Using the Triangle Inequality to Accelerate K-Means

Article · July 2003

Source: CiteSeer

1 author:

Charles Elkan
University of California, San Diego
158 PUBLICATIONS **24,156** CITATIONS

SEE PROFILE

Using the Triangle Inequality to Accelerate k-Means

Charles Elkan ELKAN@CS.UCSD.EDU
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California 92093-0114

Abstract

The k-means algorithm is by far the most widely used method for discovering clusters in data. We show how to accelerate it dramatically, while still always computing exactly the same result as the standard algorithm. The accelerated algorithm avoids unnecessary distance calculations by applying the triangle inequality in two different ways, and by keeping track of lower and upper bounds for distances between points and centers. Experiments show that the new algorithm is effective for datasets with up to 1000 dimensions, and becomes more and more effective as the number k of clusters increases. For $k \geq 20$ it is many times faster than the best previously known accelerated k-means method.

1. Introduction

The most common method for finding clusters in data used in applications is the algorithm known as k-means. k-means is considered a fast method because it is not based on computing the distances between all pairs of data points. However, the algorithm is still slow in practice for large datasets. The number of distance computations is nke where n is the number of data points, k is the number of clusters to be found, and e is the number of iterations required. Empirically, e grows sublinearly with k, n, and the dimensionality d of the data.

The main contribution of this paper is an optimized version of the standard k-means method, with which the number of distance computations is in practice closer to n than to nke.

The optimized algorithm is based on the fact that most distance calculations in standard k-means are redundant. If a point is far away from a center, it is not necessary to calculate the exact distance between the point and the center in order to know that the point should not be assigned to this center. Conversely, if a point is much closer to one center than to any other, calculating exact distances is not necessary to know that the point should be assigned to the first center. We show below how to make these intuitions concrete.

We want the accelerated k-means algorithm to be usable wherever the standard algorithm is used. Therefore, we need the accelerated algorithm to satisfy three properties. First, it should be able to start with any initial centers, so that all existing initialization methods can continue to be used. Second, given the same initial centers, it should always produce exactly the same final centers as the standard algorithm. Third, it should be able to use any black-box distance metric, so it should not rely for example on optimizations specific to Euclidean distance.

Our algorithm in fact satisfies a condition stronger than the second one above: after each iteration, it produces the same set of center locations as the standard k-means method. This stronger property means that heuristics for merging or splitting centers (and for dealing with empty clusters) can be used together with the new algorithm. The third condition is important because many applications use a domain-specific distance metric. For example, clustering to identify duplicate alphanumeric records is sometimes based on alphanumeric edit distance (Monge & Elkan, 1996), while clustering of protein structures is often based on an expensive distance function that first rotates and translates structures to superimpose them. Even without a domain-specific metric, recent work shows that using a non-traditional L_p norm with $0 < p < 1$ is beneficial when clustering in a high-dimensional space (Aggarwal et al., 2001).

This paper is organized as follows. Section 2 explains how to use the triangle inequality to avoid redundant distance calculations. Then Section 3 presents the new algorithm, and Section 4 discusses experimental results on six datasets of dimensionality 2 to 1000. Section 5 outlines possible improvements to the method, while Section 6 reviews related work, and Section 7 explains three open research issues.

Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

2. Applying the triangle inequality

Our approach to accelerating k-means is based on the triangle inequality: for any three points x, y, and z, $d(x,z) \leq d(x,y)+d(y,z)$. This is the only "black box" property that all distance metrics d possess.

The difficulty is that the triangle inequality gives upper bounds, but we need lower bounds to avoid calculations. Let x be a point and let b and c be centers; we need to know that $d(x,c) \geq d(x,b)$ in order to avoid calculating the actual value of $d(x,c)$.

The following two lemmas show how to use the triangle inequality to obtain useful lower bounds.

Lemma 1: Let x be a point and let b and c be centers. If $d(b,c) \geq 2d(x,b)$ then $d(x,c) \geq d(x,b)$.

Proof: We know that $d(b,c) \leq d(b,x) + d(x,c)$. So $d(b,c) - d(x,b) \leq d(x,c)$. Consider the left-hand side: $d(b,c) - d(x,b) \geq 2d(x,b) - d(x,b) = d(x,b)$. So $d(x,b) \leq d(x,c)$. ■

Lemma 2: Let x be a point and let b and c be centers. Then $d(x,c) \geq \max\{0, d(x,b) - d(b,c)\}$.

Proof: We know that $d(x,b) \leq d(x,c) + d(b,c)$, so $d(x,c) \geq d(x,b) - d(b,c)$. Also, $d(x,c) \geq 0$. ■

Note that Lemmas 1 and 2 are true for any three points, not just for a point and two centers, and the statement of Lemma 2 can be strengthened in various ways.

We use Lemma 1 as follows. Let x be any data point, let c be the center to which x is currently assigned, and let c' be any other center. The lemma says that if $\frac{1}{2}d(c,c') \geq d(x,c)$, then $d(x,c') \geq d(x,c)$. In this case, it is not necessary to calculate $d(x,c')$.

Suppose that we do not know $d(x,c)$ exactly, but we do know an upper bound u such that $u \geq d(x,c)$. Then we need to compute $d(x,c')$ and $d(x,c)$ only if $u > \frac{1}{2}d(c,c')$.

If $u \leq \frac{1}{2}\min d(c,c')$ where the minimum is over all $c' \neq c$, then the point x must remain assigned to the center c, and all distance calculations for x can be avoided.

Lemma 2 is applied as follows. Let x be any data point, let b be any center, and let b' be the previous version of the same center. (That is, suppose the centers are numbered 1 through k, and b is center number j; then b' is center number j in the previous iteration.) Suppose that in the previous iteration we knew a lower bound l' such that $d(x,b') \geq l'$. Then we can infer a lower bound l for the current iteration:

$$\begin{aligned} d(x,b) &\geq \max\{0, d(x,b') - d(b,b')\} \\ &\geq \max\{0, l' - d(b,b')\} = l. \end{aligned}$$

Informally, if l' is a good approximation to the previous distance between x and the jth center, and this center has moved only a small distance, then l is a good approximation to the updated distance.

The algorithm below is the first k-means variant that uses lower bounds, as far as we know. It is also the first algorithm that carries over varying information from one k-means iteration to the next. According to the authors of (Kanungo et al., 2000): "The most obvious source of inefficiency in [our] algorithm is that it passes no information from one stage to the next. Presumably in the later stages of Lloyd's algorithm, as the centers are converging to their final positions, one would expect that the vast majority of the data points have the same closest center from one stage to the next. A good algorithm would exploit this coherence to improve running time." The algorithm in this paper achieves this goal. One previous algorithm also reuses information from one k-means iteration in the next, but that method, due to (Judd et al., 1998), does not carry over lower or upper bounds.

Suppose $u(x) \geq d(x,c)$ is an upper bound on the distance between x and the center c to which x is currently assigned, and suppose $l(x,c') \leq d(x,c')$ is a lower bound on the distance between x and some other center c'. If $u(x) \leq l(x,c')$ then $d(x,c) \leq u(x) \leq l(x,c') \leq d(x,c')$ so it is necessary to calculate neither $d(x,c)$ nor $d(x,c')$. Note that it will never be necessary in this iteration of the accelerated method to compute $d(x,c')$, but it may be necessary to compute $d(x,c)$ exactly because of some other center c'' for which $u(x) \leq l(x,c'')$ is not true.

3. The new algorithm

Putting the observations above together, the accelerated k-means algorithm is as follows.

First, pick initial centers. Set the lower bound $l(x,c) = 0$ for each point x and center c. Assign each x to its closest initial center $c(x) = \text{argmin}_c\, d(x,c)$, using Lemma 1 to avoid redundant distance calculations. Each time $d(x,c)$ is computed, set $l(x,c) = d(x,c)$. Assign upper bounds $u(x) = \min_c d(x,c)$.

Next, repeat until convergence:

1. For all centers c and c', compute $d(c,c')$. For all centers c, compute $s(c) = \frac{1}{2}\min_{c' \neq c} d(c,c')$.
2. Identify all points x such that $u(x) \leq s(c(x))$.
3. For all remaining points x and centers c such that
   (i) $c \neq c(x)$ and
   (ii) $u(x) > l(x,c)$ and
   (iii) $u(x) > \frac{1}{2}d(c(x),c)$:

3a. If $r(x)$ then compute $d(x, c(x))$ and assign $r(x) =$ false. Otherwise, $d(x, c(x)) = u(x)$.

3b. If $d(x, c(x)) > l(x, c)$
or $d(x, c(x)) > \frac{1}{2} d(c(x), c)$ then
Compute $d(x, c)$
If $d(x, c) < d(x, c(x))$ then assign $c(x) = c$.

4. For each center c, let $m(c)$ be the mean of the points assigned to c.

5. For each point x and center c, assign
$$l(x, c) = \max\{l(x, c) - d(c, m(c)), 0\}.$$

6. For each point x, assign
$$u(x) = u(x) + d(m(c(x)), c(x))$$
$$r(x) = \text{true}.$$

7. Replace each center c by $m(c)$.

In step (3), each time $d(x, c)$ is calculated for any x and c, its lower bound is updated by assigning $l(x, c) = d(x, c)$. Similarly, $u(x)$ is updated whenever $c(x)$ is changed or $d(x, c(x))$ is computed. In step (3a), if $r(x)$ is true then $u(x)$ is out-of-date, i.e. it is possible that $u(x) \neq d(x, c(x))$. Otherwise, computing $d(x, c(x))$ is not necessary. Step (3b) repeats the checks from (ii) and (iii) in order to avoid computing $d(x, c)$ if possible.

The fundamental reason why the algorithm above is effective in reducing the number of distance calculations is that at the start of each iteration, the upper bounds $u(x)$ and the lower bounds $l(x, c)$ are tight for most points x and centers c. If these bounds are tight at the start of one iteration, the updated bounds tend to be tight at the start of the next iteration, because the location of most centers changes only slightly, and hence the bounds change only slightly.

The initialization step of the algorithm assigns each point to its closest center immediately. This requires relatively many distance calculations, but it leads to exact upper bounds $u(x)$ for all x and to exact lower bounds $l(x, c)$ for many (x, c) pairs. An alternative initialization method is to start with each point arbitrarily assigned to one center. The initial values of $u(x)$ and $l(c, x)$ are then based on distances calculated to this center only. With this approach, the initial number of distance calculations is only n, but $u(x)$ and $l(c, x)$ are much less tight initially, so more distance calculations are required later. (After each iteration each point is always assigned correctly to its closest center, regardless of how inaccurate the lower and upper bounds are at the start of the iteration.) Informal experiments suggest that both initialization methods lead to about the same total number of distance calculations.

Logically, step (2) is redundant because its effect is achieved by condition (iii). Computationally, step (2) is beneficial because if it eliminates a point x from further consideration, then comparing $u(x)$ to $l(x, c)$ for every c separately is not necessary. Condition (iii) inside step (3) is beneficial despite step (2), because $u(x)$ and $c(x)$ may change during the execution of step (3).

We have implemented the algorithm above in Matlab. When step (3) is implemented with nested loops, the outer loop can be over x or over c. For efficiency in Matlab and similar languages, the outer loop should be over c since $k \ll n$ typically, and the inner loop should be replaced by vectorized code that operates on all relevant x collectively.

Step 4 computes the new location of each cluster center c. Setting $m(c)$ to be the mean of the points assigned to c is appropriate when the distance metric in use is Euclidean distance. Otherwise, $m(c)$ may be defined differently. For example, with k-medians the new center of each cluster is a representative member of the cluster.

4. Experimental results

This section reports the results of running the new algorithm on six large datasets, five of which are high-dimensional. The datasets are described in Table 1, while Table 2 gives the results.

Our experimental design is similar to the design of (Moore, 2000), which is the best recent paper on speeding up the k-means algorithm for high-dimensional data. However, there is only one dataset used in (Moore, 2000) for which the raw data are available and enough information is given to allow the dataset to be reconstructed. This dataset is called "covtype." Therefore, we also use five other publicly available datasets. None of the datasets have missing data.

In order to make our results easier to reproduce, we use a fixed initialization for each dataset X. The first center is initialized to be the mean of X. Subsequent centers are initialized according to the "furthest first" heuristic: each new center is $\text{argmax}_{x \in X} \min_{c \in C} d(x, c)$ where C is the set of initial centers chosen so far (Dasgupta, 2002).

Following the practice of past research, we measure the performance of an algorithm on a dataset as the number of distance calculations required. All algorithms that accelerate k-means incur overhead to create and update auxiliary data structures. This means that speedup compared to k-means is always less in clock time than in number of distance calculations. Our algorithm reduces the number of distance calculations so dramatically that its overhead time is often greater than the time spent on distance calculations. However, the total execution time is always much less than the time required by standard k-means. The overhead of the l and u data structures will be much smaller with a C implementation than with the Matlab implementation used

name	cardinality	dimensionality	description
birch	100000	2	10 by 10 grid of Gaussian clusters, DS1 in (Zhang et al., 1996)
covtype	150000	54	remote soil cover measurements, after (Moore, 2000)
kddcup	95413	56	KDD Cup 1998 data, un-normalized
mnist50	60000	50	random projection of NIST handwritten digit training data
mnist784	60000	784	original NIST handwritten digit training data
random	10000	1000	uniform random data

Table 1. Datasets used in experiments.

		$k = 3$	$k = 20$	$k = 100$
birch	iterations	17	38	56
	standard	5.100e+06	7.600e+07	5.600e+08
	fast	4.495e+05	1.085e+06	1.597e+06
	speedup	11.3	70.0	351
covtype	iterations	18	256	152
	standard	8.100e+06	7.680e+08	2.280e+09
	fast	9.416e+05	7.147e+06	7.353e+06
	speedup	8.60	107	310
kddcup	iterations	34	100	325
	standard	9.732e+06	1.908e+08	3.101e+09
	fast	6.179e+05	3.812e+06	1.005e+07
	speedup	15.4	50.1	309
mnist50	iterations	38	178	217
	standard	6.840e+06	2.136e+08	1.302e+09
	fast	1.573e+06	9.353e+06	3.159e+07
	speedup	4.35	22.8	41.2
mnist784	iterations	63	60	165
	standard	1.134e+07	7.200e+07	9.900e+08
	fast	1.625e+06	7.396e+06	3.055e+07
	speedup	6.98	9.73	32.4
random	iterations	52	33	18
	standard	1.560e+06	6.600e+06	1.800e+07
	fast	1.040e+06	3.020e+06	5.348e+06
	speedup	1.50	2.19	3.37

Table 2. Rows labeled "standard" and "fast" give the number of distance calculations performed by the unaccelerated k-means algorithm and by the new algorithm. Rows labeled "speedup" show how many times faster the new algorithm is, when the unit of measurement is distance calculations.

for the experiments reported here. For this reason, clock times are not reported.

Perhaps the most striking observation to be made from Table 2 is that the relative advantage of the new method increases with k. The number of distance calculations grows only slowly with k and with e (the number of passes over the data, called "iterations" in Table 2). So much redundant computation is eliminated that the total number of distance calculations is closer to n than to nke as for standard k-means.

A related observation is that for $k \geq 20$ we obtain a much better speedup than with the anchors method (Moore, 2000). The speedups reported by Moore for the "covtype" dataset are 24.8, 11.3, and 19.0 respectively for clustering with 3, 20, and 100 centers. The speedups we obtain are 8.60, 107, and 310. We conjecture that the improved speedup for $k \geq 20$ arises in part from using the actual cluster centers as adaptive "anchors," instead of using a set of anchors fixed in preprocessing. The worse speedup for $k = 3$ remains to be explained.

Another striking observation is that the new method remains effective even for data with very high dimensionality. Moore writes "If there *is* no underlying structure in the data (e.g. if it is uniformly distributed) there will be little or no acceleration in high dimensions no matter what we do. This gloomy view, supported by recent theoretical work in computational geometry (Indyk et al., 1999), means that we can only accelerate datasets that have interesting internal structure." While this claim is almost certainly true asymptotically as the dimension of a dataset tends to infinity, our results on the "random" dataset suggest that worthwhile speedup can still be obtained up to at least 1000 dimensions. As expected, the more clustered a dataset is, the greater the speedup obtained. Random projection makes clusters more Gaussian (Dasgupta, 2000), so speedup is better for the "mnist50" dataset than for the "mnist784" dataset.

5. Limitations and extensions

During each iteration of the algorithm proposed here, the lower bounds $l(x, c)$ are updated for all points x and centers c. These updates take $O(nk)$ time, so the time complexity of the algorithm remains at least $O(nke)$ even though the number of distance calculations is roughly $O(n)$ only. It may be possible to avoid updating many lower bounds in most iterations, and hence to reduce the nominal complexity of the algorithm. Note that if a point x is eliminated from further consideration in step (2), then $l(x, c)$ is not used at all.

In some clustering applications, $k \gg d$. This is the case in particular for vector quantization for image compression. For these applications the memory required to store the lower bounds $l(x, c)$ may be the dominant storage cost. However, the entire matrix $l(x, c)$ never needs to be kept in main memory. If the data are streamed into memory at each iteration from disk, then the $l(x, c)$ matrix can also be streamed into memory in synchrony.

Moreover, the algorithm remains beneficial even if lower bounds are not used, so condition (ii) becomes $u(x) > d(x, c)$, where $d(x, c)$ is computed if necessary.

When the algorithm is used with a distance function that is fast to evaluate, such as an L_p norm, then in practice the time complexity of the algorithm is dominated by the bookkeeping used to avoid distance calculations. Therefore, future work should focus on reducing this overhead.

The point above is especially true because a Euclidean distance (or other L_p distance) in d dimensions can often be compared to a known minimum distance in $o(d)$ time. The simple idea is to stop evaluating the new squared distance when the sum of squares so far is greater than the known squared minimum distance. (This suggestion is usually ascribed to (Bei & Gray, 1985), but in fact it is first mentioned in (Cheng et al., 1984).) Distance calculations can be stopped even quicker if axes of high variation are considered first. Axes of maximum variation may be found by principal component analysis (PCA) (McNames, 2000), but the preprocessing cost of PCA may be prohibitive.

At the end of each iteration, centers must be recomputed. Computing means takes $O(nd)$ time independent of k. This can be reduced to $O((k + b)d)$ time where b is the number of points assigned to a different center during the iteration. Typically $b \ll n$ in all except the first few iterations. As mentioned above, the algorithm of this paper can also be used when centers are not recomputed as means.

During each iteration, distances between all centers must be recomputed, so the minimum number of distance computations per iteration is $k(k - 1)/2$. For large k, as in vector quantization, this may be a dominant expense. Future research should investigate the best way to reduce this cost by computing approximations for inter-center distances that are large.

6. Related work

Many papers have been published on the topic of accelerating the k-means algorithm, in several different research communities. Some of the most important of these papers are described briefly in this section. Most of the papers cited below only cite previous papers from the same research community, so one of the contributions of this paper is an attempt to collect references that otherwise cannot be found in one place. All the relevant papers that we know

of can be found by following chains of citations from the papers mentioned here.

A version of the k-means algorithm was first published by (MacQueen, 1965). The history of different variants of the algorithm is discussed by (Faber, 1994). The basic algorithm used most commonly today, and used in this paper, where centers are recomputed once after each pass through the data, is usually attributed to a paper written by Lloyd in 1957 but not published until 1982 (Lloyd, 1982). However, that paper only discusses quantization (i.e. clustering) for some special one-dimensional cases.

The central operation in the k-means algorithm is to find the nearest center for each data point. At least three general approaches have been developed for accelerating this operation.

One general approach is based on locality-sensitive hashing (Indyk & Motwani, 1998), but these methods are not well-suited for finding exact nearest neighbors. A second general approach organizes points into trees where nearby points are in the same subtree. Approaches using kd-trees or similar have been proposed independently by several authors (Ramasubramanian & Paliwal, 1990; Deng & Moore, 1993; Pelleg & Moore, 1999; Alsabti et al., 1998; Kanungo et al., 2000), but these methods are not effective for $d > 10$ about. By using metric trees Moore's "anchors" method is effective for much larger d (Moore, 2000).

The third general approach to the nearest neighbor task is to use triangle inequalities to eliminate unnecessary distance calculations. Using Lemma 1 above appears to have been proposed first by (Hodgson, 1988), then again independently by (Orchard, 1991; Montolio et al., 1992; Phillips, 2002) among others. Our application of Lemma 1 is more fine-grained than previous applications. The lemma says that if $d(x, c) < \frac{1}{2}d(c, c')$, then $d(x, c) < d(x, c')$. The algorithm of (Hodgson, 1988) only considers the center c' that is closest to c. If $d(x, c) < d(c, c')/2$ for this c' then x remains assigned to c. Otherwise, no distance calculations are eliminated. Our algorithm applies the lemma for every center different from c, so for most x some distance calculations are avoided, even if others must be performed.

Variants of Lemma 2 have been used by many authors, starting with (Burkhard & Keller, 1973; Vidal, 1986), but using the lemma to update a lower bound on the distance between moving points appears to be novel.

The triangle inequality applies to all distance metrics. Many papers have also been published on speeding up k-means or nearest neighbor search using inequalities that are specific for Euclidean distance, for example (Wu & Lin, 2000; Mielikainen, 2002).

Many papers have been published on on approximating k-means quickly; well-known papers include (Zhang et al., 1996; Farnstrom et al., 2000). However, the exact algorithm presented here is so fast that it is not clear when an approximate algorithm is necessary.

7. Open issues

A basic open theoretical question is whether one can find a lower bound on how many distance calculations are needed by any implementation of exact k-means. Can one construct an adversary argument showing that if any algorithm omits certain distance computations, then an opponent can choose values for these distances that, together with all other distances, satisfy the triangle inequality, yet also make the output of the algorithm incorrect?

Perhaps the most fundamental practical question for future work is how to find better clusterings, i.e. better local optima. Now that we can run k-means fast, how can we use additional computation to get answers of better quality?

One common approach to finding better local optima is to run k-means with many different initializations. The algorithm above allows many more initializations to be tried in the same total time. Another widespread heuristic for finding better clusterings is to run k-means with a large value for k, and then to merge or prune the clusters obtained into a good clustering with smaller k. Since our algorithm makes the running time of k-means sublinear in k, it is especially useful for this approach.

A third important open question is how to accelerate clustering methods that use soft assignment of points to centers. Two important methods in this class are Gaussian expectation-maximization (EM) (Dempster et al., 1977) and harmonic k-means (Hamerly & Elkan, 2002). In these methods each center is recomputed as the weighted average of all points, where weights are related to distances. Can triangle inequalities (or other inequalities!) be applied to obtain upper bounds on weights that are close to zero, and hence to obtain approximate soft assignment solutions quickly?

Acknowledgments

Thanks to Sanjoy Dasgupta, Ari Frank, Greg Hamerly, Doug Turnbull, and others for providing useful comments and datasets.

References

Aggarwal, C. C., Hinneburg, A., & Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional spaces. *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings* (pp. 420–434). Springer.

Alsabti, K., Ranka, S., & Singh, V. (1998). An efficient *k*-means clustering algorithm. *IPPS/SPDP Workshop on High Performance Data Mining*. IEEE Computer Society Press.

Bei, C.-D., & Gray, R. M. (1985). An improvement of the minimum distortion encoding algorithm for vector quantization. *IEEE Transactions on Communications*, *33*, 1132–1133.

Burkhard, W. A., & Keller, R. M. (1973). Some approaches to best-match file searching. *Communications of the ACM*, *16*, 230–236.

Cheng, D.-Y., Gersho, A., Ramamurthi, B., & Shoham, Y. (1984). Fast search algorithms for vector quantization and pattern matching. *International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 9.11.1–9.11.4). IEEE Computer Society Press.

Dasgupta, S. (2000). Experiments with random projection. *Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI'00)* (pp. 143–151). Morgan Kaufmann.

Dasgupta, S. (2002). Performance guarantees for hierarchical clustering. *Fifteenth Annual Conference on Computational Learning Theory (COLT'02)* (pp. 351–363). Springer Verlag.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, *39*, 1–38.

Deng, K., & Moore, A. W. (1993). Multiresolution instance-based learning. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1233–1239). San Francisco: Morgan Kaufmann.

Faber, V. (1994). Clustering and the continuous k-means algorithm. *Los Alamos Science*, 138–144.

Farnstrom, F., Lewis, J., & Elkan, C. (2000). Scalability for clustering algorithms revisited. *ACM SIGKDD Explorations*, *2*, 51–57.

Hamerly, G., & Elkan, C. (2002). Alternatives to the *k*-means algorithm that find better clusterings. *Proceedings of the Eleventh International Conference on Information and Knowledge Management* (pp. 600–607). McLean, Virginia, USA: ACM Press.

Hodgson, M. E. (1988). Reducing computational requirements of the minimum-distance classifier. *Remote Sensing of Environments*, *25*, 117–128.

Indyk, P., Amir, A., Efrat, A., & Samet, H. (1999). Efficient algorithms and regular data structures for dilation, location and proximity problems. *Proceedings of the Annual Symposium on Foundations of Computer Science* (pp. 160–170).

Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. *Proceedings of the Annual ACM Symposium on the Theory of Computing* (pp. 604–613).

Judd, D., McKinley, P. K., & Jain, A. K. (1998). Large-scale parallel data clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*, 871–876.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., & Wu, A. Y. (2000). The analysis of a simple *k*-means clustering algorithm. *ACM Symposium on Computational Geometry* (pp. 100–109). ACM Press.

Lloyd, S. P. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, *28*, 129–137.

MacQueen, J. B. (1965). On convergence of *k*-means and partitions with minimum average variance. *Annals of Mathematical Statistics*, *36*, 1084. Abstract only.

McNames, J. (2000). Rotated partial distance search for faster vector quantization encoding. *IEEE Signal Processing Letters*, *7*, 244–246.

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

Monge, A. E., & Elkan, C. P. (1996). The field matching problem: Algorithms and applications. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (pp. 267–270). Portland, Oregon: AAAI Press (distributed by MIT Press).

Montolio, P., Gasull, A., Monte, E., Torres, L., & Marques, F. (1992). Analysis and optimization of the k-means algorithm for remote sensing applications. In A. Sanfeliu (Ed.), *Pattern recognition and image analysis*, 155–170. World Scientific.

Moore, A. W. (2000). The anchors hierarchy: Using the triangle inequality to survive high dimensional data. *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence* (pp. 397–405). Morgan Kaufmann.

Orchard, M. T. (1991). A fast nearest-neighbor search algorithm. *International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 2297–2300). IEEE Computer Society Press.

Pelleg, D., & Moore, A. (1999). Accelerating exact k-means algorithms with geometric reasoning. *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99)* (pp. 277–281).

Phillips, S. J. (2002). Acceleration of *k*-means and related clustering algorithms. *Fourth International Workshop on Algorithm Engineering and Experiments (ALENEX)* (pp. 166–177). Springer Verlag.

Ramasubramanian, V., & Paliwal, K. K. (1990). A generalized optimization of the $k-d$ tree for fast nearest-neighbour search. *Fourth IEEE Region 10 International Conference (TENCON'89)* (pp. 565–568). IEEE Computer Society Press.

Vidal, E. (1986). An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognition Letters*, *4*, 145–157.

Wu, K.-S., & Lin, J.-C. (2000). Fast VQ encoding by an efficient kick-out condition. *IEEE Transactions on Circuits and Systems for Video Technology*, *10*, 59–62.

Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: an efficient data clustering method for very large databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 103–114). ACM Press.