, only C, D are valid HE points because they are oriented in the same direction as $\overrightarrow{M_1\mu_1}$. For clusters μ_2 and μ_3, P, Q are HE as they are oriented in same direction as $\overrightarrow{M_2\mu_3}$

distances for HE data (avoid LE data). HE data is formally elucidated below:

Lemma 2.0.3 (Geometrical Orientation and HE) *Recall that* M_{ij} *denotes the coordinates for the midpoint of line segment,* $\overline{\mu_i\mu_j}$. *For a pair of centroids* $\mu_i, \mu_j \wedge \mu_j \in n(\mu_i)$. *A data point* $\mathbf{x} \in \mu_i(D)$ *is* HE *when:*

$$\overrightarrow{M_{ij}\mathbf{x}} \cdot \overrightarrow{M_{ij}\mu_j} \;>\; 0 \tag{9}$$

PROOF
For angle θ *and vectors* $\mathbf{a}, \mathbf{b}$

$$\cos(\theta) \;=\; \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}||\,||\mathbf{b}||} \tag{10}$$

We then define scalar s *as:*

$$s \;=\; ||\mathbf{a}||\cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{b}||} \tag{11}$$

$$\text{sign}(s) \;=\; \begin{cases} 0^o \leq \theta \leq 90^o & > 0 \\ 90^o < \theta \leq 180^o & \leq 0 \end{cases} \tag{12}$$

from the definition of cos.
Using assumption 9 and Eq. 12, we have $cos(\theta) > 0$, *so* $0^o \leq \theta \leq 90^o$. *Hence,* $\overrightarrow{M_{ij}\mathbf{x}}$ *and* $\overrightarrow{M_{ij}\mu_j}$ *are oriented in the same direction; therefore* $\mathbf{x}$ *is* HE. ■

Lemma 2.0.4 (Distance, Orientation, and HE) $\mathbf{x}$ *is* HE *and* $\mathbf{x} \in \mu_i(D)$, **Lemma** 2.0.2 $\rightarrow$ **Lemma** 2.0.3.

PROOF
$\mathbf{x}$ *lies in the quadrants nearest to the neighboring centroid which makes the projection positive.* ■

In essence, k-means-d not only restricts the distance computations to neighbors but also uses Lemma 2.0.4 to determine HE data in each iteration. The pseudo-code for k-means-d is shown in Alg. 1.

Algorithm 1 k-means-d

Require: D (data), k (clusters), $maxIter$ (maximum iterations), ϵ (convergence threshold)
{**Output:** $\boldsymbol{\mu}$ (set of centroids), $\mathbf{C}$ (set of final clusters/partitions)}
Ensure: $\boldsymbol{\mu}^{old}$ = randomly assign k points from data as initial centroids
{Assign data to it's closest centroid; $\mu_1(D), \mu_2(D) \dots \mu_k(D)$}
$counter = 0$
while $counter < maxIter$ **do**
  for $i = 1$ k **do**
    Calculate μ_i^{new} as the centroid of data points in cluster $\mathbf{c}_i$
  end for
  if $||\boldsymbol{\mu}^{old} - \boldsymbol{\mu}^{new}|| \leq \epsilon$ **then**
    break
  end if
  {NEIGHBORS, Sec. B, Appendix}
  $N, M, A, CDIST \leftarrow$ NEIGHBORS($\boldsymbol{\mu}^{new}, R$)
  for $i = 1$ to k **do**
    $D^{'} \leftarrow \mu_i(D)$
    for $\mathbf{x}$ in $D^{'}$ **do**
      if $|n(\mu_i)| == 0$ **then**
        continue
      end if
      {Get closest neighbor; $j \leftarrow n(\mu_i)[1]$}
      {if LE, continue}
      if $||\mathbf{x} - \mu_i|| < CDIST_{ij}$ **then**
        continue
      end if
      for $\mu_l \in n(\mu_i)$ **do**
        if $\overrightarrow{M_{il}\mathbf{x}} \cdot A_{il} > 0$ **then**
          compute $||\mathbf{x} - \mu_l||$; if $||\mathbf{x} - \mu_l|| < ||\mathbf{x} - \mu_i||$ assign $\mathbf{x}$ to $\mathbf{c}_l$.
        end if
      end for
    end for
  end for
  $\boldsymbol{\mu}^{old} = \boldsymbol{\mu}^{new}$
end while
$\boldsymbol{\mu}, \mathbf{C}$

2.4. Proof of Symmetry & Complexity Analysis

Due to page limit constraints, we defer the detailed discussion of proof and complexity to Sec. A.2 and A.3 (Appendix), and share the intuition with the reader. We proof that assignment of data to clusters by k-means-d is identical with k-means, thus k-means-d converges to same local optimum as k-means. Specifically, note that in a given iteration of k-means-d, data is categorized as HE or LE. Recall that HE data is subject to distance computation similar to k-means, therefore all HE points will undergo distance computation and are assigned to the closest centroid (this is same in both k-means and k-means-d). However, LE data is not included in distance computations, results can differ only if k-means-d assign LE data differently than k-means. In the proof, we show that this is not possible. Consequently, iterations of the algorithm are actively channeled towards specific data by dynamically identifying HE and LE points.

3. Experiments & Analysis

Experiments are performed with varying values of m: size, d: dimension, and k: cluster number. Specifically, 20 data sets are used (6 real-world and 14 synthetically generated large data). Additionally, two different centroid initialization schemes, random and k++ seeding are used to confirm seeding does not bias results. Table 2 gives details. Ball-$k$means has been shown to perform at-par or better than existing k-means alternatives, consequently, we compare k-means-d with Ball-$k$means, whereas k-means is used as the baseline. Since all algorithms converge to same solution, performance is evaluated by comparing runtime and distance computations. Further details on computing platform and generation of synthetic data are found in Appendix A.

Table 2. Data used in experiments

(A) Real-world Data

Data	m	d	k
BreastCancer	569	30	K
CreditRisk	1000	7	K
Census	45222	6	K
Birch	10^6	2	K
Cropland	321093	10	K
Twitter	583249	78	K

K $= 5, 8, 12, 15, 20$; source (Asuncion & Newman, 2007)

(B) Synthetic Data

Data	m	d	k
Clus.	10^6	10	K
Dim.	10^6	D	10
Size	M	10	10

M $= \{1, 3, 5, 8\} \times 10^6$;
D $= \{2, 3, 4, 5, 6\} \times 10^2$;
K $= \{2, 4, 6, 8, 10\} \times 10$

3.1. Experiments on Real Data

Experiments are repeated twice (for the two types of seeding) over 10 trials for a total of 20. Iterations (max = 2000) and convergence (threshold = 0.001) are used for stopping; algorithm terminates when either maximum iterations or convergence is reached. Summary statistics are shown via box plots, and average results are reported in the appendix.

Results & Analysis (Random Seeding) Fig. 3 (A) show runtime comparison for random seeding. k-means-d consistently outperforms both k-means and Ball-$k$means. The performance gap widens as m, k increase. In particular, k-means-d achieves significant reduction in distance computations, with maximum distance speedup of $19\times$ on Twitter (Table 5, Appendix C). On the other hand, Ball-$k$means

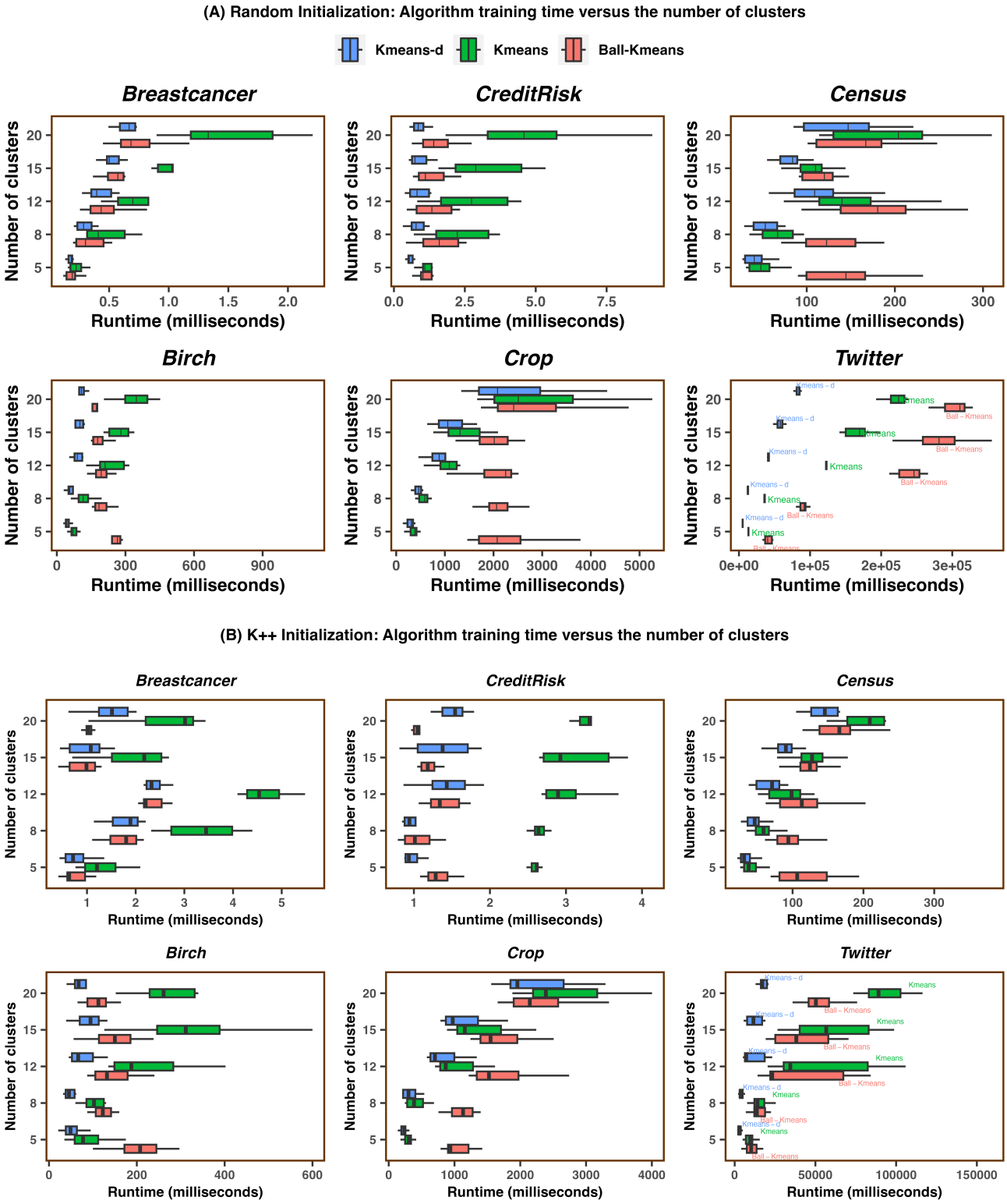

Figure 3. Evaluation on real world data via random and k++ seeding. k-means-d performs significantly better than k-means and Ball-$k$means as m, k increase.

performs better than k-means only on small data (Breastcancer and CreditRisk). On moderately sized data (Census and Birch), Ball-$k$means takes more time than k-means for small values of k, and improves on $k = 12, 15, 20$. On Crop data, Ball-$k$means performs better only for $k = 20$, but runs slower than k-means for the remaining values of k. On Twitter data, Ball-$k$means runs slower than k-means for all values of k. Since, Ball-$k$means does fewer distance computations, its performance appears counter-intuitive. The cause is found in the structures for DC reduction: on small clusters, the computation of neighbors and stable/active areas is relatively costly. k-means performs much better since an extra overhead does not exist. As the data becomes larger, neighborhood computations in specific active areas reduce increasingly larger DC. Further analysis is found in Sec. 3.4.

Results & Analysis (k++ Seeding) Results of k++ initialization are shown in Fig. 3 (B). On smaller datasets-Breastcancer and CreditRisk, k-means-d and Ball-$k$means perform similarly, and Ball-$k$means is only slightly better for $k = 15, 20$. For remaining data sets, k-means-d outperforms Ball-$k$means and k-means, and the performance gap widens with increase in m, k. The design overhead for small data, discussed in the previous sections, is observed again here. On Census and Birch Ball-$k$means does worse than k-means on small values of k. On Crop, Ball-$k$means only performs better for $k = 20$. On Twitter dataset, Ball-$k$means does better than k-means only on $k = 15, 20$.

Experimental results demonstrate that, k-means-d achieves good performance, and is able to sustain higher speed-up across different sizes, cluster count, dimensions and initialization schemes. It should be noted that on smaller datasets, all algorithms execute quickly-the actual runtime is in microseconds (for consistency, time was converted to milliseconds); therefore, even though it may appear that one method is better than others, it is difficult to determine whether the difference is due to algorithmic design or variation in system workload. Nevertheless, k-means-d still has comparable efficiency. Average runtime and DC are reported in Table 6, Appendix C.

3.2. Synthetic Data

Experiments are designed with two objectives: (1) to observe how the algorithms would perform on big data, *i.e.,* large values of m, d and k; (2) to study the impact of varying a single property while holding others constant. To complete the experiments in a reasonable time, we omitted k-means due to its long runtime. Results are given in Table 3. k-means-d demonstrate better performance across experiments and did notably better in scalability experiments. In clustering and dimensionality experiments, the results are average of 10 trials. Scalability experiments are not repeated due to long runtime.

Table 3. Synthetic Data Experiments. Cluster number, dimension, and size (scalability) are varied examining DC and runtimes.

Clustering Experiments $m = 10^6, d = 10$

Clusters	Ball-$k$means		k-means-d			
k	DC	RT	DC	RT	$DC(S)$	$RT(S)$
20	1e10	81.9	**8e8**	**12.3**	12.4	6.6
40	1e10	69.0	**1e9**	**21.2**	14.4	3.2
60	2e10	86.6	**1e9**	**38.7**	16.1	2.2
80	2e10	107.1	**1e9**	**46.4**	17.6	2.3
100	5e10	138.6	**2e9**	**95.5**	21.2	1.4

Dimensionality Experiments $k = 10, m = 10^6$

Dimensions	Ball-$k$means		k-means-d			
d	DC	RT	DC	RT	$DC(S)$	$RT(S)$
200	5e8	40	**1e8**	**13.6**	0.32	4.5
300	3e8	38.2	**1e8**	**15.2**	0.38	3.1
400	5e8	78	**1e8**	**26.9**	0.31	4
500	3e8	59	**1e8**	**24.7**	0.41	2.4
600	3e8	67.3	**1e8**	**30.7**	0.36	2.6

Scalability Experiments $k = 10, d = 10$

$\|D\| = m$e6	Ball-$k$means		k-means-d			
m	DC	RT	DC	RT	$DC(S)$	$RT(S)$
1	2e9	65	**5e8**	**3**	4.7	21.6
3	4e9	402	**1e9**	**7**	4.5	57.4
5	1e10	1110	**2e9**	**18**	6.1	61.6
8	2e10	5464	**2e9**	**18**	7.1	303

DC is distance computations; RT is run time in seconds; $DC(S)$ is the distance speed-up of k-means-d over Ball-$k$means; $RT(S)$ is the runtime speed-up of k-means-d over Ball-$k$means. Entries in **bold** indicate superior performance.

3.3. Quantifying Reduction in DC

Two kinds of experiments are done to observe the effect of cluster number and data size on DC reduction of k-means-d over Ball-$k$means. In the first experiment, samples without replacement of 20%, 40% and 80% from D are made. Ten trials for each sample size are performed, and the average DC is found. Fig. 4 (A) shows k-means-d reduces the most DC, becoming more significant on larger data. In the second experiment, k is doubled for each data set while observing reduction in DC. Ten trials are performed for each k, and the average DC is calculated. The results (Fig. 5, Appendix C) replicate the previous conclusion and show that there are less DC in k-means-d than Ball-$k$means. For instance, on Twitter data, where the observed gain is minimum, Ball-$k$means performed $1.12\times$, $1.42\times$, $1.89\times$, and $2.69\times$ more DC than k-means-d. On Crop dataset, Ball-$k$means executed $2.61\times$, $3.27\times$, $4.35\times$, $5.77\times$ more DC (Table 7, B.0.2)

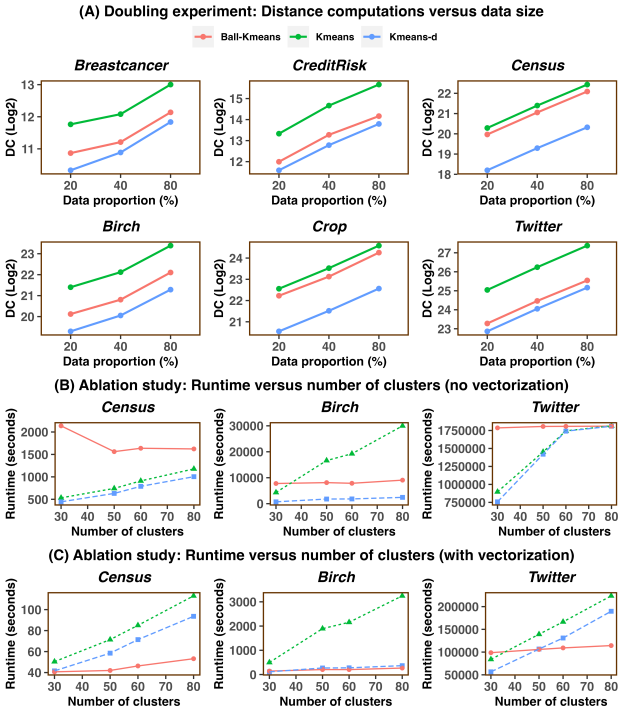

Figure 4. Doubling experiments & ablation study: k-means-d does fewer distance computations as compared to k-means and Ball-$k$means.

3.4. Ablation Experiments & Analysis

As observed in Sec.3.1, on some datasets, Ball-$k$means underperform with respect to k-means and achieves similar performance only for large k. This appears to be independent of seeding even though Ball-$k$means does fewer DC. Interestingly, the current implementation of Ball-$k$means relies on an external linear algebra library that includes vectorization, possibly achieving orders of magnitude performance improvement for some operations. To better understand what role this plays, vectorization is disabled for all algorithms, and while leveraging optimizations is sensible, making comparisons become less meaningful when juxtaposing the algorithms themselves.

Three data sets are selected where the performance of Ball-$k$means improved using $k = 30, 40, 50, 60, 80$. Results without vectorization are shown in Fig. 4 (B). k-means-d still performs better. On Census, Ball-$k$means is the slowest across all values of k, and on Birch, k-means is slowest with the exception at $k = 30$ where it is better than Ball-$k$means. On Twitter, Ball-$k$means takes the most execution time but all algorithms began to converge at $k = 70, 80$. However, k-means-d is still better than both Ball-$k$means and k-means for $k = 30, 40, 50$. The experimental findings with vectorization are given in Fig. 4 (C). When vectorization is enabled, Ball-$k$means generally performs better on larger values of k and agrees with findings in (Xia et al., 2022). On Census, Ball-$k$means is better than other algorithms for all k (except for $k = 30$ where k-means-d performs similar). On Birch, Ball-$k$means and k-means-d perform similar. On Twitter, Ball-$k$means is superior except for $k = 30$ where k-means-d is the fastest. The results indicate that in direct algorithmic comparison without vectorization, k-means-d outperform other algorithms by significant margin. When vectorization is enabled, performance of Ball-$k$means primarily improves for large values of k, and is partially driven by vectorization.

4. Summary & Future Work

This work introduces k-means-d, a novel data-centric adaptation of the classic k-means clustering algorithm. By integrating data expressiveness principles into the algorithm's core, k-means-d achieves significant computational speedups while preserving exactness. It exemplifies the systematic identification of quality signals (HE points) from data that significantly affect the algorithm's convergence. By actively focusing distance computations on HE points, k-means-d avoids redundant calculations for LE points. Experiments show that, compared to it's counterparts, k-means-d performs noticeably better. The work demonstrates the successful application of data-centric thinking in optimizing fundamental ML algorithms. Future work could explore the impact of data expressiveness on robustness and scalability to massive datasets, and applications to other core machine learning techniques.

5. Impact Statement

This work on data-centric k-means (k-means-d) has the potential for significant broader impact in the fields of data-centric AI, machine learning, and data science. By demonstrating how core algorithmic improvements can be achieved by focusing on the data itself rather than just model architecture, this research opens up new avenues for enhancing the efficiency and effectiveness of fundamental machine learning techniques. As data continues to grow in volume and complexity, the ability to extract insights and make decisions quickly and accurately will be a key competitive advantage. Consequently, data-centric techniques like k-means-d could become essential tools in domains such as scientific research, business strategy, and government policy. However, we do not foresee any imminent harmful impact of our work on the society.

6. Code & Reproducibility

Code and reproducibility instructions are shared at `https://github.com/parichit/Data_Centric_KMeans`.

References

An efficient enhanced k-means clustering algorithm. J. Zhejiang Univ. - Sci. A 7, 1626–1633 (2006). URL https://doi.org/10.1631/jzus.2006.A1626.

Web-scale k-means clustering. In Proceedings of the 19th international conference on World wide web, ACM, 2010.

Asuncion, A. and Newman, D. Uci machine learning repository, 2007.

Bachem, O., Lucic, M., Hassani, H., and Krause, A. Fast and provably good seedings for k-means. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016a. URL https://proceedings.neurips.cc/paper/2016/file/d67d8ab4f4c10bf22aa353e27879133c-Paper.pdf.

Bachem, O., Lucic, M., Hassani, S. H., and Krause, A. Approximate k-means++ in sublinear time. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Feb. 2016b. doi: 10.1609/aaai.v30i1.10259. URL https://ojs.aaai.org/index.php/AAAI/article/view/10259.

Bradley, P. S. and Fayyad, U. M. Refining initial points for k-means clustering. pp. 91–99. Morgan kaufmann, 1998.

Curtin, R. R. A dual-tree algorithm for fast k-means clustering with large k. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pp. 300–308. SIAM, 2017.

Ding, Y., Zhao, Y., Shen, X., Musuvathi, M., and Mytkowicz, T. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 579–587, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/ding15.html.

Elkan, C. Using the triangle inequality to accelerate k-means. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pp. 147–153. AAAI Press, 2003. ISBN 1577351894.

Fränti, P. and Sieranoja, S. How much can k-means be improved by using better initialization and repeats? *Pattern Recognition*, 93:95–112, 2019. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2019.04.014. URL https://www.sciencedirect.com/science/article/pii/S0031320319301608.

Hamerly, G. Making k-means even faster. In *Proceedings of the 2010 SIAM international conference on data mining*, pp. 130–140. SIAM, 2010.

Hamerly, G. and Drake, J. Accelerating lloyd's algorithm for k-means clustering. In *Partitional clustering algorithms*, pp. 41–78. Springer, 2015.

Ikotun, A. M., Ezugwu, A. E., Abualigah, L., Abuhaija, B., and Heming, J. K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences*, 622:178–210, 2023.

Jain, A. K. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., Wu, A. Y., Member, S., and Member, S. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:881–892, 2002.

Kurban, H. and Dalkilic, M. M. A novel approach to optimization of iterative machine learning algorithms: Over heap structure. In *2017 IEEE International Conference on Big Data (Big Data)*, pp. 102–109, 2017a. doi: 10.1109/BigData.2017.8257917.

Kurban, H. and Dalkilic, M. M. A novel approach to optimization of iterative machine learning algorithms: over heap structure. In *2017 IEEE International Conference on Big Data (Big Data)*, pp. 102–109. IEEE, 2017b.

Kurban, H., Jenne, M., and Dalkilic, M. Using Data to Build a Better EM: EM* for Big Data. *International Journal of Data Science and Analytics*, 4(2):83–97, 2017.

Kurban, H., Sharma, P., and Dalkilic, M. Data expressiveness and its use in data-centric AI. 2021.

Moore, A. W. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *UAI*, 2000.

Newling, J. and Fleuret, F. Fast k-means with accurate bounds. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 936–944. JMLR.org, 2016.

Newling, J. and Fleuret, F. K-medoids for k-means seeding. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/a8345c3bb9e3896ea538ce77ffaf2c20-Paper.pdf.

Pelleg, D. and Moore, A. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 277–281, 1999.

Philbin, J. and of Oxford, U. *Scalable Object Retrieval in Very Large Image Collections*. Oxford University, 2010. URL https://books.google.com/books?id=mjsJxQEACAAJ.

Philbin, J., Chum, O., Isard, M., Sivic, J., and Zisserman, A. Object retrieval with large vocabularies and fast spatial matching. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2007. doi: 10.1109/CVPR.2007.383172.

Phillips, S. J. Acceleration of k-means and related clustering algorithms. In *Workshop on Algorithm Engineering and Experimentation*, pp. 166–177. Springer, 2002.

Ryšavỳ, P. and Hamerly, G. Geometric methods to accelerate k-means algorithms. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pp. 324–332. SIAM, 2016.

Sharma, P., Kurban, H., and Dalkilic, M. Dcem: An r package for clustering big data via data-centric modification of expectation maximization. *SoftwareX*, 17:100944, 2022. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2021.100944. URL https://www.sciencedirect.com/science/article/pii/S2352711021001771.

Xia, S., Peng, D., Meng, D., Zhang, C., Wang, G., Giem, E., Wei, W., and Chen, Z. Ball k-means: Fast adaptive clustering with no bounds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):87–99, 2022. doi: 10.1109/TPAMI.2020.3008694.

A. Introduction

This document serves as the supplementary and provide additional information that could not be described in the manuscript due to constraints on page limit. For the reviewers, the document provides information such as – supplementary proofs, system and hardware details and additional results.

A.1. Distance and LE

For a data point $\mathbf{x} \in \mu_i(D)$, $\exists \mu_j \in n(\mu_i)$, if $d(\mathbf{x}, \mu_i) = \frac{1}{2} d(\mu_i, \mu_j)$; whether membership of $\mathbf{x}$ changes or not depends on how one choose to break the ties.

DISCUSSION
Assume: $d(\mathbf{x}, \mu_i) = \frac{1}{2} d(\mu_i, \mu_j)$. Then:

$$2d(\mathbf{x}, \mu_i) \quad = \quad d(\mu_i, \mu_j) \tag{13}$$

From triangle inequality: $d(\mu_i, \mu_j) \leq d(\mathbf{x}, \mu_i) + d(\mathbf{x}, \mu_j)$. We can write $d(\mu_i, \mu_j)$ as:

$$d(\mu_i, \mu_j) \quad = \quad d(\mathbf{x}, \mu_i) + d(\mathbf{x}, \mu_j) - \epsilon \ (\epsilon \geq 0) \tag{14}$$

Using Eqs. 13 and 14

$$2d(\mathbf{x}, \mu_i) \quad = \quad d(\mathbf{x}, \mu_i) + d(\mathbf{x}, \mu_j) - \epsilon \tag{15}$$
$$d(\mathbf{x}, \mu_i) \quad = \quad d(\mathbf{x}, \mu_j) - \epsilon \tag{16}$$
$$d(\mathbf{x}, \mu_i) + \epsilon \quad = \quad d(\mathbf{x}, \mu_j) \tag{17}$$

CASE 1: If $\epsilon = 0$:

$$d(\mathbf{x}, \mu_i) \quad = \quad d(\mathbf{x}, \mu_j) \tag{18}$$

From Eq. 18, $\mathbf{x}$ is equidistant from μ_i and μ_j, hence we have a tie for membership; therefore one can choose to either keep $\mathbf{x}$ assigned to μ_i, or re-assign $\mathbf{x}$ to μ_j. It depends on how one choose to implement the algorithm. However, **as long as $\mathbf{x}$ is assigned the same way in k-means and k-means-d** - both algorithms will converge to the same solution. In our implementation, we keep $\mathbf{x}$ assigned to it's current cluster. This is similar to considering $\mathbf{x}$ as LE, since LE does not change it's membership.

CASE 2: If $\epsilon > 0$:

$$d(\mathbf{x}, \mu_j) \quad = \quad d(\mathbf{x}, \mu_i) + \epsilon \ (from\ Eq.\ 17) \tag{19}$$

If $\epsilon > 0$ then Eq. 19 is true when: $d(\mathbf{x}, \mu_j) > d(\mathbf{x}, \mu_i)$. Hence, $\mathbf{x}$ is nearer to μ_i and will not change it's membership. Therefore, distance does not need to be computed.

From **case 1**, we observe that $\mathbf{x}$ can be treated as LE by keeping it assigned to the current cluster. Further, if ties are broken in same way in both k-means and k-means-d then both algorithms will converge to the same solution. From **case 2**, we see that $\mathbf{x}$ is closer to μ_i than μ_j; hence distance computation is not required. Hence, when $\mathbf{x} \in \mu_i(D)$, $\exists \mu_j \in n(\mu_i)$, if $d(\mathbf{x}, \mu_i) = \frac{1}{2} d(\mu_i, \mu_j)$ then $\mathbf{x}$ can be safely treated as LE.

A.2. Proof of symmetry with k-means

We proof that assignment of data to clusters by k-means-d is identical with k-means, thus k-means-d converges to same local optimum as k-means. We show that k-means-d will converge to the same solution as k-means and take the same number of steps while potentially minimizing redundant distance computations affected by the separation of HE and LE data. Specifically, note that k-means-d is similar to k-means except that the distance computation is only done for HE data. First we describe critical steps that both algorithms share. We then prove assignment of data points to centroids is symmetric.

1. ***Initialize:*** Initialize k centroids for both algorithms as follows: $\mu_1^{(1)}, \mu_2^{(1)}, ..., \mu_k^{(1)}$ and $\underline{\mu_1}^{(1)}, \underline{\mu_2}^{(1)}, ..., \underline{\mu_k}^{(1)}$, where $\underline{\mu_i}^{(1)} \leftarrow \mu_i^{(1)}$ for $1 \leq i \leq k$.

2. ***Assign data:*** Calculate pairwise distances between data and centroids-assign data to its nearest centroid. This step is same in both k-means and k-means-d so we will obtain identical partitions i.e. $\underline{\mathbf{c}_i}^{(2)} = \mathbf{c}_i^{(2)}$ for $1 \leq i \leq k$.

3. ***Update centroids:*** The update is based on the assignments in step 2); thus $\underline{\mu_i}^{(2)} = \mu_i^{(2)}$ for $1 \leq i \leq k$.

4. ***Re-assign data:*** At this step, the algorithms begin to diverge, particularly k-means will recalculate distances for all data, but k-means-d will only calculate distances for HE data and assign them to the nearest centroid. Consider a cluster represented by the centroid μ_i. Notice that $\mu_i(D) = \underline{\mu_i}(D)$ due to step-3. After distance computations, the results can differ only if k-means-d assigns data differently than k-means. At this point, we observe by Theorem $A.1$ that this is not possible, and therefore, both k-means and k-means-d will produce exactly the same assignments at each step, thus resulting in identical results.

Theorem A.1 (Symmetry of assignment) *No data point can be assigned differently by k-means and k-means-d.*

Consider a data point $\mathbf{x} \in \mu_i(D)$, assume that $\mathbf{x}$ is assigned to cluster $\mathbf{c}_i$. Since, $\mathbf{x}$ can be categorized as either HE or LE; we examine each case separately.

Case 1: $\mathsf{HE}(\mathbf{x})$: *Since $\mathsf{HE}(\mathbf{x})$, both k-means and k-means-d will compute distances identically. $\mathbf{x}$ will get assigned to it's closest centroid.*

Case 2: $\mathsf{LE}(\mathbf{x})$: *Since $\mathsf{LE}(\mathbf{x})$ k-means-d will not compute distances. $\mathbf{x}$ cannot be assigned to a different cluster $\mathbf{c}_z \neq \mathbf{c}_i$ by k-means.*

PROOF*: From the triangle inequality*

$$||\mathbf{x} - \mu_i|| + ||\mathbf{x} - \mu_z|| \quad \geq \quad ||\mu_i - \mu_z|| \tag{20}$$

Assume that k-means assign $\mathbf{x}$ differently than k-means-d i.e. $\mathbf{x}$ got assigned to $\mathbf{c}_z$. Then

$$||\mathbf{x} - \mu_z|| \quad < \quad ||\mathbf{x} - \mu_i|| \tag{21}$$

Since, $\mathbf{x}$ is LE, hence by Lemma 2.0.1:

$$||\mathbf{x} - \mu_i|| \quad < \quad \frac{1}{2}||\mu_i - \mu_z|| \tag{22}$$

From Eqs. 21 and 22

$$||\mathbf{x} - \mu_z|| \quad < \quad \frac{1}{2}||\mu_i - \mu_z|| \tag{23}$$

Add Eqs. 22 and 23

$$||\mathbf{x} - \mu_i|| + ||\mathbf{x} - \mu_z|| \quad < \quad ||\mu_i - \mu_z|| \tag{24}$$

Eq. 24 contradicts the law of triangle inequality (Eq. 20). Specifically, Eqs. 20-24 forms a contradiction; thus, k-means won't assign LE *data differently than k-means-d.* ■

5. ***Continue with the next iterations:*** After the second iteration, both algorithms will assign data points in exactly the same way; as a result, $\underline{\mathbf{c}_i}^{(3)} = \mathbf{c}_i^{(3)}$ for $1 \leq i \leq k$. Iterations continue, and both algorithms will converge to the same solution.

A.3. Runtime and Space Complexity

Table 4 summarizes the time and space complexity of the three algorithms. For k clusters and data having m points, worst case time complexity of k-means is $\mathcal{O}(mk)$. k-means-d is different from traditional Lloyd's k-means by including 1) neighbor computation, and 2) performing distance computation only between HE and neighbor clusters. Although, our implementation for finding neighbors use a nested loop over k, but the worst case complexity is not $\mathcal{O}(k^2)$, rather $\mathcal{O}(k^2 - k)$. We observe due to symmetric structure of distance matrix: corresponding elements in the upper triangular part of the matrix will be equal to the lower triangular elements, with diagonal elements being 0 i.e. $d(i, j) = d(j, i)$ and $d(i, i) = 0$. Therefore, the inner loop shown in $NEIGHBORS$ pseudo code (Appendix B) will compute distances only if $i < j$, this is akin to calculating values for upper triangular elements. A matrix with k rows contains $\frac{1}{2}(k^2 - k)$ upper triangular elements (ignoring the diagonal elements). Thus, worst case complexity of our neighbor finding algorithm is $\mathcal{O}(k^2 - k)$.

If we denote the average number of neighbor centroids by n and average number of HE data by m', distance computations between HE data and neighbor centroids will cost $\mathcal{O}(m'n)$. Additionally, to determine if a data point is LE, we calculate the distance of each data point to it's own cluster, the runtime complexity of this step is $\mathcal{O}(m)$. Time complexity of k-means-d can be obtained by adding together time complexities of its component steps as: $\mathcal{O}(k^2 - k + m'n + m)$. As the algorithm proceeds, the centroids become stable and an increasingly large fraction of data become LE; by complement HE data decreases, hence $m' \lll m$, and results of our empirical evaluations confirms this belief. In contrast, if we denote the average number neighbor clusters by n and average number of data points for which Ball-$k$means perform distance computations by m'' then time complexity of Ball-$k$means is $\mathcal{O}(k^2 + knlog(n) + nm'' + m)$ (Xia et al., 2022). Experimental evaluations show that $m'' \gg m'$, and in some cases, m'' is one to two order of magnitude more than m'. As a consequence, depending on the data, the worst case complexity of Ball-$k$means could be higher than k-means-d.

Standard k-means requires $\mathcal{O}(k + m)$ space to store k centroids and m data points. Ball-$k$means needs $\mathcal{O}(k^2 + kn)$ space to store, 1) distances between the centroids ($\mathcal{O}(k^2)$), 2) distances between data and centroids for finding the active or stable areas ($\mathcal{O}(kn)$). Space complexity of k-means-d is $\mathcal{O}(k^2 + kn)$ and subsumes the costs of storing mid-points and vectors.

Table 4. Runtime and Space Complexity

Algorithm	Runtime	Space
k-means	$\mathcal{O}(mk)$	$\mathcal{O}(k + m)$
Ball-$k$means	$\mathcal{O}(k^2 + knlog(n) + nm'' + m)$	$\mathcal{O}(k^2 + km)$
k-means-d	$\mathcal{O}(k^2 + m'n + m)$	$\mathcal{O}(k^2 + m)$

A.3.1. Details of computing platform(s) used in the experiments

Experiments on real and synthetic data were done on a 64-bit Ubuntu Linux system with 512GB of main memory and 24 AMD cores. The C++ code used in real-world and synthetic data experiments was compiled with g++ version 9.4.0. The -O3 flag (used for vectorization) was toggled in ablation experiments to compare direct algorithmic performance with and without vectorization.

A.3.2. Synthetic Data Generation Process

The data with large dimensions and size was generated by sampling from a multivariate Gaussian via the MASS [1] package. The cluster centers were randomly selected from the range [1, k+100], where k is the total number of clusters in the data. R

package clusterGeneration [2] was used to generate the initial co-variances matrices and the ratio of highest to lowest eigen values for the co-variance was set to 10. For generating the clustering data, we used the make_blobs() function of the sklearn package, and the data was generated to contain both well separable and overlapping clusters to simulate real-world data sets.

B. Pseudo code for Neighbor Finding Algorithm

Algorithm 2 NEIGHBORS

Require: Centroids $\boldsymbol{\mu}$, Radii $R = \{r(\mathbf{c}_1), \ldots, r(\mathbf{c}_k)\}$
{**Output:** $N = \{n_i, \ldots, n_k\}$ (Neighbors), M (matrix), $CDIST$ (matrix), A (matrix)}
{n_i : neighbors of centroid μ_i, M : stores mid-point coordinates, $CDIST$: stores inter centroid distances, A : stores vectors}
for $i = 1$ to k **do**
  $limit = \infty$
  $closest = 1$
  for $j = 1$ to k **do**
    if $i < j$ **then**
      $dist = ||\mathbf{c}_i - \mathbf{c}_j||$
      $CDIST_{ij} = (\frac{1}{2})dist$
      $CDIST_{ji} = CDIST_{ij}$
    end if
    if $i \neq j \wedge CDIST_{ij} < r_i$ **then**
      Add μ_j to $n(\mu_i)$
      if $CDIST_{ij} < limit$ **then**
        $closest = j$
        $limit = CDIST_{ij}$
      end if
      M_{ij} = midpoint of $\overline{\mu_i, \mu_j}$
      $M_{ji} = M_{ij}$
      $A_{ij} = \overrightarrow{M_{ij}\mu_j}$
    end if
    {Track closest neighbor}
    if $n(\mu_i)[1] \neq closest$ **then**
      $temp \leftarrow n(\mu_i)[1]$
      $n(\mu_i)[1] \leftarrow n(\mu_i)[closest]$
      $n(\mu_i)[closest] \leftarrow temp$
    end if
  end for
end for
$N, M, A, CDIST$

C. Additional experimental results

Results in the following sections show the average values of RT and DC. Note that the reported speed-up is in comparison with k-means, and the corresponding entries will be 0 for k-means. These results provide detailed insight into the performance of different algorithms.

C.0.1. Experiments on real data sets

The results on real-world data sets including random, and k++ seeding based experiments are reported in Table 5 and 6, respectively. Results are average of 10 trials.

Algorithm	Data	Clusters	RT	RT/Iter	DC	DC(S)
Kmeans	Breastcancer	5	0.2276	0.075867	8819	0
Kmeans-d	Breastcancer	5	0.179	0.059667	4255	2.07262
Ball-Kmeans	Breastcancer	5	0.1909	0.063633	5086	1.733976
Kmeans	Breastcancer	8	0.4709	0.117725	21394	0

Kmeans-d	Breastcancer	8	0.2981	0.074525	7148	2.993005
Ball-Kmeans	Breastcancer	8	0.3363	0.084075	9355	2.286905
Kmeans	Breastcancer	12	0.7804	0.15608	38236	0
Kmeans-d	Breastcancer	12	0.4391	0.08782	10075	3.795136
Ball-Kmeans	Breastcancer	12	0.4849	0.09698	13299	2.875103
Kmeans	Breastcancer	15	1.0125	0.2025	50356	0
Kmeans-d	Breastcancer	15	0.544	0.1088	12206	4.125512
Ball-Kmeans	Breastcancer	15	0.5829	0.11658	16399	3.070675
Kmeans	Breastcancer	20	1.4703	0.24505	75108	0
Kmeans-d	Breastcancer	20	0.6956	0.115933	15958	4.706605
Ball-Kmeans	Breastcancer	20	0.7353	0.12255	21612	3.475291
Kmeans	CreditRisk	5	1.1186	0.0799	72500	0
Kmeans-d	CreditRisk	5	0.5765	0.041179	19215	3.773094
Ball-Kmeans	CreditRisk	5	1.0823	0.077307	26667	2.718716
Kmeans	CreditRisk	8	2.2948	0.11474	164800	0
Kmeans-d	CreditRisk	8	0.7988	0.042042	27600	5.971014
Ball-Kmeans	CreditRisk	8	1.5766	0.07883	39588	4.162878
Kmeans	CreditRisk	12	2.7521	0.161888	211200	0
Kmeans-d	CreditRisk	12	0.8654	0.048078	30936	6.826998
Ball-Kmeans	CreditRisk	12	1.3759	0.080935	38621	5.468527
Kmeans	CreditRisk	15	3.2282	0.201762	252000	0
Kmeans-d	CreditRisk	15	0.8824	0.051906	32994	7.637753
Ball-Kmeans	CreditRisk	15	1.3134	0.082088	39772	6.336116
Kmeans	CreditRisk	20	4.8293	0.254174	380000	0
Kmeans-d	CreditRisk	20	0.9052	0.064657	37128	10.234863
Ball-Kmeans	CreditRisk	20	1.5167	0.079826	49552	7.668712
Kmeans	Census	5	48.799999	1.952	5833638	0
Kmeans-d	Census	5	41.299999	1.652	1388441	4.201574
Ball-Kmeans	Census	5	140.699997	5.628	4716650	1.236818
Kmeans	Census	8	66.099998	2.754167	9008222	0
Kmeans-d	Census	8	52.700001	2.195833	1496347	6.020143
Ball-Kmeans	Census	8	128.300003	5.345833	5284831	1.704543
Kmeans	Census	12	143.600006	3.682051	21435228	0
Kmeans-d	Census	12	109	2.794872	2345608	9.138453
Ball-Kmeans	Census	12	180.800003	4.409756	9845485	2.177163
Kmeans	Census	15	120.699997	4.47037	18654076	0
Kmeans-d	Census	15	91.300003	3.381482	1935190	9.639402
Ball-Kmeans	Census	15	126.800003	4.696296	7449954	2.503918
Kmeans	Census	20	199.199997	5.691429	31836288	0
Kmeans-d	Census	20	144.100006	4.117143	2512365	12.671841
Ball-Kmeans	Census	20	160.899994	4.597143	10266363	3.101029
Kmeans	Crop	5	355.799988	17.789999	33554218	0
Kmeans-d	Crop	5	286.5	14.325	8183553	4.100202
Ball-Kmeans	Crop	5	2230.300049	111.514999	28148246	1.192054
Kmeans	Crop	8	619.099976	23.811537	66787340	0
Kmeans-d	Crop	8	492.600006	18.946154	10828380	6.167805
Ball-Kmeans	Crop	8	2093.100098	80.503853	41694028	1.601844
Kmeans	Crop	12	1086.900024	32.936363	127152832	0
Kmeans-d	Crop	12	875.5	26.530304	14399448	8.830397
Ball-Kmeans	Crop	12	2262.899902	68.572723	61506744	2.067299
Kmeans	Crop	15	1458.099976	40.502777	176280048	0
Kmeans-d	Crop	15	1182.599976	32.849998	16528090	10.665483
Ball-Kmeans	Crop	15	2101.5	56.797298	76177720	2.314063
Kmeans	Crop	20	2951.399902	51.778946	367330368	0

Kmeans-d	Crop	20	2424	42.526318	24805820	14.808233
Ball-Kmeans	Crop	20	2787.100098	48.053452	136842400	2.684331
Kmeans	Twitter	5	13386.29981	157.485886	249631008	0
Kmeans-d	Twitter	5	5190	61.058823	53728632	4.646145
Ball-Kmeans	Twitter	5	40887.80078	492.624115	68305232	3.65464
Kmeans	Twitter	8	36062.60156	238.82518	708765376	0
Kmeans-d	Twitter	8	12561.09961	83.186089	95132880	7.450267
Ball-Kmeans	Twitter	8	89997.39844	596.009277	152914048	4.635057
Kmeans	Twitter	12	124328.1016	343.447784	2535037696	0
Kmeans-d	Twitter	12	41141.30078	113.650002	220661376	11.488361
Ball-Kmeans	Twitter	12	240938.9063	671.139038	423068032	5.992033
Kmeans	Twitter	15	167048.9063	423.981995	3450506752	0
Kmeans-d	Twitter	15	57582.19922	146.14772	241474256	14.289336
Ball-Kmeans	Twitter	15	282325	720.216858	525011616	6.572248
Kmeans	Twitter	20	220736.2031	558.825806	4618173440	0
Kmeans-d	Twitter	20	83566.20313	212.636642	244296528	18.903967
Ball-Kmeans	Twitter	20	304495	778.759583	632416960	7.302419
Kmeans	Birch	5	76.400002	3.321739	11550000	0
Kmeans-d	Birch	5	46.400002	2.017391	2750928	4.198583
Ball-Kmeans	Birch	5	251.5	10.934783	4784961	2.413813
Kmeans	Birch	8	119	4.576923	21120000	0
Kmeans-d	Birch	8	60.5	2.326923	3386407	6.236699
Ball-Kmeans	Birch	8	187.300003	7.203846	5079951	4.15752
Kmeans	Birch	12	225.5	6.094594	44760000	0
Kmeans-d	Birch	12	90.699997	2.451351	4883625	9.165323
Ball-Kmeans	Birch	12	194.100006	5.245946	6925377	6.463186
Kmeans	Birch	15	275.100006	7.435135	56849996	0
Kmeans-d	Birch	15	99	2.675676	5249538	10.829524
Ball-Kmeans	Birch	15	186.899994	5.051351	7158197	7.941944
Kmeans	Birch	20	414.600006	9.422728	88400000	0
Kmeans-d	Birch	20	121.5	2.761364	6379389	13.857126
Ball-Kmeans	Birch	20	178.199997	4.05	7517125	11.759815

Table 5: Experimental results of **random seeding** on real-world data sets: average of 10 trials are shown. The reported runtime is in milliseconds. **RT/Iter** indicates the runtime per iteration.

Algorithm	**Data**	**Clusters**	**RT**	**RT/Iter**	**DC**	**DC(S)**
Kmeans	Breastcancer	5	1.287	0.6435	5974	0
Kmeans-d	Breastcancer	5	0.7723	0.38615	3514	1.700057
Ball-Kmeans	Breastcancer	5	0.7468	0.3734	3726	1.603328
Kmeans	Breastcancer	8	3.3793	1.68965	9559	0
Kmeans-d	Breastcancer	8	1.7899	0.89495	5225	1.829474
Ball-Kmeans	Breastcancer	8	1.7314	0.8657	5499	1.738316
Kmeans	Breastcancer	12	4.6266	2.3133	15021	0
Kmeans-d	Breastcancer	12	2.3801	1.19005	7619	1.971519
Ball-Kmeans	Breastcancer	12	2.3378	1.1689	8034	1.869679
Kmeans	Breastcancer	15	2.2615	1.13075	23044	0
Kmeans-d	Breastcancer	15	1.1389	0.56945	9716	2.371758
Ball-Kmeans	Breastcancer	15	1.0425	0.52125	10420	2.211516

Kmeans	Breastcancer	20	2.6915	0.897167	38692	0
Kmeans-d	Breastcancer	20	1.473	0.491	13240	2.922357
Ball-Kmeans	Breastcancer	20	1.0477	0.349233	14599	2.650319
Kmeans	CreditRisk	5	2.5952	0.51904	25000	0
Kmeans-d	CreditRisk	5	1.0302	0.25755	8715	2.868617
Ball-Kmeans	CreditRisk	5	1.3134	0.26268	9800	2.55102
Kmeans	CreditRisk	8	2.5845	0.8615	28800	0
Kmeans-d	CreditRisk	8	1.0549	0.351633	10772	2.673598
Ball-Kmeans	CreditRisk	8	1.0523	0.350767	11198	2.571888
Kmeans	CreditRisk	12	2.976	0.5952	66000	0
Kmeans-d	CreditRisk	12	1.4516	0.29032	16112	4.096326
Ball-Kmeans	CreditRisk	12	1.397	0.2794	17461	3.779852
Kmeans	CreditRisk	15	3.1062	0.77655	73500	0
Kmeans-d	CreditRisk	15	1.3839	0.345975	18924	3.883957
Ball-Kmeans	CreditRisk	15	1.2269	0.306725	20202	3.638254
Kmeans	CreditRisk	20	3.339	0.83475	86000	0
Kmeans-d	CreditRisk	20	1.4748	0.3687	24170	3.55813
Ball-Kmeans	CreditRisk	20	1.0545	0.263625	24901	3.453676
Kmeans	Census	5	40.900002	1.947619	4770921	0
Kmeans-d	Census	5	34.200001	1.628572	1156349	4.125849
Ball-Kmeans	Census	5	117.699997	5.604762	3401185	1.402723
Kmeans	Census	8	68.300003	2.732	9297643	0
Kmeans-d	Census	8	53.200001	2.128	1498130	6.206166
Ball-Kmeans	Census	8	110.400002	4.416	5074189	1.832341
Kmeans	Census	12	99.400002	3.681482	14706195	0
Kmeans-d	Census	12	71.5	2.648148	1745856	8.423487
Ball-Kmeans	Census	12	115.400002	4.274074	6374667	2.306975
Kmeans	Census	15	129.300003	4.458621	19671570	0
Kmeans-d	Census	15	91.400002	3.151724	1969797	9.986598
Ball-Kmeans	Census	15	124.599998	4.296552	7557563	2.602899
Kmeans	Census	20	223.300003	5.725641	35634936	0
Kmeans-d	Census	20	157.399994	4.035897	2673354	13.329674
Ball-Kmeans	Census	20	178.899994	4.363414	11404094	3.124749
Kmeans	Crop	5	296.700012	17.452942	27774544	0
Kmeans-d	Crop	5	226.699997	13.335294	6926706	4.009777
Ball-Kmeans	Crop	5	1157	68.058823	20509556	1.354225
Kmeans	Crop	8	414.299988	24.370588	44182396	0
Kmeans-d	Crop	8	325.100006	19.123529	7869399	5.614456
Ball-Kmeans	Crop	8	1108	65.176468	26101188	1.692735
Kmeans	Crop	12	1013.900024	33.796669	117134720	0
Kmeans-d	Crop	12	816.900024	27.230001	13473291	8.693846
Ball-Kmeans	Crop	12	1688.099976	56.27	55601108	2.106698
Kmeans	Crop	15	1405.599976	40.16	169055456	0
Kmeans-d	Crop	15	1155.5	33.014286	15944056	10.60304
Ball-Kmeans	Crop	15	1702.800049	50.082355	70772104	2.38873
Kmeans	Crop	20	2671.199951	52.376469	331368000	0
Kmeans-d	Crop	20	2226.399902	43.6549	22904080	14.467641
Ball-Kmeans	Crop	20	2290.600098	44.913727	119144432	2.781229
Kmeans	Twitter	5	9573.200195	159.553329	176724752	0
Kmeans-d	Twitter	5	3141.600098	52.360001	37719880	4.685189
Ball-Kmeans	Twitter	5	10587.79981	179.454239	43043568	4.105718
Kmeans	Twitter	8	16304.2002	239.767654	320087616	0
Kmeans-d	Twitter	8	4275.899902	62.880882	44163036	7.247863
Ball-Kmeans	Twitter	8	15475.09961	227.574997	57027768	5.612838

Kmeans	Twitter	12	52064.19922	344.796021	1058248832	0
Kmeans-d	Twitter	12	11319.7002	75.464668	94552144	11.192225
Ball-Kmeans	Twitter	12	40057.89844	267.052643	140855008	7.513037
Kmeans	Twitter	15	67950	427.35849	1398925056	0
Kmeans-d	Twitter	15	14093.09961	88.635849	101578952	13.7718
Ball-Kmeans	Twitter	15	47725.60156	300.161011	167272784	8.363136
Kmeans	Twitter	20	95092.60156	562.678101	1980717056	0
Kmeans-d	Twitter	20	18007.69922	106.554436	110250664	17.96558
Ball-Kmeans	Twitter	20	52723.5	311.973358	197060736	10.051303
Kmeans	Birch	5	90.900002	3.366667	13950000	0
Kmeans-d	Birch	5	54	2	3214585	4.339596
Ball-Kmeans	Birch	5	207.600006	7.688889	5565331	2.506589
Kmeans	Birch	8	108.800003	4.533333	19280000	0
Kmeans-d	Birch	8	50.400002	2.1	3132497	6.154834
Ball-Kmeans	Birch	8	128.399994	5.35	4518277	4.267113
Kmeans	Birch	12	222.100006	6.169445	44040000	0
Kmeans-d	Birch	12	76.699997	2.130555	4793662	9.187131
Ball-Kmeans	Birch	12	143.699997	3.991667	6110797	7.206916
Kmeans	Birch	15	325.399994	7.395454	67049996	0
Kmeans-d	Birch	15	96	2.181818	5902405	11.359776
Ball-Kmeans	Birch	15	149.399994	3.395454	6968780	9.621483
Kmeans	Birch	20	308.399994	9.345454	66200000	0
Kmeans-d	Birch	20	76	2.30303	5236634	12.641708
Ball-Kmeans	Birch	20	117	3.545455	6108654	10.837085

Table 6: Experimental results of k**++ seeding** on real-world data sets: average of 10 trials are shown. The reported runtime is in milliseconds. **RT/Iter** indicates the runtime per iteration.

C.0.2. Doubling Experiments

The results of doubling the data proportion and clusters are reported in Table 7 and 8, respectively. Additionally, for doubling clusters experiment, the 5 point summary statistics are shown in Fig. 5.

Algorithm	**Data**	**Clusters**	**Proportion(%)**	**DC**
Kmeans	Census	5	0.2	1279867
Kmeans-d	Census	5	0.2	301041
Ball-Kmeans	Census	5	0.2	1025282
Kmeans	Census	5	0.4	2758572
Kmeans-d	Census	5	0.4	641974
Ball-Kmeans	Census	5	0.4	2178464
Kmeans	Census	5	0.8	5661857
Kmeans-d	Census	5	0.8	1310568
Ball-Kmeans	Census	5	0.8	4463415
Kmeans	Census	5	1	6579801
Kmeans-d	Census	5	1	1539259
Ball-Kmeans	Census	5	1	5249293
Kmeans	Crop	5	0.2	6165024
Kmeans-d	Crop	5	0.2	1535281
Ball-Kmeans	Crop	5	0.2	4904793
Kmeans	Crop	5	0.4	12073172
Kmeans-d	Crop	5	0.4	3000988
Ball-Kmeans	Crop	5	0.4	9163316

Kmeans	Crop	5	0.8	25173750
Kmeans-d	Crop	5	0.8	6196285
Ball-Kmeans	Crop	5	0.8	20039320
Kmeans	Crop	5	1	22637056
Kmeans-d	Crop	5	1	5986605
Ball-Kmeans	Crop	5	1	18869396
Kmeans	Twitter	5	0.2	34586728
Kmeans-d	Twitter	5	0.2	7631821
Ball-Kmeans	Twitter	5	0.2	10173062
Kmeans	Twitter	5	0.4	79322000
Kmeans-d	Twitter	5	0.4	17382364
Ball-Kmeans	Twitter	5	0.4	23151786
Kmeans	Twitter	5	0.8	174275104
Kmeans-d	Twitter	5	0.8	37635800
Ball-Kmeans	Twitter	5	0.8	48953868
Kmeans	Twitter	5	1	243798496
Kmeans-d	Twitter	5	1	52370316
Ball-Kmeans	Twitter	5	1	66806380
Kmeans	Birch	5	0.2	2770000
Kmeans-d	Birch	5	0.2	643677
Ball-Kmeans	Birch	5	0.2	1140333
Kmeans	Birch	5	0.4	4560000
Kmeans-d	Birch	5	0.4	1087669
Ball-Kmeans	Birch	5	0.4	1834783
Kmeans	Birch	5	0.8	10960000
Kmeans-d	Birch	5	0.8	2556481
Ball-Kmeans	Birch	5	0.8	4508577
Kmeans	Birch	5	1	13950000
Kmeans-d	Birch	5	1	3226912
Ball-Kmeans	Birch	5	1	5656765

Table 7: Results of **doubling proportion** experiments. The **Proportion** column indicates the percentage of data used and **DC** indicates the actual number of distance computations.

Algorithm	**Data**	**Clusters**	**DC**
Kmeans	Census	3	3866480
Kmeans-d	Census	3	1407186
Ball-Kmeans	Census	3	4248497
Kmeans	Census	6	6457701
Kmeans-d	Census	6	1346151
Ball-Kmeans	Census	6	4509229
Kmeans	Census	12	17365248
Kmeans-d	Census	12	1998822
Ball-Kmeans	Census	12	7394168
Kmeans	Census	24	47537368
Kmeans-d	Census	24	3093131
Ball-Kmeans	Census	24	12934750
Kmeans	Crop	3	7224593
Kmeans-d	Crop	3	3170468
Ball-Kmeans	Crop	3	8266640
Kmeans	Crop	6	32558828

Kmeans-d	Crop	6	7180463
Ball-Kmeans	Crop	6	23506770
Kmeans	Crop	12	142565280
Kmeans-d	Crop	12	15648925
Ball-Kmeans	Crop	12	68095344
Kmeans	Crop	24	505528832
Kmeans-d	Crop	24	28524432
Ball-Kmeans	Crop	24	164678368
Kmeans	Twitter	3	58966572
Kmeans-d	Twitter	3	21270556
Ball-Kmeans	Twitter	3	23812666
Kmeans	Twitter	6	326853312
Kmeans-d	Twitter	6	58969016
Ball-Kmeans	Twitter	6	83568736
Kmeans	Twitter	12	2560234240
Kmeans-d	Twitter	12	222402208
Ball-Kmeans	Twitter	12	419586240
Kmeans	Twitter	24	8416997376
Kmeans-d	Twitter	24	367948800
Ball-Kmeans	Twitter	24	989714944
Kmeans	Birch	3	5520000
Kmeans-d	Birch	3	2080539
Ball-Kmeans	Birch	3	3871890
Kmeans	Birch	6	14040000
Kmeans-d	Birch	6	2885427
Ball-Kmeans	Birch	6	4861603
Kmeans	Birch	12	43200000
Kmeans-d	Birch	12	4750219
Ball-Kmeans	Birch	12	7053666
Kmeans	Birch	24	133920000
Kmeans-d	Birch	24	7956828
Ball-Kmeans	Birch	24	8771822

Table 8: Results of **doubling clusters** experiments. The **Clusters** column indicates the of number of clusters and **DC** indicates the actual number of distance computations.

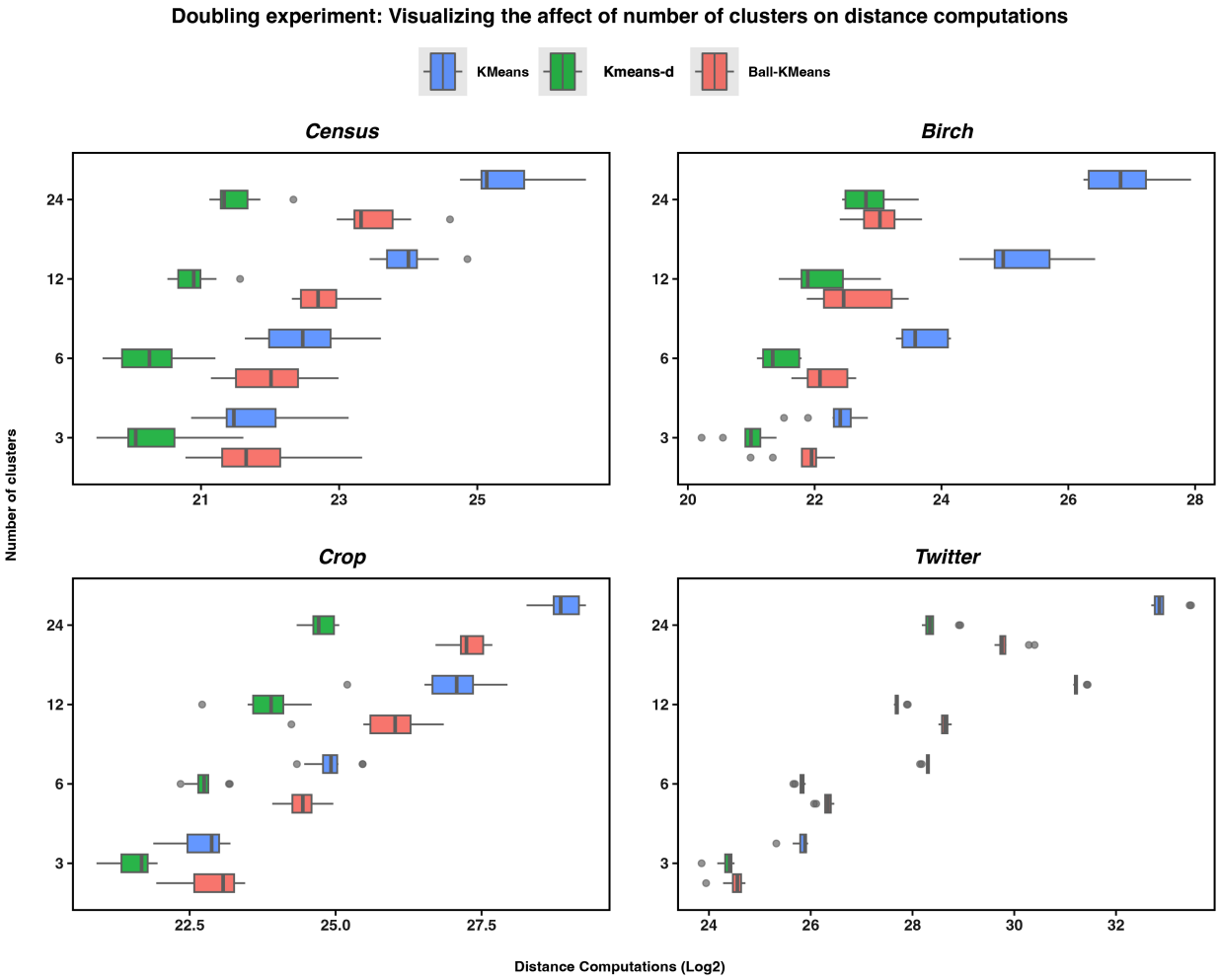

Figure 5. Doubling experiments: Comparison of DC as a function of doubling the number of clusters.

C.0.3. Ablation Experiments

Results of experiments with and without vectorization are reported in Table 9 and 10, respectively.

Algorithm	Data	Clusters	RT	RT/Iter	DC	DC(S)
Kmeans	Census	30	50.400002	10.08	7868628	0
Kmeans-d	Census	30	41.599998	8.32	1624226	4.84454
Ball-Kmeans	Census	30	40.700001	8.14	3191343	2.465616
Kmeans	Census	50	71.400002	14.280001	11305500	0
Kmeans-d	Census	50	58.5	11.7	2485040	4.549424
Ball-Kmeans	Census	50	42	8.4	4020819	2.811741
Kmeans	Census	60	85	17	13837933	0
Kmeans-d	Census	60	71.400002	14.280001	2952545	4.686781
Ball-Kmeans	Census	60	46.299999	9.26	4703294	2.942179
Kmeans	Census	80	113.099998	22.619999	18450576	0
Kmeans-d	Census	80	93.599998	18.719999	3867476	4.770702
Ball-Kmeans	Census	80	53.200001	10.64	5722497	3.224218
Kmeans	Twitter	30	84029.29688	840.292969	1767247488	0
Kmeans-d	Twitter	30	56579.10156	565.791016	79366000	22.267059
Ball-Kmeans	Twitter	30	98689.29688	986.892944	321812544	5.491543
Kmeans	Twitter	50	139168.9063	1391.689087	2945412608	0
Kmeans-d	Twitter	50	106728.8984	1067.28894	91476304	32.198639
Ball-Kmeans	Twitter	50	105830.2969	1058.302979	464467104	6.341488
Kmeans	Twitter	60	166479.5938	1664.795898	3534494976	0
Kmeans-d	Twitter	60	130971.6016	1309.716064	97433752	36.275879
Ball-Kmeans	Twitter	60	109170.5	1091.704956	536030304	6.593834
Kmeans	Twitter	80	223673.7031	2236.737061	4712659968	0
Kmeans-d	Twitter	80	189776.9063	1897.769043	109485784	43.043579
Ball-Kmeans	Twitter	80	114305.6016	1143.05603	695732800	6.773664
Kmeans	Birch	30	498.100006	15.09394	99000000	0
Kmeans-d	Birch	30	111.5	3.378788	6265351	15.801189
Ball-Kmeans	Birch	30	142.800003	4.327273	7218448	13.714859
Kmeans	Birch	50	1893.800049	23.672501	401500000	0
Kmeans-d	Birch	50	269.700012	3.37125	13114883	30.614075
Ball-Kmeans	Birch	50	204.800003	2.56	11611740	34.577076
Kmeans	Birch	60	2155.600098	27.635899	471600032	0
Kmeans-d	Birch	60	276.700012	3.502532	14114643	33.412113
Ball-Kmeans	Birch	60	203.699997	2.611538	12881577	36.610428
Kmeans	Birch	80	3246.199951	35.284782	744000000	0
Kmeans-d	Birch	80	365.899994	3.977174	17571762	42.34066
Ball-Kmeans	Birch	80	266.799988	2.9	17451788	42.631737

Table 9: Ablation study **with** vectorization: average of 10 trials are shown. **RT/Iter** indicates the runtime per iteration.

Algorithm	Data	Clusters	RT	RT/Iter	DC	DC(S)
Kmeans	Census	30	532.299988	106.459999	7868628	0
Kmeans-d	Census	30	443.600006	88.720001	1624226	4.84454
Ball-Kmeans	Census	30	2136.800049	427.360016	3191343	2.465616
Kmeans	Census	50	742.5	148.5	11305500	0
Kmeans-d	Census	50	631.400024	126.280006	2485040	4.549424
Ball-Kmeans	Census	50	1562.300049	312.460022	4020819	2.811741

Kmeans	Census	60	909.900024	181.980011	13837933	0
Kmeans-d	Census	60	788.799988	157.759995	2952545	4.686781
Ball-Kmeans	Census	60	1637.599976	327.519989	4703294	2.942179
Kmeans	Census	80	1177.400024	235.480011	18450576	0
Kmeans-d	Census	80	1004.099976	200.819992	3867476	4.770702
Ball-Kmeans	Census	80	1622.5	324.5	5722497	3.224218
Kmeans	Twitter	30	896613.3125	8966.132812	1767247488	0
Kmeans-d	Twitter	30	757723.6875	7577.236816	79366000	22.267059
Ball-Kmeans	Twitter	30	1786030.75	21780.86328	283065696	6.243241
Kmeans	Twitter	50	1451841.75	14518.41797	2945412608	0
Kmeans-d	Twitter	50	1420208.125	14202.08106	91476304	32.198639
Ball-Kmeans	Twitter	50	1806122.625	22297.81055	407760000	7.223398
Kmeans	Twitter	60	1744898	17448.98047	3534494976	0
Kmeans-d	Twitter	60	1740760	17583.43359	97374136	36.298088
Ball-Kmeans	Twitter	60	1807583.25	22594.79102	465726208	7.589212
Kmeans	Twitter	80	1814403.625	23261.58398	3709469952	0
Kmeans-d	Twitter	80	1809125.625	27410.99414	88936560	41.709167
Ball-Kmeans	Twitter	80	1810063.625	23205.94336	593397632	6.251238
Kmeans	Birch	30	4319.700195	130.900009	99000000	0
Kmeans-d	Birch	30	711.700012	21.566668	6265351	15.801189
Ball-Kmeans	Birch	30	7774.799805	235.599991	7218448	13.714859
Kmeans	Birch	50	16619.19922	207.73999	401500000	0
Kmeans-d	Birch	50	1787.5	22.34375	13114883	30.614075
Ball-Kmeans	Birch	50	8113.799805	101.422501	11611740	34.577076
Kmeans	Birch	60	19244.90039	246.729492	471600032	0
Kmeans-d	Birch	60	1828.400024	23.144304	14114643	33.412113
Ball-Kmeans	Birch	60	7855.299805	100.708969	12881577	36.610428
Kmeans	Birch	80	29984.90039	325.922821	744000000	0
Kmeans-d	Birch	80	2428.800049	26.4	17571762	42.34066
Ball-Kmeans	Birch	80	9069.5	98.58152	17451788	42.631737

Table 10: Ablation study **without** vectorization: average of 10 trials are shown. **RT/Iter** indicates the runtime per iteration.

[1] Venables, W. N., & Ripley, B. D. (2002). Modern Applied Statistics with S (Fourth ed.). New York: Springer. Retrieved from https://www.stats.ox.ac.uk/pub/MASS4/

[2] Qiu, W., & Joe, H. (2020). clusterGeneration: Random Cluster Generation (with Specified Degree of Separation) (R package version 1.3.7). Retrieved from https://CRAN.R-project.org/package=clusterGeneration