Accelerated k-means with adaptive distance bounds

Jonathan Drake
Department of Computer Science
Baylor University
Waco, TX 76798-7356
jonathan_drake@baylor.edu

Greg Hamerly
Department of Computer Science
Baylor University
Waco, TX 76798-7356
greg_hamerly@baylor.edu

Abstract

We propose an acceleration that computes the same answer as the standard k-means algorithm in substantially less time. Like Hamerly's [3] and Elkan's [2] accelerated algorithms, we avoid unnecessary distance calculations using distance bounds. For each point, Hamerly keeps one lower bound and Elkan keeps k, but we keep b lower bounds, where $1 < b < k$. Our algorithm adaptively selects and adjusts b at runtime for better performance. For $k \geq 50$, experiments show our algorithm is faster than other methods on datasets of about 20-120 dimensions.

1 Introduction

In the 1960s, Lloyd and others independently contributed ideas and algorithms that led to the formulation of k-means as a method of clustering multidimensional data [4]. Despite its age, k-means is ubiquitous and continues to have many and diverse applications. An unsupervised machine learning method, k-means is deployed in many areas to automatically cluster or summarize large amounts of data, e.g. in bioinformatics, astrophysics, vector quantization, and computer vision.

More formally, k-means is an optimization problem that seeks to partition data into clusters while minimizing distortion, which is defined as the sum of the squared distances between each data point and the center of the cluster it belongs to. Given a set of points x, a set of centers c, and binary flags $a(i, j)$ indicating point $x(i)$ is assigned to center $c(j)$, distortion is given by

$$J(x, c, a) := \sum_{i=1}^{n} \sum_{j=1}^{k} a(i, j)||x(i) - c(j)||_{\cdot}^{2} \tag{1}$$

The standard implementation of k-means clustering is an iterative process named Lloyd's algorithm [4]. Our goal is to make k-means faster, yet compute exactly the same answer as the standard algorithm. The primary expense in Lloyd's algorithm is repeated calculation of point-center distances, especially as dimension grows. Our proposed algorithm is a hybrid of two previous accelerated algorithms that retain additional geometric information to avoid unnecessary distance calculations.

2 Related work

There are many avenues of ongoing research into improving naive k-means, both in terms of speed and cluster quality, as discussed in [2, 3]. For lack of space, we omit a thorough discussion of other k-means work [6, 5, 1] in order to concentrate on Elkan's and Hamerly's algorithms.

In [2], Elkan demonstrates how to use the triangle inequality to significantly accelerate k-means by keeping track of 1 upper bound and k lower bounds for each data point x. The upper bound limits the distance from x to its assigned center, and the lower bounds correspond to the distance between

x and each center. By efficiently updating these bounds from one iteration to the next, we avoid having to calculate the explicit distance between a point and a center whenever the point's upper bound is less than its lower bound for that center. To proceed to the next iteration, we only need to know the membership of each cluster, not necessarily the exact point-center distances.

Hamerly's algorithm is a variant of Elkan's algorithm that keeps only 1 lower bound for each data point [3]. The semantics of the lower bound are also different. For a given point x, the single bound l now corresponds to the distance to x's *second*-closest center. This simplification avoids fewer distance calculations than Elkan's, but it greatly reduces the overhead of keeping bounds, and still gives the algorithm the opportunity to skip the innermost loop across all centers some 80% of the time [3]. For low-dimensional data, this strategy pays off well; however, it breaks down as dimension increases, driving up the relative cost of distance calculations. For $d \geq 50$, roughly, Elkan's algorithm regains superiority.

3 Methodology

Our proposed method leverages the complementary strengths of Elkan's and Hamerly's algorithms by keeping a *variable number* of lower bounds, which is automatically adjusted at runtime (see Section 3.2). For sufficiently large k, this approach achieves superior efficiency on medium-dimensional data. In particular, we keep the same upper bound on the distance to a each point's assigned center, but we track $1 < b < k$ lower bounds on the distance to its b next-closest centers, always ordered by increasing distance.

3.1 Proposed algorithm

The new algorithm achieves speedups by storing additional information beyond the requirements of Lloyd's algorithm. Given data points $x(i)$, $1 \leq i \leq n$, and initial centers $c(j)$, $1 \leq j \leq k$, the algorithm maintains the following structures, where $1 \leq z \leq b$:

$u(i)$	an upper bound on the distance between point $x(i)$ and its assigned center y,
$l(i,z)$	a lower bound on the distances between point $x(i)$ and its $(z+1)^{\text{th}}$ closest center,
$a(i,z)$	the index of the actual center corresponding to $l(i,z)$, and
$t(j)$	the distance that center $c(j)$ moved in the last iteration.

We give pseudocode for the new algorithm, omitting minor details and optimizations for clarity and brevity. Given a dataset x, initial centers c, and a number of lower bounds b to keep, the algorithm proceeds as described in Algorithm 1, which in turn calls the subroutines in Algorithms 2 and 3.

Algorithm 1 is similar to Hamerly's and Elkan's algorithm. In our innermost loop (lines 5-10), we are hoping the upper bound $u(i)$ satisfies one of our b lower bounds. If the z^{th} lower bound works, we only need to re-sort the first $(z+1)$ closest centers rather than the entire set c.

Algorithm 3 is more subtle. We pre-compute m (line 1) in order to update the outermost lower bound (line 4). Lower bounds shrink by the distance moved by their corresponding center (as in

Algorithm 1 K-MEANS(x, b, c)

```
 1: for i = 1 to |x| do
 2:     SORT-CENTERS(x(i), b, c)
 3: while not converged do
 4:     for i = 1 to |x| do
 5:         for z = 1 to b do
 6:             if u(i) ≤ l(i, z) then
 7:                 r ← {y, a(i, 1), ..., a(i, z)}
 8:                 SORT-CENTERS(x(i), z, r)
 9:                 skip to next iteration of for i loop
10:         SORT-CENTERS(x(i), b, c)
11:     move centers to centroid of assigned points, updating t
12:     UPDATE-BOUNDS()
```

Algorithm 2 SORT-CENTERS($x(i), q, r$)

1: sort r by increasing distance from $x(i)$
2: $y \leftarrow r(1)$
3: $u(i) \leftarrow ||x(i) - y||$
4: **for** $z = 1$ **to** q **do**
5: $\quad a(i, z) \leftarrow r(z + 1)$
6: $\quad l(i, z) \leftarrow ||x(i) - a(i, z)||$

Algorithm 3 UPDATE-BOUNDS()

1: $m \leftarrow \max_{1 \le j \le k} t(j)$
2: **for** $i = 1$ **to** $|x|$ **do**
3: $\quad u(i) \leftarrow u(i) + t(y)$
4: $\quad l(i, b) \leftarrow l(i, b) - m$
5: $\quad$ **for** $z = b - 1$ **to** 1 **do**
6: $\quad\quad l(i, z) \leftarrow l(i, z) - t(a(i, z))$
7: $\quad\quad$ **if** $l(i, z) > l(i, z + 1)$ **then**
8: $\quad\quad\quad l(i, z) = l(i, z + 1)$

Elkan's algorithm), but the outermost bound must also account for the movement of the $(k - b - 1)$ farthest centers not tracked by our algorithm. Computing this bound tightly would be expensive, so we settle for m, which we only need to compute once per iteration. Finally, we sacrifice additional tightness in order to force the bounds to stay in increasing order (lines 6-8).

3.2 Adaptive tuning of b

Keeping b bounds introduces another parameter to k-means. In order to make our algorithm more useful, we built an adaptive tuning mechanism. We generated several training datasets with uniform random distribution and found that runtime is generally good in the interval $\frac{k}{8} \le b \le \frac{k}{4}$.

Next, we observed that, for the majority of data points, the first lower bound is enough to avoid distance calculations (line 6 in Algorithm 1), the second lower bound is enough for the majority of any remaining points, and so on. This trend becomes stronger as the algorithm progresses, such that in later iterations, some of the outer bounds are *never* used. Figure 2 shows the fraction of a particular dataset for which each bound enables us to avoid distance calculations over time.

We use the following tuning strategy. Initially, $b = \frac{k}{4}$. After each iteration, we calculate the number of useful bounds, i.e. bounds that allowed the algorithm to avoid distance calculations, taking the maximum across all data points. We then reduce b to that number of bounds, subject to a minimum of $b = \frac{k}{8}$, saving on subsequent bound-maintenance overhead. Advantageously, this strategy allows us to proceed without knowing the optimal b in advance.

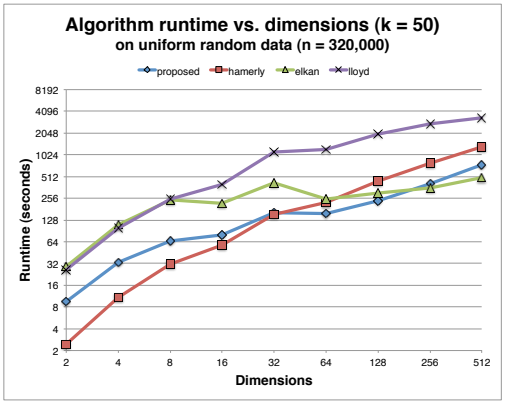
4 Results and analysis

We incur more overhead keeping a variable number of bounds than keeping just 1 or even all k. As $b \to k$, performance approximates that of Elkan's algorithm. As $b \to 1$, performance approaches that of Hamerly's. For properly tuned b, our algorithm beats at least one of its parents in all experiments. However, at some point, the benefits of avoiding distance calculations begin to outweigh the cost of keeping distance bounds. In medium dimension, our hybrid outperforms *both* of its parents.

We ran a battery of experiments for $k = 25, 50, 100, 200, 400$ on a 4-core 3GHz Xeon machine with 8GB memory. Uniform random data is roughly worst-case for k-means: there is no underlying structure to learn, and bounds are less effective when data is not well-clustered. However, it provides a consistent baseline for comparing performance, which we then confirm on real-world datasets.

Figure 1 shows the runtime of our proposed algorithm alongside Elkan's, Hamerly's, and Lloyd's on uniform random data for $k = 50$ and $k = 200$, with varying dimension, plotted on logarithmic axes. As expected, Hamerly's has the best runtime in low dimension and Elkan's in high dimension. For $k = 50$, our algorithm is superior for about $30 \le d \le 130$, which shifts to about $20 \le d \le 120$ for the larger $k = 200$ case.

For consistency, we tested the same real-world datasets as Hamerly and Elkan: `mnist50`, `mnist784`, `kddcup98`, `birchDS1`, `covtype` [2, 3]. Figure 3 compares algorithm runtime on each dataset for $k = 200$. Our algorithm outperforms others on all datasets except `birchDS1` ($d = 2$) and `mnist784` ($d = 784$), which are won by Hamerly's and Elkan's algorithms, respectively, as expected based on their dimensionality.



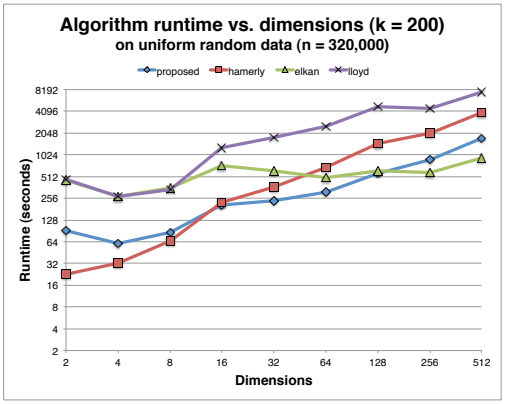


Figure 1: Comparative algorithm runtime vs. dimension on uniform random data, $k = 50$; $k = 200$.

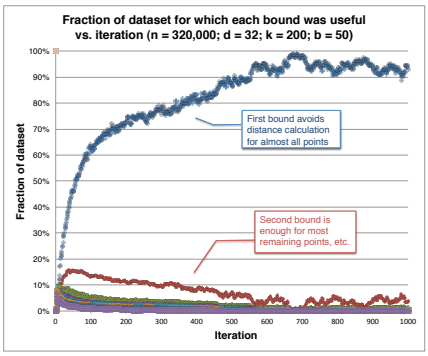


Figure 2: Efficiency of bounds, $k = 200$

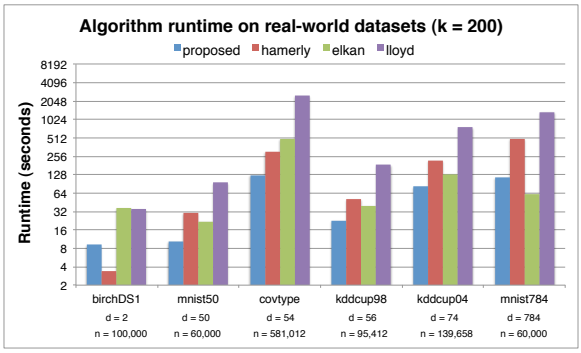


Figure 3: Runtime on real-world datasets, $k = 200$.

5 Conclusion

Elkan's algorithm performs well for high-dimensional datasets, and Hamerly's algorithm dominates low-dimensional spaces [3]. Our algorithm successfully blends these two algorithms for better efficiency by keeping an adaptive number of lower bounds, and it achieves this goal in medium-dimensional spaces. Notably, the strengths of these three algorithms are complementary.

Ideally, we would like a k-means algorithm performing well for different values of n, k, and d. Lacking this universal solution, we can construct a piecewise algorithm using Hamerly's algorithm for $d < 20$, our algorithm for $20 \leq d < 120$, and Elkan's algorithm for $d \geq 120$. Meanwhile, we seek a deeper theoretical understanding of the clustering problem that will enable us to write an algorithm that performs optimally under all conditions.

References

[1] David Arthur and Sergei Vassilvitskii. kmeans++: The advantages of careful seeding. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, 2007.

[2] Charles Elkan. Using the triangle inequality to accelerate k-means. In *ICML*, pages 147–153, 2003.

[3] Greg Hamerly. Making k-means even faster. In *SIAM International Conference on Data Mining*, 2010.

[4] Stuart Lloyd. Least squares quantization in PCM. In *IEEE Transactions Information Theory*, 28, pages 129–137, 1982.

[5] Andrew Moore. The anchors hierarchy: Using the triangle inequality to survive high-dimensional data. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 397–405. AAAI Press, 2000.

[6] Dan Pelleg and Andrew Moore. Accelerating exact k-means algorithms with geometric reasoning. In *ACM SIGKDD Fifth International Conference on Knowledge Discovery and Data Mining*, pages 277–281, 1999.