Speeding up k-means by approximating Euclidean distances via block vectors

Thomas Bottesch[1,2] THOMAS.BOTTESCH@AVIRA.COM
Thomas Bühler[1] THOMAS.BUEHLER@AVIRA.COM
Markus Kächele[2] MARKUS.KAECHELE@UNI-ULM.DE

[1]Avira Operations GmbH & Co. KG, Tettnang, Germany
[2]Institute of Neural Information Processing, Ulm University, Ulm, Germany

Abstract

This paper introduces a new method to approximate Euclidean distances between points using block vectors in combination with the Hölder inequality. By defining lower bounds based on the proposed approximation, cluster algorithms can be considerably accelerated without loss of quality. In extensive experiments, we show a considerable reduction in terms of computational time in comparison to standard methods and the recently proposed Yinyang k-means. Additionally we show that the memory consumption of the presented clustering algorithm does not depend on the number of clusters, which makes the approach suitable for large scale problems.

1. Introduction

The problem of finding meaningful clusters in large quantities of data is an essential task with a wide range of applications in data analysis and machine learning. Intuitively, the clustering problem can be described as finding groups of points which are similar to each other but are different from the members of other groups. One of the most popular clustering techniques is k-means clustering. The standard algorithm for the k-means problem (Lloyd, 1982) has been applied to a wide range of applications including image segmentation and compression (Kanungo et al., 2002), information retrieval (Steinbach et al., 2000), and bioinformatics (Yeung et al., 2001). Among the reasons for the popularity of Lloyd's algorithm is its simplicity, geometric intuition as well as its experimentally proven ability to find meaningful clusters.

At the heart of the k-means algorithm is the computation of distances between the points and the cluster centers. However, in a naïve implementation of the algorithm, one would need to compute the Euclidean distances between all points and all cluster centers in the assignment step. For large sample sizes, this becomes the main bottleneck and prevents the algorithm from being scalable to large datasets and high dimensions. Moreover, note that the standard version of k-means is not adapted to large sparse datasets.

For this reason, recent research by several authors focused on accelerating k-means. Their approaches can be divided into two categories. The algorithms of the first category try to find an *approximate* solution of the standard k-means efficiently, see e.g. Philbin et al. (2007); Sculley (2010); Wang et al. (2012). A second class of algorithms is *exact* in the sense that it can be guaranteed that the same result as standard k-means is achieved, given the same initialization. Examples of this type of methods are techniques based on more efficient data structures (Alsabti, 1998; Pelleg & Moore, 1999; Kanungo et al., 2002) or methods utilizing the triangle inequality (Elkan, 2003; Hamerly, 2010; Drake & Hamerly, 2012; Ding et al., 2015).

In this paper, we propose a set of exact optimization techniques for k-means which decrease the computational time while guaranteeing the same results as standard k-means. Our techniques are based on efficiently obtained lower bounds on the distances to the cluster centers, allowing us to skip corresponding distance calculations. In recent research (Ding et al., 2015) it was shown that an initial grouping of the cluster centers can be used to obtain an efficient filtering mechanism to avoid redundant distance calculations. This approach is known as Yinyang k-means. We show that a similar technique is possible by utilizing a grouping of the *input dimensions*, which will be referred to as *blockification*. Thus, our approach can be seen as complementary to the one by Ding et al. (2015). Indeed, we will see in the experiments that the best results are obtained by combining the strengths of both methods.

The remainder of the paper is organized as follows: Section 2 contains an overview of the k-means algorithm and sev-

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48. Copyright 2016 by the author(s).

eral approaches for its algorithmic improvement. Section 3 deals with lower bounds on the Euclidean distances between points and cluster centers. In particular, we discuss how to use the bounds which originated from *blockification*. These results are applied in Section 4 to obtain exact optimizations of standard and Yinyang k-means. Finally, in Section 5, a series of experiments is performed demonstrating the superiority of our method to previous approaches.

2. k-means clustering

The k-means cost function is a widely-used clustering criterion. The goal is to partition some given data $x_1 \dots x_n \in \mathbb{R}^d$ into k disjoint clusters $\mathcal{C} = \{C_1, \dots, C_k\}$ such that the within-cluster sum of squared distortions is minimized. Given a set of k cluster centers $C = \{c_1, \dots, c_k\}$, the k-means objective is defined as (see e.g. Hastie et al., 2001)

$$\mathrm{cost}(\mathcal{C}, C) = \sum_{j=1}^{k} \sum_{x_i \in C_j} d(x_i, c_j)^2, \tag{1}$$

where $d(x, y)$ denotes the Euclidean distance between x and $y \in \mathbb{R}^d$, which we will formalize in the next section.

Geometrically, the above cost function means that our aim is to find groups of points surrounding the cluster centers $c_1, \dots, c_k$. Note that this is a challenging problem since the cost function in (1) is non-convex. In fact, one can show that the problem of finding the global minimum of the k-means objective is an NP-hard problem (Dasgupta, 2007; Mahajan et al., 2009; Vattani, 2009). Thus, several techniques have been proposed to find locally optimal solutions of (1). The most popular technique is *Lloyd's algorithm* (Lloyd, 1982), which is commonly just referred to as the *k-means algorithm*. The main idea is to compute a locally optimal solution of (1) by iterating between determining the assignment to the nearest clusters and finding the centers of the current cluster assignment.

Algorithm 1 Lloyd's algorithm for k-means

```
Input: data $X = \{x_1 \dots x_n\} \subset \mathbb{R}^d$, no. of clusters $k \in \mathbb{N}$
Initialization: random centers $c_1 \dots c_k \in \mathbb{R}^d$
repeat
  Find closest cluster centers:
    $\forall x \in X$ :        $b(x) = \mathrm{argmin}_{j=1 \dots k}\{d(x, c_j)\}$
  Update cluster centers $c_1 \dots c_k$:
    $\forall j \in \{1 \dots k\}$ :     $c_j = \mathrm{mean}(\{x \in X \mid b(x) = j\})$
until cluster centers stop changing
Output: cluster centers $c_1 \dots c_k$
```

This process, shown in Alg. 1, is repeated until the clusters do not change anymore. One can show that each of the two steps of Alg. 1 reduces the objective in (1) and therefore one converges towards a local minimum of (1), see e.g. Hastie et al. (2001). However, note that due to the non-convexity of (1) the convergence to a global minimum is not guaranteed. Hence, several authors proposed different techniques to find good initializations of Lloyd's algorithm, see e.g. Arthur & Vassilvitskii (2007); Ostrovsky et al. (2013); Bahmani et al. (2012).

Despite its lack of theoretical guarantees, the algorithm in Alg. 1 is one of the most widely used clustering algorithms (Wu et al., 2007). One reason for the popularity of Alg. 1 is its simplicity. However, note that in a naïve implementation, in the assignment step, the Euclidean distances between all points and all cluster centers have to be computed, which is of complexity $O(nk)$. Since this is not feasible for a large number of points or clusters, several authors considered more optimized variants of Alg. 1, which will be discussed in the following.

2.1. Previous work on optimizing k-means

Several authors proposed different variants of k-means with the goal of reducing the amount of required distance computations. One approach is to organize the data in a k-d tree, which is a binary tree constructed by recursively splitting the data so that each node represents the corresponding subspace. Thus, the tree corresponds to a hierarchical partitioning which is used to find the closest points to a given cluster center efficiently (Alsabti, 1998; Pelleg & Moore, 1999; Kanungo et al., 2002). However, these methods only work well for small dimensions, as for large dimensions traversing the tree can become prohibitively expensive.

A different approach is to effectively remove redundant distance calculations. Several authors observed that a simple enhancement is possible by using the fact that some cluster centers do not change in successive iterations: if during the cluster update step, the assigned center moves closer to a point x, all centers which did not move cannot be closer to x, thus the corresponding distance calculations can be skipped (Kaukoranta et al., 2000; Fahim et al., 2006).

In the method by Elkan (2003), a matrix of pairwise distances between cluster centers is maintained in each step. Then, the triangle inequality is applied to obtain lower bounds on the distances between points and centers, which are then used to eliminate redundant distance calculations. While this technique causes a significant speed-up compared to standard k-means, maintaining the bounds implies memory and runtime requirements of $O(kn)$ and hence it is unfeasible for large cluster sizes. Hamerly (2010) proposed a modification using only one lower bound per point to the distance to its second-closest center. In low dimensions, this technique yields a better trade-off between the time needed to maintain the bounds and the time saved by skipping distance calculations. Drake & Hamerly (2012) combined the strengths of the above methods by considering lower bounds to the b next-closest centers, where the value of b is found by an adaptive tuning strategy.

A related problem from the signal processing community is the task of vector quantization by finding a good codebook. Based on similar lower bounds on the distance to a given code vector, several authors derived sufficient conditions which guarantee that the considered code vector cannot be optimal. For instance, Mielikainen (2002) proposed a lower bound based on cosine similarities. Torres & Huguet (1994) presented a bound using the max-norm, and Wu & Lin (2000) proposed a bound following from the Cauchy-Schwarz inequality, both of which can be seen as special cases of the bound we will discuss in Section 3. Furthermore, similar techniques were used for matrix compression in the context of top-k search (Low & Zheng, 2012). We are not aware of any work accelerating k-means using bounds based on the blockification strategy proposed in this paper.

The above methods are *exact* in the sense that they can guarantee to achieve the same result as Lloyd's algorithm, given the same initialization. In a different line of research, several authors developed methods to compute *approximate* solutions of the k-means problem very efficiently. Examples include approaches based on random subsampling (Sculley, 2010) and approximate nearest neighbour search (Philbin et al., 2007; Wang et al., 2012). While these approximate techniques have been shown to lead to significant speed-ups compared to standard k-means, they cannot guarantee to obtain the same results as Lloyd's algorithm.

2.2. Yinyang k-means

Recently, Ding et al. (2015) proposed an enhanced version of k-means which we will discuss in the following. At the heart of their algorithm is the idea of filtering unnecessary distance calculations by using continuously maintained lower bounds on the distances of each point to the cluster centers, as well as an upper bound to the cluster center it is currently assigned to.

The algorithm, which is shown in Alg. 2, utilizes a coarse partitioning of the cluster centers into t groups $\mathcal{G} = \{G_1 \ldots G_t\}$. Ding et al. (2015) proposed to obtain this grouping by means of five iterations of standard k-means. The groups are then used as follows: given a point x, a group G, and a lower bound $\mathrm{lb}(x, G)$ on the minimum distance of x to any cluster center $c \in G$, one observes that if there is a different cluster center c_b such that the condition $\mathrm{lb}(x, G) > d(x, c_b)$ holds, no $c \in G$ can be closer to x than c_b. Therefore, all distance calculations to elements of G can be avoided. If the above condition holds for all $G_i \in \mathcal{G}$, all centers are at least as far away as c_b. Finally, if the stronger condition $\mathrm{lb}(x, G_i) > \mathrm{ub}(x)$ holds, even the calculation of $d(x, c_b)$ is unnecessary. Here, $\mathrm{ub}(x)$ is an upper bound on the distance between x and its closest cluster center. Thus, checking the above conditions in reverse order, as done in the *global* and *group filtering* stages in Alg. 2, yields an efficient way to reduce the number of needed distance calculations.

Algorithm 2 Yinyang k-means

```
**Input:** data $X = \{x_1 \ldots x_n\} \subset \mathbb{R}^d$, number of clusters $k \in \mathbb{N}$, number of groups $t \leq \lfloor \frac{k}{10} \rfloor$
**Initialize centers/groups:** random centers $c_1 \ldots c_k \in \mathbb{R}^d$ with indexes split into $t$ groups $G_1 \ldots G_t \subset \{1 \ldots k\}$
**Initialize bounds:** after one $k$-means iteration: $\forall x \in X$: $b(x) = \mathrm{argmin}_{j=1 \ldots k}\{d(x, c_j)\}$, $\mathrm{ub}(x) = d(x, c_{b(x)})$ and $\forall i \in \{1 \ldots t\}$: $\mathrm{lb}(x, G_i) = \min\{d(x, c_j) | j \in G_i - b(x)\}$
**repeat**
    Update cluster centers and center/group drifts:
    **for** $j = 1 \ldots k$ **do**
        $c'_j = c_j$;   $c_j = \mathrm{mean}(\{x \in X \mid b(x) = j\})$
        $\delta(c_j) = d(c_j, c'_j)$
    Set $\delta(G_i) = \max_{j \in G_i} \delta(c_j)$ for $i = 1 \ldots t$
    **for** $x = x_1 \ldots x_n$ **do**
        Use shift to update bounds:
            $\mathrm{ub}(x) = \mathrm{ub}(x) + \delta(c_{b(x)})$
            $\mathrm{lb}(x, G_i)_{old} = \mathrm{lb}(x, G_i)$ for $i = 1 \ldots t$
            $\mathrm{lb}(x, G_i) = \mathrm{lb}(x, G_i) - \delta(G_i)$ for $i = 1 \ldots t$
            $b(x)_{old} = b(x)$
        **Global filtering**:
            If $\min_{i=1}^{t} \mathrm{lb}(x, G_i) \geq \mathrm{ub}(x)$: **continue**
            Tighten bound: $\mathrm{ub}(x) = d(x, c_{b(x)})$
            If $\min_{i=1}^{t} \mathrm{lb}(x, G_i) \geq \mathrm{ub}(x)$: **continue**
        **Group filtering**:
            Let $\widehat{\mathcal{G}} = \{i \in \{1 \ldots t\} \mid \mathrm{lb}(x, G_i) < \mathrm{ub}(x)\}$
            Set $\mathrm{lb}(x, G_i) = \infty, \forall i \in \widehat{\mathcal{G}}$
        **Local filtering**:
            Perform local filtering according to Alg. 3
**until** cluster centers stop changing
**Output:** cluster centers $c_1 \ldots c_k$
```

Algorithm 3 Local filtering in Yinyang k-means

```
**for** $i \in \widehat{\mathcal{G}}$ **do**
    **for** $j \in G_i$ **do**
        If $b(x)_{old} = j$: **continue**
        If $\mathrm{lb}(x, G_i) < \mathrm{lb}(x, G_i)_{old} - \delta(c_j)$: **continue**
        If $d(x, c_j) < \mathrm{ub}(x)$:
            Find $l \in \{1 \ldots t\}$ s.t. $b(x) \in G_l$
            $\mathrm{lb}(x, G_l) = \mathrm{ub}(x)$
            $\mathrm{ub}(x) = d(x, c_j)$;   $b(x) = j$
        else if $d(x, c_j) < \mathrm{lb}(x, G_i)$:
            $\mathrm{lb}(x, G_i) = d(x, c_j)$
```

In the *local filtering* stage (Alg. 3), one now considers every cluster center c in the remaining groups G_i. Here, similarly to before one checks whether for a different cluster center c_b, the condition $d(x, c_b) < \mathrm{lb}(x, G_i) - d(c, c')$ is valid, where c' is the location of the cluster center at the previous iteration. In this case, c cannot be optimal, which again follows from the triangle inequality (Ding et al., 2015). The resulting steps can be seen in Alg. 3. While the global and local filtering steps introduced by Ding et al. (2015) typically yield a significant speed-up compared to standard k-means, note that in the beginning the algorithm still needs to compute one full iteration of k-means, which requires the computation of all distances to the centers.

One recurring theme in the optimizations of k-means discussed above is that they rely on the computation of lower bounds on the distances to the cluster centers which are then used to eliminate the actual computation of the distance. The efficiency of the resulting method hinges on the tightness of the bounds as well as the efficiency in which the bounds can be computed. For this reason, in the following section, we will further investigate the question how to obtain lower bounds on the Euclidean distance.

3. Lower bounds on the Euclidean distance

One of the main building blocks of the k-means algorithm is the computation of Euclidean distances between points $x \in \mathbb{R}^d$ and cluster centers $c \in \mathbb{R}^d$. In standard k-means, the distances of all n points to k cluster centers need to be computed in every iteration. Thus, if d is large, the computation of the Euclidean distances becomes the main bottleneck of the k-means algorithm.

For this reason, it is crucial for the performance of the algorithm to limit the number of computed distance calculations. The main idea is the following: assume that, for a given point $x \in \mathbb{R}^d$, one has already computed the distance $d(x, c)$ to a given cluster center $c \in \mathbb{R}^d$. Assume now that for a different cluster center $c' \in \mathbb{R}^d$ one has obtained a lower bound $\mathrm{lb}(x, c')$ on the distance $d(x, c')$. If it holds that $\mathrm{lb}(x, c') \geq d(x, c)$, then clearly also $d(x, c')$ has to be larger than $d(x, c)$. Thus we can conclude that c' cannot be the closest cluster center and the computation of the distance to $d(x, c')$ can be skipped.

In order for this technique to work efficiently, the lower bound $\mathrm{lb}(x, c')$ ideally needs to fulfill two conditions: on one hand, it needs to be as tight as possible, in order to maximize the probability that indeed $\mathrm{lb}(x, c') \geq d(x, c)$. On the other hand, one needs to be able to compute it efficiently, in order to achieve the desired overall improvement in runtime and memory requirement. In practice, one usually observes a certain trade-off between a bound which is easy to compute but quite loose, and a very tight bound which leads to many skipped distance calculations but is more costly. We will now discuss several different techniques to compute lower bounds on the Euclidean distance.

In the following, let $\|x\|_p$ denote the l^p-norm of $x \in \mathbb{R}^d$, defined for $p \geq 1$ as $\|x\|_p := (\sum_{i=1}^n |x_i|^p)^{1/p}$. Moreover, we use the usual definition $\|x\|_\infty := \lim_{p\to\infty} \|x\|_p = \max_i |x_i|$ (Horn & Johnson, 1990). For $p = 2$, one obtains the standard Euclidean norm. Moreover, for points $x, y \in \mathbb{R}^d$, the Euclidean distance is given as $d(x, y) := \|x - y\|_2$. Our goal is to find lower bounds on $d(x, y)$.

3.1. Bounds based on Hölder's inequality

We start by giving a simple bound on the Euclidean distance which is based on Hölder's inequality.

Proposition 3.1. *Let $x, y \in \mathbb{R}^d$. For any $1 \leq p, q \leq \infty$ with $\frac{1}{p} + \frac{1}{q} = 1$, a lower bound on $d(x, y)$ is given by*

$$\mathrm{lb}_p^H(x, y) := \sqrt{\max\left\{0, \|x\|_2^2 + \|y\|_2^2 - 2\|x\|_p\|y\|_q\right\}}.$$

Proof. Writing the squared Euclidean distance as $\|x - y\|_2^2 = \|x\|_2^2 + \|y\|_2^2 - 2\langle x, y\rangle$, one can apply Hölder's inequality $\langle x, y\rangle \leq \|x\|_p\|y\|_q$, which directly yields $d(x, y)^2 \geq \|x\|_2^2 + \|y\|_2^2 - 2\|x\|_p\|y\|_q$. Clearly also $d(x, y)^2 \geq 0$. The result then follows by taking the square root and using that $d(x, y) \geq 0$ since d is a norm. □

One now immediately obtains the following lower bound (which is the analogous result based on the Cauchy-Schwarz inequality) as special case.

Corollary 3.2. *Let $x, y \in \mathbb{R}^d$. A lower bound on the Euclidean distance $d(x, y)$ is given by*

$$\mathrm{lb}^C(x, y) := \sqrt{\|x\|_2^2 + \|y\|_2^2 - 2\|x\|_2\|y\|_2}.$$

Proof. This follows from Prop. 3.1 for $p = q = 2$, using that $\|x\|_2^2 + \|y\|_2^2 - 2\|x\|_2\|y\|_2 = (\|x\|_2 - \|y\|_2)^2 \geq 0$. □

Thus, for $p = q = 2$, an approximation of the inner product is given by $\|x\|_2\|y\|_2$. Moreover, one also observes from Prop. 3.1 that in the limit cases $p \to 1$ and $q \to 1$ the inner product becomes approximated by $\|x\|_1 \max_i |y_i|$ and $\max_i |x_i| \, \|y\|_1$, respectively. Thus, the general result in Prop. 3.1 includes the bounds utilized by Torres & Huguet (1994) and Wu & Lin (2000) in the context of vector quantization as special cases.

Prop. 3.1 means that x has a distance of at least $\mathrm{lb}_p^H(x, y)$ to y. Suppose x is a point and c is a cluster center. With the reasoning given at the beginning of this section, all $c' \in C$ with $\mathrm{lb}_p^H(x, c') \geq d(x, c)$ cannot be closer to x than c and can be skipped in the distance calculation step.

Assume that the values of $\|x\|_p$ and $\|c\|_q$ are precomputed, then for any choice of p and q the bound in Prop. 3.1 can be obtained very efficiently. However, while it can already lead to some avoided computations it is too loose to be effective in general. To see this, consider the following example for the case $p = 2$: Suppose $x \in \mathbb{R}^d$ is a point with $x_1 = 1$ and zero else, and $y \in \mathbb{R}^d$ is a second point only consisting of ones. Then the dot product $\langle x, y\rangle$ between these two points is equal to 1. The approximation $\|x\|_2\|y\|_2$ however is $\sqrt{d}$, which can be considerably higher, thus leading to a loose lower bound.

3.2. Using block vectors to tighten the lower bounds

We will first describe the *blockification* process which will be used to obtain tighter lower bounds. A point $x \in \mathbb{R}^d$ can be subdivided into b *blocks* $x_{(1)} \dots x_{(b)}$ of sizes ≥ 1 by splitting along its dimensions, i.e. one has $x_{(1)} = (x_1 \dots x_{l_1})$, $x_{(2)} = (x_{l_1+1} \dots x_{l_2})$ and so on, where $0 = l_0 < \dots < l_b = d$ is an increasing sequence of indexes. Then a vector $x_{\mathbb{B}p} \in \mathbb{R}^b$ with $b \leq d$ which is referred to as *block vector* is constructed by calculating the p-norm of every block and setting $(x_{\mathbb{B}p})_i := \|x_{(i)}\|_p$.

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_{d-1} \\ x_d \end{pmatrix} \longrightarrow \begin{pmatrix} \left\| \begin{pmatrix} x_1 \\ \cdots \\ x_i \end{pmatrix}_{(1)} \right\|_p \\ \cdots \\ \left\| \begin{pmatrix} x_j \\ \cdots \\ x_d \end{pmatrix}_{(b)} \right\|_p \end{pmatrix} = x_{\mathbb{B}p} \qquad (2)$$

Note that above for simplicity we used the notation $i := l_1$ and $j := l_{d-1}+1$. The *blockification* process is equal to the compression technique of Low & Zheng (2012) applied to a one-dimensional matrix when replacing the used norm. The above block vector can now be used to obtain a tighter lower bound than the one in Prop. 3.1.

Lemma 3.3. *(Tightened Hölder's Inequality). Let $x, y \in \mathbb{R}^d$, and let $x_{\mathbb{B}p}, y_{\mathbb{B}q} \in \mathbb{R}^b$ be block vectors as defined above, where $1 \leq p, q \leq \infty$ with $\frac{1}{p} + \frac{1}{q} = 1$. Then*

$$\langle x, y\rangle \leq \langle x_{\mathbb{B}p}, y_{\mathbb{B}q}\rangle \leq \|x\|_p \|y\|_q.$$

Proof. The inner product can be decomposed into blocks as $\langle x, y\rangle = \sum_{i=1}^{b} \langle x_{(i)}, y_{(i)}\rangle$. By Hölder's inequality, one has for each of the blocks, $\langle x_{(i)}, y_{(i)}\rangle \leq \|x_{(i)}\|_p \|y_{(i)}\|_q$ and hence $\langle x, y\rangle \leq \sum_{i=1}^{b} \|x_{(i)}\|_p \|y_{(i)}\|_q = \langle x_{\mathbb{B}p}, y_{\mathbb{B}q}\rangle$. On the other hand, again due to Hölder one has $\langle x_{\mathbb{B}p}, y_{\mathbb{B}q}\rangle \leq \|x_{\mathbb{B}p}\|_p \|y_{\mathbb{B}q}\|_q$. By construction of $x_{\mathbb{B}p}$, it holds that $\|x_{\mathbb{B}p}\|_p^p = \sum_{i=1}^{b} |x_{\mathbb{B}p}|_i^p = \sum_{i=1}^{b} \|x_{(i)}\|_p^p = \sum_{i=1}^{b} \sum_{j=l_{i-1}+1}^{l_i} |x_j|^p = \sum_{j=1}^{d} |x_j|^p = \|x\|_p^p$ and analogously for $y_{\mathbb{B}q}$. It follows that $\|x_{\mathbb{B}p}\|_p \|y_{\mathbb{B}q}\|_q = \|x\|_p \|y\|_q$, which concludes the proof. □

Different values of b lead to different intuitive solutions. If $b = n$ then $x_{\mathbb{B}p} = x$ and $\langle x, y\rangle = \langle x_{\mathbb{B}p}, y_{\mathbb{B}q}\rangle$ and hence the left inequality in Lemma 3.3 is tight. On the other hand, if $b = 1$ then $\langle x_{\mathbb{B}p}, y_{\mathbb{B}q}\rangle = \|x\|_p \|y\|_q$, and we obtain Hölder's inequality. With the option to choose b the quality of the dot product approximation can be controlled. Note that in the case where $x_{(i)} = 0$ for some block i (or analogously $y_{(i)} = 0$), we have $0 = \langle x_{(i)}, y_{(i)}\rangle \leq \|x_{(i)}\|_p \|y_{(i)}\|_q = 0$, which means that in this case, the corresponding part of the inner product $\langle x, y\rangle$ gets approximated exactly. Note that in this case the corresponding part $y_{(i)} \neq 0$ (or analogously $x_{(i)} \neq 0$) would still contribute to $\|x\|_p \|y\|_q$ while it has no influence on $\|x_{(i)}\|_p \|y_{(i)}\|_q$. For large sparse datasets, typically this case occurs very frequently.

Corollary 3.4. *(Tightened Cauchy-Schwarz Inequality). Let $x, y \in \mathbb{R}^d$, and let $x_{\mathbb{B}2}, y_{\mathbb{B}2} \in \mathbb{R}^b$ be block vectors as defined above. Then it holds that*

$$\langle x, y\rangle \leq \langle x_{\mathbb{B}2}, y_{\mathbb{B}2}\rangle \leq \|x\|_2 \|y\|_2.$$

The above tightened versions of Hölder and Cauchy-Schwarz inequality can now be used to obtain the following tightened lower bounds on the Euclidean distance.

Proposition 3.5. *Let $x, y \in \mathbb{R}^d$, and let $x_{\mathbb{B}p}, y_{\mathbb{B}q}$ be block vectors as defined above, where $1 \leq p, q \leq \infty$ with $\frac{1}{p} + \frac{1}{q} = 1$. Then a lower bound on $d(x, y)$ is given by*

$$\mathrm{lb}_p^B(x, y) := \sqrt{\max\left\{0, \|x\|_2^2 + \|y\|_2^2 - 2\langle x_{\mathbb{B}p}, y_{\mathbb{B}q}\rangle\right\}}.$$

Proof. Writing $\|x - y\|_2^2 = \|x\|_2^2 + \|y\|_2^2 - 2\langle x, y\rangle$ and applying Lemma 3.3, one obtains $d(x, y)^2 \geq \|x\|_2^2 + \|y\|_2^2 - 2\langle x_{\mathbb{B}p}, y_{\mathbb{B}q}\rangle$. The result follows since $d(x, y) \geq 0$. □

Corollary 3.6. *Let $x, y \in \mathbb{R}^d$, and let $x_{\mathbb{B}p}, y_{\mathbb{B}q}$ be block vectors as defined above, where $1 \leq p, q \leq \infty$ with $\frac{1}{p} + \frac{1}{q} = 1$. Then a lower bound on $d(x, y)$ is given by*

$$\mathrm{lb}_2^B(x, y) := \sqrt{\|x\|_2^2 + \|y\|_2^2 - 2\langle x_{\mathbb{B}2}, y_{\mathbb{B}2}\rangle}.$$

Proof. This follows from Prop. 3.5 by setting $p = q = 2$ and using that $\langle x_{\mathbb{B}2}, y_{\mathbb{B}2}\rangle \leq \|x\|_2 \|y\|_2$, which implies that $\|x\|_2^2 + \|y\|_2^2 - 2\langle x_{\mathbb{B}2}, y_{\mathbb{B}2}\rangle \geq (\|x\|_2 - \|y\|_2)^2 \geq 0$. □

With the *block vectors* precomputed, the evaluation of the lower bound in Prop. 3.5 is very efficient. However, for a too high value of b the precomputation and storing of the *block vectors* becomes the main bottleneck and hence obtaining the lower bound can get very expensive. On the other hand, while for the choice of $b = 1$, the lower bound in Prop. 3.5 coincides with the one in Prop. 3.1, in the case of a high value of b one obtains a better approximation of the Euclidean distance. To see this, let us again consider our example from the end of the previous section where the dot product of the vectors x and y was 1. Suppose the dimensions of x and y are split into b blocks of equal sizes. The approximation with *block vectors* then is $\langle x_{\mathbb{B}2}, y_{\mathbb{B}2}\rangle = \sqrt{d/b}$ which can be considerably closer to 1 in this ideal example. We will further investigate the trade-off between achieving a good approximation of the Euclidean distance and fast computation of the bounds for different choices of b in the experimental section.

4. Exact optimizations for k-means and Yinyang k-means

In the following we show how the proposed techniques are used to derive optimized variants of standard k-means as well as *yinyang* k-means. For simplicity of notation, we restrict ourselves to the case $p = 2$ and omit the subscripts for p in the notation for the block vectors, norms, etc. However, all results also carry over to the general case.

4.1. k-means lower bound optimized

In general *yinyang* and *Elkans's* k-means save a lot of distance calculations. However, with increasing k the memory consumption and computational cost required to maintain the necessary data structures grow linearly for *yinyang* and quadratically for *Elkans's* k-means, making them slow for large numbers of clusters. In Alg. 4 we mitigate this problem by using the lower bounds from Section 3, which yield almost no maintenance costs with increasing k compared to the previously mentioned algorithms.

Algorithm 4 kmeans_optimized

```
Input: data $X = \{x_1 \dots x_n\} \subset \mathbb{R}^d$, no. of clusters $k \in \mathbb{N}$
Initialization: random centers $c_1 \dots c_k \subset \mathbb{R}^d$
Precalculate $\forall x \in X$: $||x||$, $x_{\mathbb{B}}$ and initialize $b(x)$ with any center index $i = 1 \dots k$
Set $Y = \emptyset$, $Z = \emptyset$
repeat
  Precalculate $||c_j||$ and $c_{j_{\mathbb{B}}}$ for $j = 1 \dots k$
  for $x = x_1 \dots x_n$ do
    for $j = 1 \dots k$ do
      If $x \in Z$ and $c_j \in Y$: continue
      If $\mathrm{lb}^C(x, c_j) \geq d(x, c_{b(x)})$: continue
      If $\mathrm{lb}^B(x, c_j) \geq d(x, c_{b(x)})$: continue
      If $d(x, c_j) < d(x, c_{b(x)})$: $b(x) = j$
  $Y = \emptyset, Z = \emptyset$;
  for $j = 1 \dots k$ do
    $c'_j = c_j$;   $c_j = \mathrm{mean}(\{x \in X \mid b(x) = j\})$
    If $(d(c_j, c'_j) = 0)$: $Y = Y \cup \{c_j\}$
    $Z = Z \cup \{x \in X \mid b(x) = j \wedge d(x, c_j) \leq d(x, c'_j)\}$
until cluster centers stop changing
Output: centers $c_1 \dots c_k$
```

The key features of Alg. 4 are the two precalculation steps. Initially all data required to compute the lower bounds are computed for all points x. At the start of every iteration the same is done for all cluster centers $c_1 \dots c_k$. While iterating through the centers for every x the lower bounds in the order *cheapest first* are evaluated. Only if no lower bound condition was met $d(x, c)$ has to be calculated and c eventually becomes the new closest center pointed to by $b(x)$. After updating $b(x)$ for all x, the clusters are shifted analogously to standard k-means.

A consequence of the proposed technique is the need to calculate the lower bounds between every point and center in every iteration. In the initial phase this is hardly an issue but near convergence, when few clusters are shifting, the additional cost of calculating the lower bounds becomes severe. We avoid this by using the following additional optimization, see e.g. (Kaukoranta et al., 2000; Fahim et al., 2006): Let c' be the closest cluster center to x before shifting, and c the center after shifting. If $d(x, c) \leq d(x, c')$, then all cluster centers $c_1 \dots c_k$ which did not shift in the last iteration cannot be closer to x than c. Thus, in Alg. 4 we maintain two sets Y and Z: the set Y contains all cluster centers which did not shift in the last iteration, while the set Z contains all points x with $d(x, c) \leq d(x, c')$.

One main disadvantage of *yinyang* is that in order to save any computations, in a preprocessing iteration a regular k-means step of complexity of $O(nk)$ has to be performed. Often more than 50% of the running time of *yinyang* is spent in this step. The design of Alg. 4 avoids this issue and even with an initial random choice of $c_{b(x)}$ it saves a considerable amount of calculations in the first iteration, giving it an advantage in the initial phase of k-means.

4.2. Yinyang k-means lower bound optimized

In *yinyang* at every step $\mathrm{lb}(x, G_i)$ contains an exact distance to a cluster in the local filtering stage. In order to achieve a faster algorithm we propose to store lower bounds in $\mathrm{lb}(x, G_i)$ instead of exact distances. This is achieved by three lines of code (marked grey) added to the local filtering step of *yinyang* (see Alg. 5). If the lower bound $\mathrm{lb}^B(x, c_j)$ is larger than $\mathrm{ub}(x)$ then $d(x, c_j)$ cannot become the new upper bound. However it can still become the new $\mathrm{lb}(x, G_i)$. The normal procedure would be to calculate $d(x, c_j)$ and check if it is smaller than $\mathrm{lb}(x, G_i)$. If it is smaller, $\mathrm{lb}(x, G_i)$ would be set to $d(x, c_j)$. Instead Alg. 5 only checks if $\mathrm{lb}^B(x, c_j)$ is smaller than $\mathrm{lb}(x, G_i)$. If it is smaller, $\mathrm{lb}(x, G_i)$ gets set to $\mathrm{lb}^B(x, c_j)$.

Algorithm 5 Local filtering in fast_yinyang

```
for $i \in \widehat{\mathcal{G}}$ do
  for $j \in G_i$ do
    If $b(x)_{old} = j$: continue
    If $\mathrm{lb}(x, G_i) < \mathrm{lb}(x, G_i)_{old} - \delta(c_j)$: continue
    If $\mathrm{lb}^B(x, c_j) \geq \mathrm{ub}(x)$:
      If $\mathrm{lb}^B(x, c_j) < \mathrm{lb}(x, G_i)$: $\mathrm{lb}(x, G_i) = \mathrm{lb}^B(x, c_j)$
      continue
    If $d(x, c_j) < \mathrm{ub}(x)$:
      Find $l \in \{1 \dots t\}$ s.t. $c_{b(x)} \in G_l$
      $\mathrm{lb}(x, G_l) = \mathrm{ub}(x)$
      $\mathrm{ub}(x) = d(x, c_j)$; $b(x) = j$
    else if $d(x, c_j) < \mathrm{lb}(x, G_i)$:
      $\mathrm{lb}(x, G_i) = d(x, c_j)$
```

The proposed method influences all levels of filtering directly, since $\mathrm{lb}(x, G_i)$ is used in global-, group- and local filtering. Using lower bounds as $\mathrm{lb}(x, G_i)$ can be very efficient in the first iterations, but when converging, using

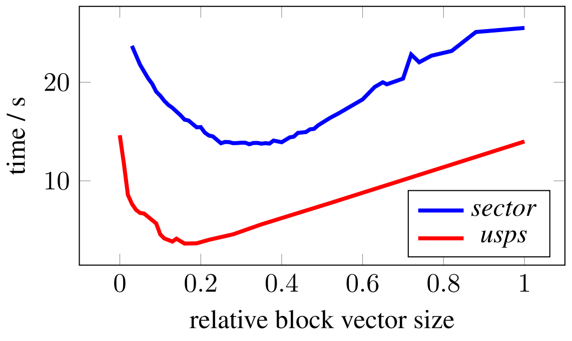

Figure 1. Observing the speed of *kmeans_optimized* for $k = 1000$ while varying the relative block vector size, computed as the average number of non-zero values of the block vectors relative to the average number of non-zero values of X.

exact distances for $\mathrm{lb}(x, G_i)$ is more efficient. Thus a good strategy is to switch to the standard *yinyang* after a number of iterations, which can be done by deactivating the grey marked lines in Alg. 5. We empirically observed a value of 15 iterations to work well in practice. Thus with this proposal we solve *yinyang*'s problem of having a slow start.

5. Experimental evaluation

A series of experiments were carried out using 9 datasets, most of which are taken from the libsvm homepage[1]. Additionally, large scale image and malware datasets were analysed. The sets were selected to cover a broad range of variations in sample size, number of feature dimensions and sparsity, see the left column in Table 2. Prior to experimentation, they have all been scaled to the range between 0 and 1. In the following, for a sample $x \in \mathbb{R}^d$ and analogously for a block vector $x_{\mathbb{B}} \in \mathbb{R}^b$, we denote the number of non-zero components by $\mathrm{nnz}(x)$ or $\mathrm{nnz}(x_{\mathbb{B}})$, respectively. Moreover, for a set of samples $X = \{x_1 \dots x_n\}$, we introduce the notation $\mathrm{annz}(X)$ to denote the average number of non-zero elements, i.e. $\mathrm{annz}(X) := \frac{1}{n}\sum_{i=1}^{n} \mathrm{nnz}(x_i)$. In the experiments, the same initial cluster centers are chosen for all methods. As a consequence, the exact same clustering is generated after every iteration.

Determining the block vector size. To determine a good value for the number of blocks used in the blockification technique described in Section 3, an experiment with the algorithm *kmeans_optimized* was conducted. In Fig. 1 we observe the clustering duration for various block vector sizes for the datasets $usps$ and $sector$ while having a static k of 1000. Note that instead of reporting absolute block vector sizes in the x-axis, we use the relative number of non-zero elements in the block vectors. To be more precise, we use the ratio $\frac{\mathrm{annz}(X_{\mathbb{B}})}{\mathrm{annz}(X)}$, which is more appropriate in a sparse setting. In Fig. 1 we observe that a value b leading to $0.15\,\mathrm{annz}(X) \leq \mathrm{annz}(X_{\mathbb{B}}) \leq 0.4\,\mathrm{annz}(X)$ results in the shortest clustering duration for both datasets. Based on the results from this experiment, in the following the size of the block vectors is chosen in such a way that $\mathrm{annz}(X_{\mathbb{B}}) \approx 0.3\,\mathrm{annz}(X)$. This is achieved by starting off with a static initial block size, and iteratively reducing the block size until $\mathrm{annz}(X_B) < 0.3\,\mathrm{annz}(X)$ holds. In our experiments, this iterative procedure typically needs around 3-4 steps until the condition is met. Creating the block vectors in memory is very cheap compared to computing even one iteration of k-means. If $r = \mathrm{annz}(X) \cdot n$ is the memory needed to store the input matrix X (sparse), then the block vectors $X_{\mathbb{B}}$ require about $0.3r$ memory. The worst case memory consumption due to block vectors is therefore $0.3r$ for $X_{\mathbb{B}}$ plus an additional $0.3r$ for the storage of the cluster center block vectors. This worst case is only reached if every sample is a cluster. On the other hand, to store the groups *yinyang* needs $\frac{k}{10} \cdot n$ memory. Fig. 2 shows that when increasing k, *yinyang* exceeds the constant worst case memory consumption of the block vectors. Storing the block vectors gets cheaper (relative to total memory consumption) with increasing sparsity of X while *yinyang* does not profit from a higher sparsity.

[1] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

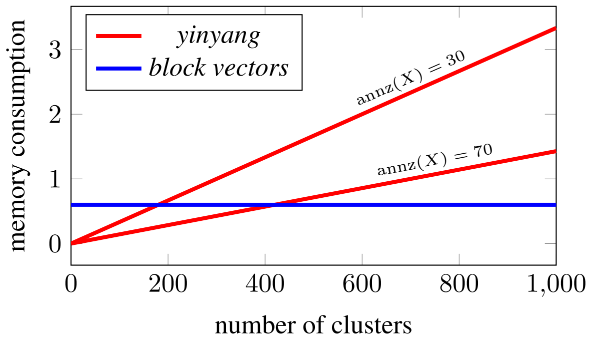

Figure 2. Additional memory consumption (rel. to input size) for *yinyang's* data structures vs. storing the *block vectors*. The worst case memory consumption of the *block vectors* is independent of the number of clusters while it grows linearly for *yinyang*.

Center count dependency. The nature of how block vectors are constructed makes them very interesting especially for sparse data. In a sparse setting, since the cluster centers are computed as the mean of the points in the corresponding cluster, they tend to become more sparse with increasing k. At the same time the evaluation of the dot product between the samples and between the block vectors gets cheaper. Additionally, for sparser data, the approximation of the distances through block vectors gets more accurate. To verify this claim, an experiment was conducted using the algorithm *kmeans_optimized* where the block vector size was fixed and the number of clusters k was varied between 2 and 1000. Figure 3 shows the results of the experiment for the $sector$ and the $usps$ dataset. The y-axis

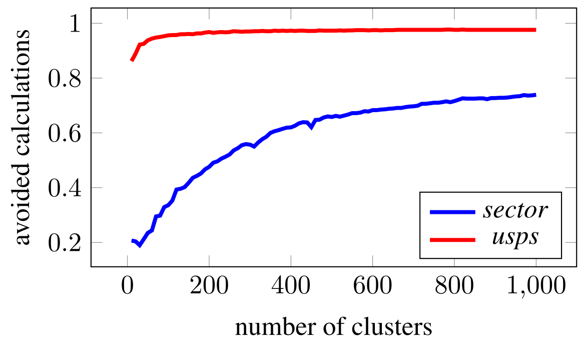

Figure 3. Observing the relative number of full distance calculations avoided by block vectors in *kmeans_optimized* with fixed block vector size while varying the number of clusters k. The number of avoided calculations increases proportionally with k.

Small datasets	100	250	1000
usps	0.956	0.968	0.976
E2006	0.699	0.777	0.811
sector	0.336	0.521	0.739
Medium-sized datasets	100	500	5000
real-sim	0.035	0.143	0.237
mediamill	0.909	0.954	0.978
Large datasets	100	1000	10000
caltech101	0.748	0.855	0.950
kdd2010	0.987	0.997	0.999
avira.201	0.882	0.944	0.976
mnist800k	0.836	0.960	0.984

Table 1. The percentage of avoided distance calculations when running *kmeans_optimized* with different values of k for various datasets. In every case, the number of avoided distance calculation increases proportional to k.

denotes the number of full distance calculations avoided, which indicates how good the lower bound $\mathrm{lb}^B(x, c)$ approximates the actual distance $d(x, c)$. It can be observed that for both datasets the percentage of avoided calculations increases proportionally with the number of k. The same can be observed for all datasets in Table 1.

Clustering duration. A standard k-means algorithm using only the proposed lower bound as optimization retains the original complexity of k-means with $O(nki)$, with i being the number of iterations until convergence. In practice however, the setting of $\mathrm{annz}(X_{\mathbb{B}}) \approx 0.3\,\mathrm{annz}(X)$ adds a linear scaling factor to the runtime. If the lower bounds never avoid any distance calculation, every iteration requires 30% additional work, implying an overall runtime increase of 30%. On the other hand, if the lower bounds always lead to skipping a full distance calculation, the runtime reduces by 70%. In Table 2 empirical evaluations with all proposed algorithms and considered datasets were done to determine the actual speedup over a non optimized standard k-means. The reported results include the time to create the initial block vectors for X as well as the block vectors for the clusters. In more than 74% of all cases

Dataset num / dim / avg. nnz	Num clusters	Speed-up relative to k-means fast yinyang	 optimized k-means	 yinyang
usps 7291 / 256 / 223	100	**9.7**	8.9	7.1
	250	12.6	**13.4**	8.3
	1000	**18.3**	17.5	7.6
E2006 16087 / 150360 / 1241	100	**3.0**	2.9	2.6
	250	2.5	**4.1**	2
	1000	4.7	**7.7**	3.4
sector 6412 / 55197 / 163	100	1.9	**2.1**	1.9
	250	3.3	**3.9**	3
	1000	5.5	**7.5**	3.2
real-sim 72309 / 20958 / 51	100	**4.7**	2.4	4.6
	500	2.5	1.4	**2.8**
	5000	1.8	**2.9**	1.8
mediamill 30993 / 120 / 119	100	**13**	5.6	11.3
	500	**16.9**	11	14.7
	5000	**19.4**	14.4	8.3
caltech101 926860 / 128 / 77	100	14.6	2.4	**14.9**
	1000	27.4	3.8	**30.1**
	10000	15.8	7.4	**16**
kdd2010 510302 / 2014669 / 37	100	**4.3**	5.7	2
	1000	10.8	**13.3**	5.6
	10000	**22.9**	16.2	11.9
avira.201 161320 / 2384 / 418	100	**11.5**	4.6	10.3
	1000	**15.6**	10.6	12.5
	10000	21.4	**22.9**	12.2
mnist800k 810000 / 783 / 193	100	23.6	3.5	**23.8**
	1000	**30.3**	5.8	28.4
	10000	**36.5**	21.7	26.6

Table 2. Observing the speed-up factor with respect to standard k-means of *optimized k-means*, *yinyang*, and *fast yinyang* for various datasets and various values of cluster sizes k. *Fast yinyang* outperforms *yinyang* in over 74% of the cases.

fast_yinyang outperforms *yinyang* in terms of clustering duration, with a speedup of up to 2.4. In the cases where *yinyang* outperforms *fast_yinyang*, the maximum observed speedup over *fast_yinyang* is 1.12. Another observation is that *kmeans_optimized* is faster than *yinyang* in seven of the nine experiments with the highest value of k. This is one of the main results of this work, since apart from storing the block vectors, *kmeans_optimized* has no additional memory and maintenance cost with increasing k compared to the linear increasing costs of *yinyang* and *fast_yinyang*.

6. Conclusion

In this paper we demonstrated the effectiveness of a novel approach to approximate Euclidean distances based on block vectors, by integrating the resulting lower bounds into *yinyang* k-means, which lead to a significant speedup especially when using many cluster centers. The usefulness of the technique became even more obvious in the block vector enabled algorithm *kmeans_optimized*. By having a memory consumption not depending on k and even getting more efficient with high k, the technique fills the gap of clustering algorithms having the ability to cluster datasets into a large number of clusters efficiently. In a future work the influence of scaling and feature remapping on the approximation of Euclidean distances will be further studied.

Acknowledgements

Markus Kächele is supported by a scholarship of the Landesgraduiertenförderung Baden-Württemberg at Ulm University.

References

Alsabti, Khaled. An efficient k-means clustering algorithm. In *Proc. IPPS/SPDP Work. High Performance Data Mining*, 1998.

Arthur, D. and Vassilvitskii, S. K-means++: The advantages of careful seeding. In *Proc. 18th Ann. ACM-SIAM Symp. Discr. Alg. (SODA)*, pp. 1027–1035, 2007.

Bahmani, B., Moseley, B., Vattani, A., Kumar, R., and Vassilvitskii, S. Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622–633, 2012.

Dasgupta, S. The hardness of k-means clustering. Technical Report CS2007-0890, UC San Diego, 2007.

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 *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, pp. 579–587, 2015.

Drake, J. and Hamerly, G. Accelerated k-means with adaptive distance bounds. In *5th NIPS Work. Optim. Mach. Learn.*, pp. 579–587, 2012.

Elkan, C. Using the triangle inequality to accelerate k-means. In *Proc. 20th Int. Conf. Mach. Learn. (ICML)*, pp. 147–153, 2003.

Fahim, A.M., Salem, A.M., Torkey, F.A., and Ramadan, M.A. An efficient enhanced k-means clustering algorithm. *J. Zhejiang Univ. SCI. A*, 7(10):1626–1633, 2006.

Hamerly, G. Making k-means even faster. In *SIAM Int. Conf. Data Mining (SDM)*, pp. 130–140, 2010.

Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning*. Springer, 2001.

Horn, R.A. and Johnson, C.R. *Matrix Analysis*. Cambridge University Press, 1990.

Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C.D., R.Silverman, and Wu, A.Y. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Patt. Anal. Mach. Intell.*, 24:881–892, 2002.

Kaukoranta, T., Franti, P., and Nevalainen, O. A fast exact GLA based on code vector activity detection. *IEEE Trans. Imag. Proc.*, 9(8):1337–1342, 2000.

Lloyd, S. Least squares quantization in PCM. *IEEE Trans. Information Theory*, 28(2):129–137, 1982.

Low, Y. and Zheng, A.X. Fast top-k similarity queries via matrix compression. In *Proc. 21st ACM Int. Conf. Inf. Knowl. Manag.*, pp. 2070–2074, 2012.

Mahajan, M., Nimbhorkar, P., and Varadarajan, K. The planar k-means problem is NP-hard. In *Proc. 3rd Int. Work. Alg. Comput. (WALCOM)*, pp. 274–285, 2009.

Mielikainen, J. A novel full-search vector quantization algorithm based on the law of cosines. *IEEE Signal Proc. Letters*, 9(6):175–176, 2002.

Ostrovsky, R., Rabani, Y., Schulman, L. J., and Swamy, C. The effectiveness of Lloyd-type methods for the k-means problem. *J. ACM*, 59(6):28:1–28:22, 2013.

Pelleg, D. and Moore, A. Accelerating exact k-means algorithms with geometric reasoning. In *Proc. 5th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining (KDD)*, pp. 277–281, 1999.

Philbin, J., Chum, O., Isard, M., Sivic, J., and Zisserman, A. Object retrieval with large vocabularies and fast spatial matching. In *IEEE Conf. Comp. Vis. Patt. Recogn. (CVPR)*, 2007.

Sculley, D. Web-scale k-means clustering. In *Proc. 19th Int. Conf. World Wide Web*, pp. 1177–1178, 2010.

Steinbach, M., Karypis, G., and Kumar, V. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, 2000.

Torres, L. and Huguet, J. An improvement on codebook search for vector quantization. *IEEE Trans. Comm.*, 42 (2-4):208–210, 1994.

Vattani, A. The hardness of k-means clustering in the plane. Unpublished manuscript, 2009. URL `http://cseweb.ucsd.edu/~avattani/papers/kmeans_hardness.pdf`.

Wang, J., Wang, J., Ke, Q., Zeng, G., and Li, S. Fast approximate k-means via cluster closures. In *IEEE Conf. Comp. Vis. Patt. Recogn. (CVPR)*, pp. 3037–3044, 2012.

Wu, K.-S. and Lin, J.-C. Fast VQ encoding by an efficient kick-out condition. *IEEE Trans. Circ. Syst. Vid. Tech.*, 10(1):59–62, 2000.

Wu, X., Kumar, V., Quinlan, J.R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Yu, P.S., Zhou, Z.-H., Steinbach, M., Hand, D.J., and Steinberg, D. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, 2007.

Yeung, K.Y., Haynor, D.R., and Ruzzo, W.L. Validating clustering for gene expression data. *Bioinformatics*, 17 (4):309–318, 2001.