k^2-means for fast and accurate large scale clustering

Eirikur Agustsson[1], Radu Timofte[1,2], and Luc Van Gool[1,3]

[1] Computer Vision Lab, D-ITET, ETH Zurich, Switzerland
[2] Merantix GmbH, Germany
[3] KU Leuven, Belgium

Abstract. We propose k^2-means, a new clustering method which efficiently copes with large numbers of clusters and achieves low energy solutions. k^2-means builds upon the standard k-means (Lloyd's algorithm) and combines a new strategy to accelerate the convergence with a new low time complexity divisive initialization. The accelerated convergence is achieved through only looking at k_n nearest clusters and using triangle inequality bounds in the assignment step while the divisive initialization employs an optimal 2-clustering along a direction. The worst-case time complexity per iteration of our k^2-means is $O(nk_nd + k^2d)$, where d is the dimension of the n data points and k is the number of clusters and usually $n \gg k \gg k_n$. Compared to k-means' $O(nkd)$ complexity, our k^2-means complexity is significantly lower, at the expense of slightly increasing the memory complexity by $O(nk_n + k^2)$. In our extensive experiments k^2-means is order(s) of magnitude faster than standard methods in computing accurate clusterings on several standard datasets and settings with hundreds of clusters and high dimensional data. Moreover, the proposed divisive initialization generally leads to clustering energies comparable to those achieved with the standard k-means++ initialization, while being significantly faster.

1 Introduction

The k-means algorithm in its standard form (Lloyd's algorithm) employs two steps to cluster n data points of d dimensions and k initial cluster centers [19]. The *expectation* or *assignment step* assigns each point to its nearest cluster while the *maximization* or *update step* updates the k cluster centers with the mean of the points belonging to each cluster. The k-means algorithm repeats the two steps until convergence, that is the assignments no longer change in an iteration i.

k-means is one of the most widely used clustering algorithms, being included in a list of top 10 data mining algorithms [27]. Its simplicity and general applicability vouch for its broad adoption. Unfortunately, its $O(ndki)$ time complexity depends on the product between number of points n, number of dimensions d, number of clusters k, and number of iterations i. Thus, for large such values even a single iteration of the algorithm is very slow.

The simplest way to handle larger datasets is parallelization [29, 28], however this requires more computation power as well. Another way is to process the data online in batches as done by the MiniBatch algorithm of Sculley [23], a variant of the Lloyd algorithm that trades off quality (i.e. the converged energy) for speed.

Table 1. Notations

n	number of data points to cluster
k	number of clusters
k_n	number of nearest clusters
d	number of dimensions of the data points
X	the data $(x_i)_{i=1}^n, x_i \in \mathbb{R}^d$
C	cluster centers $C = (c_j)_{j=1}^k, c_j \in \mathbb{R}^d$
a	cluster assignments $\{1, \cdots, n\} \to \{1, \cdots, k\}$
$a(x_i)$	cluster assignment of x_i, i.e. $a(i)$
$a(X')$	cluster assignment of some set of points X'
X_j	points assigned to cluster j, $(x_i \in X \vert a(i) = j)$
$\mu(X_j)$	the mean of X_j: $\frac{1}{\vert X_j \vert} \sum_{x \in X_j} x$
$\|x\|$	l_2 norm of $x \in \mathbb{R}^d$
$\phi(X_j)$	energy of X_j: $\sum_{x \in X_j} \|x - \mu(X_j)\|^2$
$\mathcal{N}_{k_n}(c_l)$	k_n nearest neighbours of c_l in C (including c_l)

To improve both the speed and the quality of the clustering results, Arthur and Vassilvitskii [1] proposed the k-means++ initialization method. The initialization typically results in a higher quality clustering and fewer iterations for k-means, than when using the default random initialization. Furthermore, the expected value of the clustering energy is within a $8(\ln k + 2)$ factor of the optimal solution. However, the time complexity of the method is $O(ndk)$, i.e. the same as a single iteration of the Lloyd algorithm - which can be too expensive in a large scale setting. Since k-means++ is sequential in nature, Bahman *et al.* [2] introduced a parallel version k-means$||$ of k-means++, but did not reduce the time complexity of the method.

Another direction is to speed up the actual k-means iterations. Elkan [8], Hamerly [11] and Drake and Hamerly [7] go in this direction and use the triangle inequality to avoid unnecessary distance computation between cluster centers and the data points. However, these methods still require a full Lloyd iteration in the beginning to then gradually reduce the computation of progressive iterations. The recent Yinyang k-means method of Ding *et al.* [6] is a similar method, that also leverages bounds to avoid redundant distance calculations. While typically performing 2-3$\times$ faster than Elkan method, it also requires a full Lloyd iteration to start with.

Philbin *et al.* [22] introduce an approximate k-means (AKM) method based on kd-trees to speed up the assignment step, reducing the complexity of each k-means iteration from $O(nkd)$ to $O(nmd)$, where $m < k$. In this case m, the distance computations performed per each iteration, controls the trade-off between a fast and an accurate (i.e. low energy) clustering. Wang *et al.* [26] use cluster closures for further $2.5\times$ speedups.

Mazzeo *et al.* [20] introduce a centroid-based method that combines divisive and agglomerative clustering, obtaining quickly high quality clusters as measured by the CH-index [5].

In this paper we propose k^2-means, a method aiming at both fast and accurate clustering. Following the observation that usually the clusters change gradually and affect only local neighborhoods, in the assignment step we only consider the k_n nearest neigh-

bours of a center as the candidates for the clusters members. Furthermore we employ the triangle inequality bounds idea as introduced by Elkan [8] to reduce the number of operations per each iteration. For initializing k^2-means, we propose a divisive initialization method, which we experimentally prove to be more efficient than k-means++.

Our k^2-means gives a significant algorithmic speedup, i.e. reducing the complexity to $O(nk_nd)$ per iteration, while still maintaining a high accuracy comparable to methods such as k-means++ for a chosen $k_n < k$. Similar to m in AKM, k_n also controls a trade-off between speed and accuracy. However, our experiments show that we can use a significantly lower k_n when aiming for a high accuracy.

The paper is structured as follows. In Table 1 we summarize the notations used in this paper. In Section 2 we introduce our proposed k^2-means method and our divisive initialization. In Section 3 we describe the experimental benchmark and discuss the results obtained, while in Section 4 we draw conclusions.

Algorithm 1 k^2-means

Given: k, data X, neighbourhood size k_n
Initialize centers C
Initialize assignments $a : \{1 \cdots n\} \to \{1, \cdots, k\}$.
while Not converged **do**
  Build k_n-NN graph of C:
  $\mathcal{N}_{k_n} : C \to \{1 \cdots k\}^{k_n}$
  for $x \in X$ **do**
    Get current center for x:
    $l \leftarrow a(x)$
    Assign x to nearest candidate center:
    $a(x) \leftarrow \arg\min_{l' \in \mathcal{N}_{k_n}(c_l)} \|x - c_{l'}\|$
  end for
  for $j \in \{1 \cdots k\}$ **do**
    $c_j \leftarrow \mu(X_j)$ {Update center}
  end for
end while
return C, a

2 Proposed k^2-means

In this section we introduce our k^2-means method and motivate the design decisions. The pseudocode of the method is given in Algorithm 1.

Given some data $X = (x_i)_{i=1}^n, x_i \in \mathbb{R}^d$, the k-means clustering objective is to find cluster centers $C = (c_j)_{j=1}^k, c_j \in \mathbb{R}^d$ and cluster assignments $a : \{1, \cdots, n\} \to \{1, \cdots, k\}$, such that the cluster energy

$$\sum_{j=1}^{k} \sum_{x \in X_j} \|x - c_j\|^2 \tag{1}$$

is minimized, where $X_j := (x_i \in X | a(i) = j)$ denotes the points assigned to a cluster j. For a data point x_i, we sometimes write $a(x_i)$ instead of $a(i)$ for the cluster assignment. Similarly, for a subset X' of the data, $a(X')$ denotes the cluster assignments of the corresponding points.

Standard Lloyd obtains an approximate solution by repeating the following until convergence: i) In the *assignment step*, each x is assigned to the nearest center in C. ii) For the *update step*, each center is recomputed as the mean of its members.

The assignment step requires $O(nk)$ distance computations, i.e. $O(nkd)$ operations, and dominates the time complexity of each iteration. The update step requires only $O(nd)$ operations for mean computations.

To speed up the assignment step, an approximate nearest neighbour method can be used, such as kd-trees [22, 21] or locality sensitive hashing [13]. However, these methods ignore the fact that the cluster centers *are moving* across iterations and often this movement is *slow*, affecting a *small neighborhood* of points. With this observation, we obtain a very simple fast nearest neighbour scheme:

Suppose at iteration i, a data point x was assigned to a nearby center, $l = a(x)$. After updating the centers, we still expect c_l to be close to x. Therefore, the centers nearby c_l are likely candidates for the nearest center of x in iteration $i + 1$. To speed up the assignment step, we thus only consider the k_n nearest neighbours of c_l, $\mathcal{N}_{k_n}(c_l)$, as candidate centers for the points $x \in X_l$. Since for each point we only consider k_n centers in the assignment step (in line 11 of Algorithm 1), the complexity is reduced to $O(nk_nd)$. In practice, we can set $k_n \ll k$.

We also use inequalities as in [8] to avoid redundant distance computations in the assignment step (in line 11 of Algorithm 1). We use the exact same triangle inequalities as described in the Elkan paper [8], but only maintain the nk_n lower bounds, for the neighbourhood of each point, instead of nk for the Elkan method. It is easy to see that this modification is valid as an exact speed up of the assignment step within then neighbourhood. When a point is assigned to a new cluster, we however need to update the k_n lower bounds of the point, since the neighbourhood changes in this case. We refer to the original Elkan paper [8] for a detailed discussion on triangle inequalities and bounds.

As for standard Lloyd, the total energy can only decrease in each iteration of the algorithm. In the assignment step, points are only moved to closer centers, reducing their contribution to the total energy. In the update step, the center z of a cluster S is updated as the mean of its members, $\mu(S)$. As Lemma 1 shows, this clearly reduces the energy of the cluster since the second right hand side term is positive. Thus, the total energy is monotonically decreasing as a function of iterations, which guarantees convergence.

As shown by Arthur and Vassilvitskii [1], a good initialization, such as k-means++, often leads to a higher quality clustering compared to random sampling. Since the $O(ndk)$ complexity of k-means++ would negate the benefits of the k^2-means computation savings, we propose an alternative fast initialization scheme, which also leads to high quality clustering solutions.

2.1 Greedy Divisive Initialization (GDI)

For the initialization of our k^2-means, we propose a simple hierarchical clustering method named Greedy Divisive Initialization (GDI), detailed in Algorithm 2. Similarly to other divisive clustering methods, such as [4, 24], we start with a single cluster and repeatedly split the highest energy cluster until we reach k clusters.

To efficiently split each cluster, we use Projective Split (Algorithm 3), a variant of k-means with $k = 2$, that is motivated by the following observation: Suppose we have points X' and centers (c_1, c_2) in the k-means method. Let H be the hyperplane with normal vector $c_2 - c_1$, going through $\mu(c_1, c_2)$ (see e.g. the top left corner of Figure 1). When we perform the standard k-means assignment step, we greedily assign each point to its closest centroid to get a solution with a lower energy, thus assigning the points on one side of H to c_1, and the other side of H to c_2.

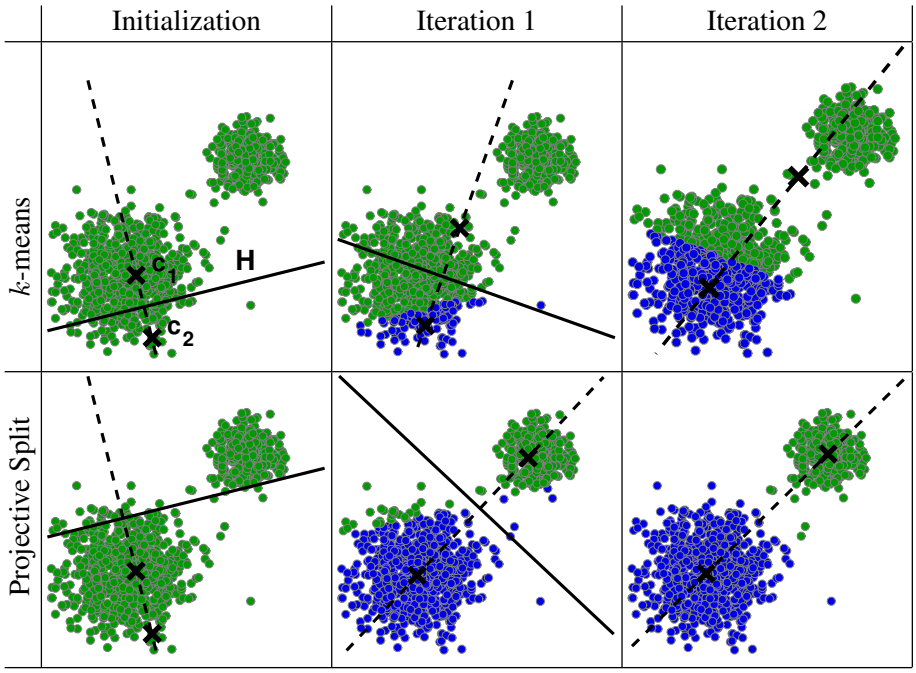

Fig. 1. Example of two iterations of Projective Split and standard k-means with $k = 2$ using the same initialization. The dashed line shows the direction defined by two centers $(c_2 - c_1)$. The solid line shows where the algorithms split the data in each iteration. The splitting line of k-means always goes through the midpoint of the two centers, while Projective Split picks the minimal energy split along the dashed line. Even though the initial centers start in the same cluster, Projective Split can almost separate the clusters in a single iteration.

Although this is the best assignment choice for the current centers c_1 and c_2, this may not be a good split of the data. Therefore, we depart from the standard assignment

step and consider instead *all* hyperplanes along the direction $c_2 - c_1$ (i.e. with normal vector $c_2 - c_1$). We project X' onto $c_2 - c_1$ and "scan" a hyperplane through the data to find the split that gives the lowest energy (lines 4-8 in Algorithm 3). To efficiently recompute the energy of the cluster splits as the hyperplane is scanned, we use the following Lemma:

Lemma 1. *[14][Lemma 2.1] Let S be a set of points with mean $\mu(S)$. Then for any point $z \in \mathbb{R}^d$*

$$\sum_{x \in S} \|x - z\|^2 = \sum_{x \in S} \|x - \mu(S)\|^2 + |S| \|z - \mu(S)\|^2 \tag{2}$$

We can now compute

$$\begin{aligned} \phi(S \cup \{y\}) &= \sum_{x \in S \cup \{y\}} \|x - \mu(S \cup \{y\})\|^2 && (3) \\ &= \sum_{x \in S} \|x - \mu(S \cup \{y\})\|^2 + \|y - \mu(S \cup \{y\})\|^2 && (4) \\ &= \phi(S) + |S| \|\mu(S \cup \{y\}) - \mu(S)\|^2 + \|y - \mu(S \cup \{y\})\|^2, && (5) \end{aligned}$$

where we used Lemma 1 in (4). Equipped with (5) we can efficiently update energy terms in line 8 in Algorithm 3 as we scan the hyperplane through the data X_j (after sorting it along $c_a - c_b$ in line 5-6), using in total only $O(|X_j|)$ distance computations and mean updates. Note that $\mu(S \cup \{y\})$ is easily computed with an add operation as $(|S|\mu(S) + y)/(|S| + 1)$.

Algorithm 2 Greedy Divisive Initialization (GDI)

1: **Given**: k, data X
2: Assign all points to one cluster
3: $C = \{\mu(X)\}$, $a(X) = 1$
4: **while** $|C| < k$ **do**
5: Pick highest energy cluster:
6: $j \leftarrow \arg\max_l \phi(X_l)$
7: Split the cluster:
8: $X_a, c_a, X_b, c_b \leftarrow$ ProjectiveSplit(X_j)
9: $c_j \leftarrow c_a$
10: $c_{|C|+1} \leftarrow c_b$
11: $a(X_b) \leftarrow |C| + 1$
12: $C \leftarrow C \cup \{c_{|C|+1}\}$
13: **end while**
14: **return** C, a

Algorithm 3 Projective Split

Given:data $X_j = (x_i)_{i=1}^{n_j}$
Pick two random samples c_a, c_b from X_j
while Not Converged **do**
    Sort X_j along $c_a - c_b$:
    $P_j \leftarrow (x_i \cdot (c_a - c_b) | x_i \in X_j)$
    $X_j \leftarrow X_j$ sorted by P_j
    Find minimum-energy split:
    $l_{min} = \arg\min_l \phi((\tilde{x}_i)_{i=1}^{l}) + \phi((\tilde{x}_i)_{i=l+1}^{n_j})$
    $X_a \leftarrow (\tilde{x}_i)_{i=1}^{l_{min}}$
    $X_b \leftarrow (\tilde{x}_i)_{i=l_{min}+1}^{n_j}$
    $c_a, c_b \leftarrow \mu(X_a), \mu(X_b)$
end while
return X_a, c_a, X_b, c_b

Compared to standard k-means with $k = 2$, our Projective Split takes the optimal split along the direction $c_2 - c_1$ but greedily considers only this direction. In Figure 1 we show how this can lead to a faster convergence.

2.2 Time Complexity

Table 2 shows the time and memory complexity of Lloyd, Elkan, MiniBatch, AKM, and our k^2-means.

Method	Time complexity	Memory complexity
Lloyd	$O(nkd)$	$O((n+k)d)$
Elkan [8]	$O(nkd + k^2d) \sim O(nd + k^2d)$	$O((n+k)d + nk + k^2)$
MiniBatch [23]	$O(bkd)$	$O((b+k)d)$
AKM [22]	$O(nmd)$	$O((n+k)d)$
k^2-means (ours)	$O(nk_nd + k^2d) \sim O(nd + k^2d)$	$O((n+k)d + nk_n + k^2)$

Table 2. Time and memory complexity per iteration for Lloyd, Elkan, MiniBatch, AKM and our k^2-means.

The time complexity of each k^2-means iteration is dominated by two factors: building the nearest neighbour graph of C (line 6), which costs $O(k^2)$ distance computations, as well as computing distances between points and candidate centers (line 11), which initially costs nk_n distance computations. Elkan and k^2-means use the triangle inequality to avoid redundant distance calculations and empirically we observe the $O(nkd)$ and $O(nk_nd)$ terms (respectively) gradually reduce down to $O(nd)$ at convergence.

In MiniBatch k-means processes only b samples per iteration (with $b \ll n$) but needs more iterations for convergence. AKM limits the number of distance computations to m per iteration, giving a complexity of $O(nmd)$.

Table 3 shows the time and memory complexity of random, k-means++ and our GDI initialization. For the GDI, the time complexity is dominated by calls to Projective

Initialization	Time complexity	Memory complexity
random	$O(k)$	$O(k)$
k-means++ [1]	$O(nkd)$	$O(n+k)$
GDI (ours)	$O(n(\log k)(d+\log n)) \sim O(nk(d+\log n))$	$O(n+kd)$

Table 3. Time and memory complexity for initialization.

Split. If we limit Projective Split to maximum $O(1)$ iterations (2 in our experiments) then a call to ProjectiveSplit(X_j) costs $O(|X_j|)$ distance computations and vector additions, $O(|X_j|)$ inner products and $O(|X_j| \log |X_j|)$ comparisons (for the sort), giving in total $O(|X_j|(\log |X_j| + d))$ complexity. However, the resulting time complexity of GDI depends on the data.

For pathological datasets, it could happen for each call to ProjectiveSplit(X'), that the minimum split is of the form $\{y\}, X' \setminus \{y\}$, i.e. only one point y is split off. In this case, for $|X| = n$, the total complexity will be $O(n(\log n + d) + (n-1)(\log(n-1) + d) + \cdots + (n-k)(\log(n-k)+d)) = O(nk(d+\log n))$. [4]

A more reasonable case is when at each call ProjectiveSplit(X') splits each cluster into two similarly large clusters, i.e. the minimum split is of the form (X'_a, X'_b) where $|X_a| \approx |X_b|$. In this case the worst case scenario is when in each split the highest energy cluster is the largest cluster (in no. of samples), resulting a total complexity of $O(n \log k(d+\log n))$. [5] Therefore the time complexity of GDI is somewhere between $O(n \log k(d+\log n)) \sim O(n(d+\log n)k)$.

In our experiments we count vector operations for simplicity (i.e. dropping the $O(d)$ factor), as detailed in the next section. To fairly account for the $O(|X_j| \log |X_j|)$ complexity of the sorting step in ProjectiveSplit, we artificially count it as $|X_j| \log_2(|X_j|)/d$ vector operations.

3 Experiments

For a fair comparison between methods implemented in various programming languages, we use the number of vector operations as a measure of complexity, i.e. distances, inner products and additions. While the operations all share an $O(d)$ complexity, the distance computations are most expensive accounting for the constant factor. However, since the runtime of all methods is dominated by distance computations (i.e. more than 95% of the runtime), for simplicity we count all vector operations equally and refer to them as "distance computations", using the terminology from [8].

[4] A simple example of such a pathological dataset is $X = (x_i)_{i=1}^n \subset \mathbb{R}$ where $x_1 = 0$, $x_2 = 1, x_3 = \phi(x_1, x_2)$, $x_4 = \phi(x_1, x_2, x_3)$ and $x_n = \phi(x_1, \cdots, x_n)$. The size of x_n grows extremely fast though, e.g. $x_{10} \approx 1581397605569$ and x_{14} has 195 digits.

[5] If we split all clusters of approximately equal size simultaneously, we need $O(\log k)$ passes and perform $O(n(d+\log n))$ computations in each pass.

3.1 Datasets

In our experiments we use datasets with 2414-150000 samples ranging from 50 to 32256 dimensions as listed in Table 5. The datasets are diverse in content and feature representation.

To create **cnnvoc** we extract 4096-dimensional CNN features [16] for 15662 bounding boxes, each belonging to 20 object categories, from PASCAL VOC 2007 [9] dataset. **covtype** uses the first 150000 entries of the Covertype dataset [3] of cartographic features. From the **mnist** database [17] of handwritten digits we also generate **mnist50** by random projection of the raw pixels to a 50-dimensional subspace. For **tinygist10k** we use the first 10000 images with extracted gist features from the 80 million tiny images dataset [25]. **cifar** represents 50000 training images from the CIFAR [15] dataset. **usps** [12] has scans of handwritten digits (raw pixels) from envelopes. **yale** contains cropped face images from the Extended Yale B Database [10, 18].

3.2 Methods

We compare our k^2**-means** with relevant clustering methods: Lloyd (standard k-means), Elkan [8] (accelerated Lloyd), MiniBatch [23] (web-scale online clustering), and AKM [22] (efficient search structure).

Aside from our **GDI** initialization, we also use **random** initialization and k**-means++** [1] in our experiments. For k-means++ we use the provided Matlab implementation. We Matlab implement MiniBatch k-means according to Algorithm 1 in [23] and use the provided codes for Elkan and AKM. **Lloyd++** and **Elkan++** combine k-means++ initialization with Lloyd and Elkan, respectively.

We run all methods, except MiniBatch, for a maximum of 100 iterations. For MiniBatch k-means we use $b = 100$ samples per batch and $t = n/2$ iterations. For the Projective Split, Algorithm 3, we perform only 2 iterations.

3.3 Initializations

We compare k-means++, random and our GDI initialization by running 20 trials of k-means (Lloyd) clustering with $k \in \{100, 200, 500\}$ on the datasets. Table 4 reports minimum and average cluster energy as well as the average number of distance computations, relative to k-means++, averaged over 20 seeds.

Our GDI gives a (slightly) better average and minimum convergence energy than the other initializations, while its runtime complexity is an order of magnitude smaller than in the case of k-means++ initialization. Notably, the speedup of GDI over k-means++ improves as k grows, and at $k = 500$ is typically more than an order of magnitude. This makes GDI a good choice for the initialization of k^2-means.

3.4 Performance

Our goal is *fast accurate clustering*, where the cluster energy differs only slightly from Lloyd with a good initialization (such as k-means++) at convergence. Therefore, we

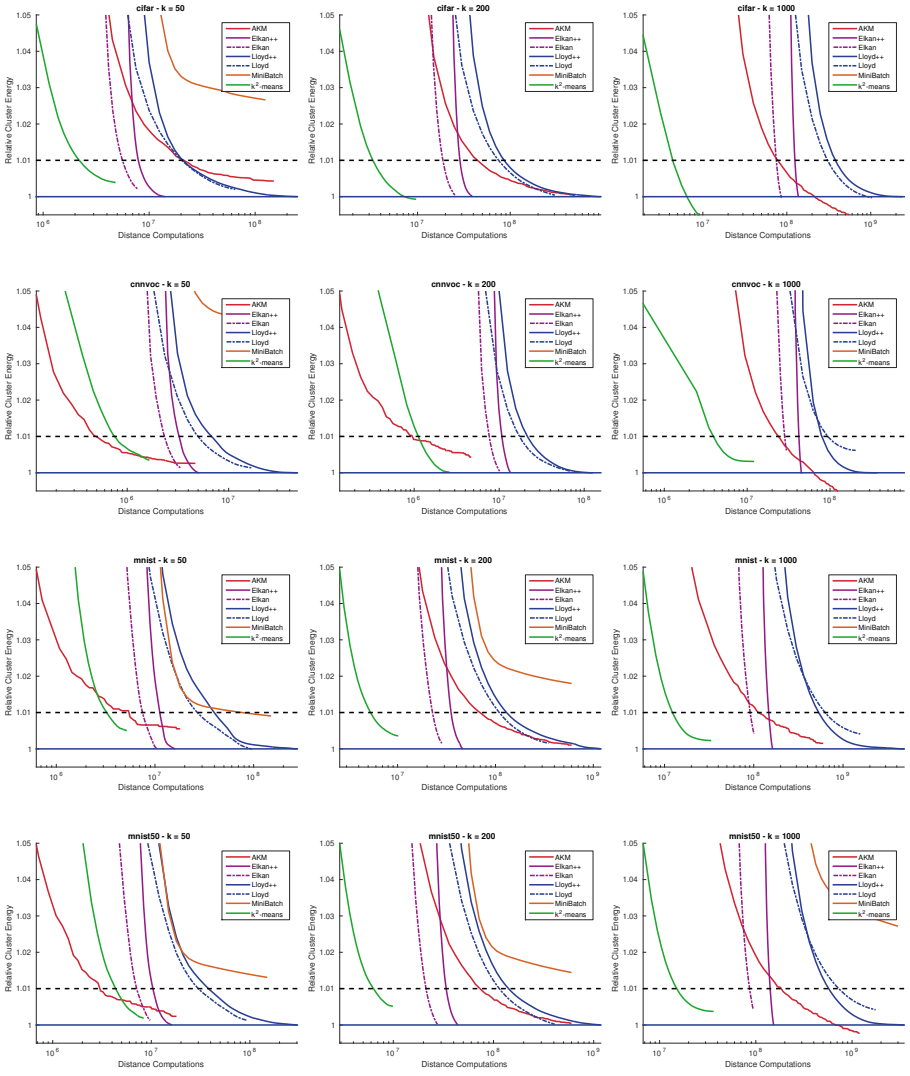

Fig. 2. Cluster Energy (relative to best Lloyd++ energy) vs distance computations on cifar, cnnvoc, mnist and mnist50 for $k \in \{50, 200, 1000\}$. For AKM and k^2-means, we use the parameter with the highest algorithmic speedup at 1% error.

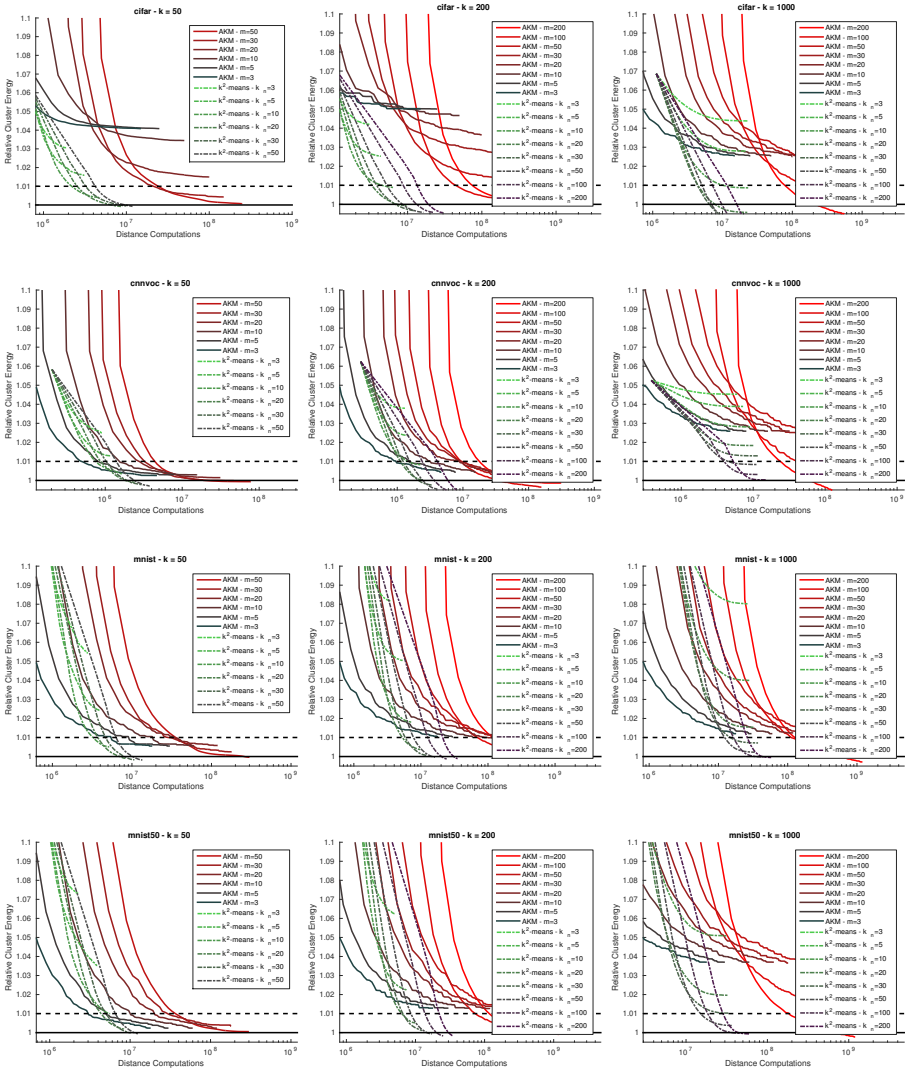

Fig. 3. Cluster Energy (relative to best Lloyd++ energy) vs distance computations on cifar, cnnvoc, mnist and mnist50 for $k \in \{50, 200, 1000\}$. For AKM and k^2-means, we use the parameter with the highest algorithmic speedup at 1% error.

		average convergence energy			*minimum convergence energy*			*average runtime complexity*	
Dataset	k	random	k-means++	**GDI**	random	k-means++	**GDI**	k-means++	**GDI**
cnnvoc	100	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	1.00	**0.16**
	200	1.00	1.00	**0.99**	1.00	1.00	**0.99**	1.00	**0.09**
	500	1.00	1.00	**0.99**	1.00	1.00	**0.99**	1.00	**0.04**
covtype	100	1.51	1.00	**0.99**	1.47	1.00	**0.99**	1.00	**0.19**
	200	1.58	1.00	**0.98**	1.38	1.00	**0.99**	1.00	**0.11**
	500	1.43	1.00	**0.99**	1.30	1.00	**0.99**	1.00	**0.05**
mnist	100	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	1.00	**0.15**
	200	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	1.00	**0.09**
	500	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	1.00	**0.04**
mnist50	100	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	1.00	**0.19**
	200	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	1.00	**0.11**
	500	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	1.00	**0.05**
tinygist10k	100	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	1.00	**0.16**
	200	1.00	1.00	**0.99**	**1.00**	**1.00**	**1.00**	1.00	**0.09**
	500	1.00	1.00	**0.99**	**1.00**	**1.00**	**1.00**	1.00	**0.04**
usps	100	1.01	1.00	**0.99**	1.01	**1.00**	**1.00**	1.00	**0.16**
	200	1.01	1.00	**0.99**	1.01	1.00	**0.99**	1.00	**0.09**
	500	1.04	**1.00**	**1.00**	1.04	**1.00**	**1.00**	1.00	**0.05**
yale	100	1.01	**1.00**	**1.00**	1.00	1.00	**0.99**	1.00	**0.16**
	200	1.02	**1.00**	**1.00**	1.02	**1.00**	**1.00**	1.00	**0.10**
	500	1.05	**1.00**	1.03	1.05	**1.00**	1.02	1.00	**0.05**
average		1.078	1.000	0.996	1.061	1.000	**0.997**	1.000	**0.103**

Table 4. Comparison of energy and runtime complexity for random, k-means++, and our GDI initialization. The results are displayed relative to k-means++, averaged over 20 seeds. Random initialization does not require distance computations. GDI is an order of magnitude faster while giving comparable energies to k-means++.

measure the runtime complexity needed to achieve a clustering energy that is within 1% of the energy obtained with Lloyd++ at convergence.

For a given budget i.e. the maximum number of iterations and parameters such as m for AKM and k_n for k^2 means, it is not known beforehand how well the algorithms approximate the targeted Lloyd++ energy. For a fair comparison we use an oracle to select the best parameters and the number of iterations for each method, i.e. the ones that give the highest speedup but still reach the reference error. In practice, one can use a rule of thumb or progressively increase k, m and the number of iterations until a desired energy has been reached.

To measure performance we run AKM, Elkan++, Elkan, Lloyd++, Lloyd, Mini-Batch, and k^2-means with $k \in \{50, 200, 1000\}$ on various datasets, with 3 different seeds and report average speedups over Lloyd++ when the energy reached is within 1% from Lloyd++ at convergence in Table 5.

Each method is stopped once it reaches the reference energy and for AKM and k^2-means, we use the parameters m and k_n from $\{3, 5, 10, 20, 30, 50, 100, 200\}$ that give the highest speedup.

Table 5 shows that for most settings, our k^2-means has the highest algorithmic speedup at 1% error. It benefits the most when both the number of clusters and the number of points are large, e.g. for $k = 200$ at least $19\times$ speedup for all datasets with $n \geq 7000$ samples. We do not reach the target energy for usps and yale with $k = 1000$, because k_n was limited to 200.

Dataset	k	AKM	Elkan++	Elkan	Lloyd++	Lloyd	k^2**-means**
cifar	50	1.0	2.6	3.7	1.0	1.0	**9.5**
$n = 50000$	200	1.9	3.0	4.6	1.0	1.1	**26.2**
$d = 3072$	1000	4.9	3.0	5.1	1.0	1.2	**86.7**
cnnvoc	50	**13.8**	2.1	2.9	1.0	1.4	9.0
$n = 15662$	200	**22.6**	2.0	2.8	1.0	1.2	19.2
$d = 4096$	1000	3.3	1.9	2.8	1.0	0.9	**20.2**
covtype	50	-	6.1	-	1.0	-	**35.1**
$n = 150000$	200	-	6.3	-	1.0	-	**78.7**
$d = 54$	1000	-	8.5	-	1.0	-	**176.6**
mnist	50	7.3	3.6	5.3	1.0	1.5	**12.3**
$n = 60000$	200	1.9	3.7	5.7	1.0	1.2	**24.6**
$d = 784$	1000	4.7	3.6	5.9	1.0	0.8	**43.4**
mnist50	50	**12.7**	3.7	5.4	1.0	1.3	8.8
$n = 60000$	200	1.9	4.2	6.7	1.0	1.2	**22.3**
$d = 50$	1000	3.1	4.1	6.6	1.0	0.8	**38.0**
tinygist10k	50	**16.2**	2.4	3.6	1.0	1.4	11.7
$n = 10000$	200	12.8	2.3	3.5	1.0	1.3	**22.3**
$d = 384$	1000	1.5	2.1	-	1.0	-	**13.6**
usps	50	5.3	4.1	-	1.0	-	**11.8**
$n = 7291$	200	16.8	4.4	-	1.0	-	**23.6**
$d = 256$	1000	**18.5**	2.7	-	1.0	-	-
yale	50	2.1	4.2	6.3	1.0	0.6	**17.9**
$n = 2414$	200	**21.9**	2.9	-	1.0	-	13.9
$d = 32256$	1000	-	**1.9**	-	1.0	-	-
avg. speedup		8.7	3.6	4.7	1.0	1.1	**33.0**

Table 5. Algorithmic speedup in reaching an energy within 1% from the final Lloyd++ energy. (-) marks failure in reaching the target of 1% relative error. For each method, the parameter(s) that gave the highest speedup at 1% error is used.

Figure 2 show the convergence curves corresponding to cifar, cnnvoc, mnist and mnist50 entries in Table 5. Figure 3 shows the convergence curves of AKM and k^2 means under same settings, when varying the parameters m and k_n. On cifar the benefit of k^2-means is clear since it reaches the reference error significantly faster than the other methods. On mnist50 k^2-means is considerably faster than AKM for $k = 1000$ but AKM reaches the 1% reference faster for $k = 50$.

In all settings of Table 5, Elkan++ gives a consistent up to $8.5\times$ speedup (since it is an exact acceleration of Lloyd++). For some settings Elkan is faster than Elkan++ in reaching the desired accuracy. This is due to the faster initialization. MiniBatch fails in all but one case (mnist, $k = 50$) to reach the reference error of 1% and is thus not shown. In 2/40 cases, we do not reach the 1% reference error - since the maximum k_n employed is $k_n = 200$.

For accurate clustering, when the reference energy is the Lloyd++ convergence energy (i.e. 0% error), Table 6 shows that the speedups of k^2-means are even higher. This is partially because in 87.5% of the cases (35/40) we obtain a lower energy than Lloyd++ since our proposed GDI initialization is comparable or better than k-means++

Dataset	k	AKM	Elkan++	Elkan	Lloyd++	Lloyd	k^2**-means**
cifar	50	-	17.8	-	1.0	-	**37.9**
	200	1.2	24.2	-	1.0	-	**139.8**
	1000	11.3	17.5	28.2	1.0	2.6	**373.6**
cnnvoc	50	2.4	9.3	-	1.0	-	**26.2**
	200	3.7	9.3	-	1.0	-	**59.7**
	1000	5.8	**8.1**	-	1.0	-	-
covtype	50	-	28.9	-	1.0	-	**172.0**
	200	-	40.2	-	1.0	-	**442.4**
	1000	-	**44.5**	-	1.0	-	-
mnist	50	1.1	17.3	26.6	1.0	2.9	**39.3**
	200	-	25.8	-	1.0	-	**81.0**
	1000	9.0	29.8	-	1.0	-	**141.1**
mnist50	50	-	18.7	-	1.0	-	**31.0**
	200	2.1	26.6	-	1.0	-	**80.3**
	1000	5.1	22.7	-	1.0	-	**94.1**
tinygist10k	50	12.5	12.5	20.1	1.0	3.6	**50.1**
	200	4.7	11.0	-	1.0	-	**71.8**
	1000	2.6	**7.8**	-	1.0	-	-
usps	50	-	12.6	-	1.0	-	**31.7**
	200	3.4	14.6	-	1.0	-	**54.4**
	1000	-	**9.4**	-	1.0	-	-
yale	50	2.8	9.5	-	1.0	-	**32.5**
	200	**20.8**	6.5	-	1.0	-	18.7
	1000	-	**4.0**	-	1.0	-	-
avg. speedup		5.9	17.9	25.0	1.0	3.0	**104.1**

Table 6. Algorithmic speedup in reaching the same energy as the final Lloyd++ energy. (-) marks failure in reaching the target of 0% relative error. For each method, the parameter(s) that gave the highest speedup at 0% error is used.

(see Table 4). For this setting, the second fastest method is Elkan++, which is designed for accelerating the exact Lloyd++.

4 Conclusions

We proposed k^2-means, a simple yet efficient method ideally suited for fast and accurate large scale clustering ($n > 10000$, $k > 10$, $d > 50$). k^2-means combines an efficient divisive initialization with a new method to speed up the k-means iterations by using the k_n nearest clusters as the new set of candidate centers for the cluster members as well as triangle inequalities. The algorithmic complexity of our k^2-means is sublinear in k for $n \gg k$ and experimentally shown to give a high accuracy on diverse datasets. For accurate clustering, k^2-means requires an order of magnitude fewer computations than alternative methods such as the fast approximate k-means (AKM) clustering. Moreover, our efficient divisive initialization leads to comparable clustering energies and significantly lower runtimes than the k-means++ initialization under the same conditions.

Acknowledgments

This work was supported by the ETH Zurich General Fund OK and by the Google 2017 Faculty Research Award.

References

1. Arthur, D., Vassilvitskii, S.: k-means++: The advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. pp. 1027–1035. Society for Industrial and Applied Mathematics (2007)
2. Bahmani, B., Moseley, B., Vattani, A., Kumar, R., Vassilvitskii, S.: Scalable k-means++. Proceedings of the VLDB Endowment 5(7), 622–633 (2012)
3. Blake, C., Keogh, E., Merz, C.: Uci repository of machine learning databases. (1998), http://www.ics.uci.edu/mlearn/MLRepository.html
4. Boley, D.: Principal direction divisive partitioning. Data mining and knowledge discovery 2(4), 325–344 (1998)
5. Caliński, T., Harabasz, J.: A dendrite method for cluster analysis. Communications in Statistics-theory and Methods 3(1), 1–27 (1974)
6. Ding, Y., Zhao, Y., Shen, X., Musuvathi, M., Mytkowicz, T.: Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. In: Proceedings of the 32nd International Conference on Machine Learning (ICML-15). pp. 579–587 (2015)
7. Drake, J., Hamerly, G.: Accelerated k-means with adaptive distance bounds. In: 5th NIPS workshop on optimization for machine learning (2012)
8. Elkan, C.: Using the triangle inequality to accelerate k-means. In: ICML. vol. 3, pp. 147–153 (2003)
9. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. International journal of computer vision 88(2), 303–338 (2010)
10. Georghiades, A., Belhumeur, P., Kriegman, D.: From few to many: Illumination cone models for face recognition under variable lighting and pose. IEEE Trans. Pattern Anal. Mach. Intelligence 23(6), 643–660 (2001)
11. Hamerly, G.: Making k-means even faster. In: SDM. pp. 130–140. SIAM (2010)
12. Hull, J.J.: A database for handwritten text recognition research. Pattern Analysis and Machine Intelligence, IEEE Transactions on 16(5), 550–554 (1994)
13. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 604–613. ACM (1998)
14. Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C.D., Silverman, R., Wu, A.Y.: A local search approximation algorithm for k-means clustering. In: Proceedings of the eighteenth annual symposium on Computational geometry. pp. 10–18. ACM (2002)
15. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Masters thesis, Department of Computer Science, University of Toronto (2009)
16. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
17. LeCun, Y., Cortes, C., Burges, C.J.: The mnist database of handwritten digits (1998)
18. Lee, K., Ho, J., Kriegman, D.: Acquiring linear subspaces for face recognition under variable lighting. IEEE Trans. Pattern Anal. Mach. Intelligence 27(5), 684–698 (2005)

19. Lloyd, S.P.: Least squares quantization in pcm. Information Theory, IEEE Transactions on 28(2), 129–137 (1982)
20. Mazzeo, G.M., Masciari, E., Zaniolo, C.: A fast and accurate algorithm for unsupervised clustering around centroids. Information Sciences 400401, 63 – 90 (2017)
21. Muja, M., Lowe, D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. VISAPP (1) 2 (2009)
22. Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A.: Object retrieval with large vocabularies and fast spatial matching. In: Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on. pp. 1–8. IEEE (2007)
23. Sculley, D.: Web-scale k-means clustering. In: Proceedings of the 19th international conference on World wide web. pp. 1177–1178. ACM (2010)
24. Su, T., Dy, J.: A deterministic method for initializing k-means clustering. In: Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on. pp. 784–786. IEEE (2004)
25. Torralba, A., Fergus, R., Freeman, W.T.: 80 million tiny images: A large data set for non-parametric object and scene recognition. Pattern Analysis and Machine Intelligence, IEEE Transactions on 30(11), 1958–1970 (2008)
26. Wang, J., Wang, J., Ke, Q., Zeng, G., Li, S.: Fast approximate k-means via cluster closures. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition (2012)
27. Wu, X., Kumar, V., Quinlan, J.R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Philip, S.Y., et al.: Top 10 algorithms in data mining. Knowledge and Information Systems 14(1), 1–37 (2008)
28. Xu, Y., Qu, W., Li, Z., Min, G., Li, K., Liu, Z.: Efficient k -means++ approximation with mapreduce. Parallel and Distributed Systems, IEEE Transactions on 25(12), 3135–3144 (Dec 2014)
29. Zhao, W., Ma, H., He, Q.: Parallel k-means clustering based on mapreduce. In: Cloud Computing, pp. 674–679. Springer (2009)