An Accelerated Nearest Neighbor Search Method for the K-Means Clustering Algorithm

Adam Fausett and **M. Emre Celebi**
Department of Computer Science,
Louisiana State University in Shreveport,
Shreveport, LA, USA
ecelebi@lsus.edu

Abstract

K-means is undoubtedly the most widely used partitional clustering algorithm. Unfortunately, the nearest neighbor search step of this algorithm can be computationally expensive, as the distance between each input vector and all cluster centers need to be calculated. To accelerate this step, a computationally inexpensive distance estimation method can be tried first, resulting in the rejection of candidate centers that cannot possibly be the nearest center to the input vector under consideration. This way, the computational requirements of the search can be reduced as most of the full distance computations become unnecessary. In this paper, a fast nearest neighbor search method that rejects impossible centers to accelerate the k-means clustering algorithm is presented. Our method uses geometrical relations among the input vectors and the cluster centers to reject many unlikely centers that are not typically rejected by similar approaches. Experimental results show that the method can reduce the number of distance computations significantly without degrading the clustering accuracy.

1 Introduction

Clustering, the unsupervised classification of patterns into groups, is one of the most important tasks in exploratory data analysis (Jain, Murty, and Flynn 1999). Primary goals of clustering include gaining insight into data (detecting anomalies, identifying salient features, etc.), classifying data, and compressing data. Clustering has a long and rich history in a variety of scientific disciplines including anthropology, biology, medicine, psychology, statistics, mathematics, engineering, and computer science. As a result, numerous clustering algorithms have been proposed since the early 1950s (Jain 2010).

Clustering algorithms can be broadly classified into two groups: hierarchical and partitional (Jain 2010). Hierarchical algorithms recursively find nested clusters either in a top-down (divisive) or bottom-up (agglomerative) fashion. In contrast, partitional algorithms find all the clusters simultaneously as a partition of the data and do not impose a hierarchical structure. Most hierarchical algorithms have quadratic or higher complexity in the number of data points (Jain, Murty, and Flynn 1999) and therefore are not suitable for large data sets, whereas partitional algorithms often have lower complexity.

Given a data set $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$, i.e., N vectors each with D attributes, hard partitional algorithms divide $\mathcal{X}$ into K exhaustive and mutually exclusive clusters $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_K\}, \quad \bigcup_{i=1}^{K} \mathcal{P}_i = \mathcal{X}, \quad \mathcal{P}_i \cap \mathcal{P}_j = \varnothing$ for $1 \leq i \neq j \leq K$. These algorithms usually generate clusters by optimizing a criterion function. The most intuitive and frequently used criterion function is the Sum of Squared Error (SSE) given by:

$$\text{SSE} = \sum_{i=1}^{K} \sum_{\mathbf{x}_j \in \mathcal{P}_i} \|\mathbf{x}_j - \mathbf{y}_i\|_2^2 \tag{1}$$

where $\|.\|_2$ denotes the Euclidean norm and $\mathbf{y}_i = 1/|\mathcal{P}_i| \sum_{\mathbf{x}_j \in \mathcal{P}_i} \mathbf{x}_j$ is the centroid of cluster $\mathcal{P}_i$ whose cardinality is $|\mathcal{P}_i|$.

The number of ways in which a set of N objects can be partitioned into K non-empty groups is given by Stirling numbers of the second kind:

$$\mathcal{S}(N, K) = \frac{1}{K!} \sum_{i=0}^{K} (-1)^{K-i} \binom{K}{i} i^N \tag{2}$$

which can be approximated by $K^N/K!$ It can be seen that a complete enumeration of all possible clusterings to determine the global minimum of (1) is clearly computationally prohibitive. In fact, this non-convex optimization problem is proven to be NP-hard even for $K = 2$ (Aloise et al. 2009) or $D = 2$ (Mahajan, Nimbhorkar, and Varadarajan 2012). Consequently, various heuristics have been developed to provide approximate solutions to this problem (Tarsitano 2003). Among these heuristics, Lloyd's algorithm (Lloyd 1982), often referred to as the k-means algorithm, is the simplest and most commonly used one (Celebi and Kingravi 2012; Celebi, Kingravi, and Vela 2013). This algorithm starts with K arbitrary centers, typically chosen uniformly at random from the data points. Each point is assigned to the nearest center and then each center is recalculated as the mean of all points assigned to it. These two steps are repeated until a predefined termination criterion is met. Algo. 1 shows the pseudocode of this procedure.

Despite its linear time complexity, k-means can be time consuming for large data sets due its computationally expensive nearest neighbor search step. In this step, the nearest

center to each input vector is determined using a full search. That is, given an input vector $\mathbf{x}$, the distance from this vector to each of the K centers is calculated and then the center with the smallest distance is determined. The dissimilarity of two vectors $\mathbf{x} = (x_1, x_2, \ldots, x_D)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_D)$ is measured using the squared Euclidean distance:

$$d^2(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2^2 = \sum_{d=1}^{D} (x_d - y_d)^2 \tag{3}$$

Equation (3) shows that each distance computation requires D multiplications and $2D - 1$ additions/subtractions. It is therefore necessary to perform KD multiplications, $(2D-1)K$ additions/subtractions, and $K-1$ comparisons to determine the nearest center to each input vector. In this paper, a novel method is proposed to minimize the number of distance computations necessary to find the nearest center by utilizing the geometrical relationships among the input vector and the cluster centers. The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 describes the proposed accelerated nearest neighbor search method. Section 4 provides the experimental results. Finally, Section 5 gives the conclusions.

input : $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \in \mathbb{R}^D$ ($N \times D$ input data set)
output: $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K\} \in \mathbb{R}^D$ (K cluster centers)
`Select a random subset` $\mathcal{C}$ `of` $\mathcal{X}$ `as the initial set of cluster centers;`
while *termination criterion is not met* **do**
  for $(i = 1; i \leq N; i = i + 1)$ **do**
    `Assign` $\mathbf{x}_i$ `to the nearest cluster;`
    $m[i] = \underset{k \in \{1,2,\ldots,K\}}{\operatorname{argmin}} \|\mathbf{x}_i - \mathbf{c}_k\|^2$;
  end
  `Recalculate the cluster centers;`
  for $(k = 1; k \leq K; k = k + 1)$ **do**
    `Cluster` $\mathcal{P}_k$ `contains the set of points` $\mathbf{x}_i$ `that are nearest to the center` $\mathbf{c}_k$;
    $\mathcal{P}_k = \{\mathbf{x}_i \mid m[i] = k\}$;
    `Calculate the new center` $\mathbf{c}_k$ `as the mean of the points that belong to` $\mathcal{P}_k$;
    $\mathbf{c}_k = \frac{1}{|\mathcal{P}_k|} \sum_{\mathbf{x}_i \in \mathcal{P}_k} \mathbf{x}_i$;
  end
end

Algorithm 1: K-Means algorithm

2 Related Work

Numerous methods have been proposed for accelerating the nearest neighbor search step of the k-means algorithm. Note that k-means is also known as the Generalized Lloyd Algorithm (GLA) or the Linde-Buzo-Gray (LBG) algorithm (Linde, Buzo, and Gray 1980) in the vector quantization literature. In this section, we briefly describe several methods that perform well compared to full search.

Triangle Inequality Elimination (TIE) Method

Let $\mathbf{x}$ be an input vector and $\mathbf{y}_l$ be a known nearby center. Center $\mathbf{y}_j$ is a candidate to be the nearest center only if it satisfies $d(\mathbf{x}, \mathbf{y}_j) \leq d(\mathbf{x}, \mathbf{y}_l)$. This inequality constrains the centers which need to be considered, however, testing the inequality requires evaluating $d(\mathbf{x}, \mathbf{y}_j)$ for each center $\mathbf{y}_j$, which is computationally expensive. To circumvent this, the TIE method bounds $d(\mathbf{y}_l, \mathbf{y}_j)$ rather than $d(\mathbf{x}, \mathbf{y}_j)$. Using the triangle inequality, center $\mathbf{y}_j$ is a candidate to be the nearest center only if the following condition is satisfied (Huang and Chen 1990; Chen and Hsieh 1991; Orchard 1991; Phillips 2002):

$$d(\mathbf{y}_i, \mathbf{y}_l) \leq d(\mathbf{x}, \mathbf{y}_j) + d(\mathbf{x}, \mathbf{y}_l) \leq 2d(\mathbf{x}, \mathbf{y}_l) \tag{4}$$

With $d(\mathbf{x}, \mathbf{y}_l)$ evaluated, (4) makes it possible to eliminate center $\mathbf{y}_j$ from consideration by evaluating the distances between $\mathbf{y}_j$ and another center, $\mathbf{y}_l$, rather than between $\mathbf{y}_j$ and the input vector $\mathbf{x}$. This means that the distance computations do not involve the input vector, and thus they can be precomputed, thereby allowing a large set of centers to be rejected by evaluating the single distance $d(\mathbf{x}, \mathbf{y}_l)$.

TIE stores the distance of each center $\mathbf{y}_i$ to every other center in a table. Therefore, the method requires K tables, each with $K - 1$ entries. The entries in each table are then sorted in increasing order.

Mean-Distance-Ordered Search (MOS) Method

Given an input vector $\mathbf{x} = (x_1, x_2, \ldots, x_D)$ and a center $\mathbf{y}_i = (y_{i1}, y_{i2}, \ldots, y_{iD})$, the MOS method (Ra and Kim 1993) uses the following inequality between the mean and the distortion:

$$\left(\sum_{d=1}^{D} x_d - \sum_{d=1}^{D} y_{id} \right)^2 \leq D\, d^2(\mathbf{x}, \mathbf{y}_i) \tag{5}$$

In this method, the centers are ordered according to their Euclidean norms prior to nearest neighbor search. For a given input vector $\mathbf{x}$ and an initial center, the search procedure moves up and down from the initial center alternately in the pre-ordered list. Let $\mathbf{y}_i$ be the center with the current minimum distance, $d(\mathbf{x}, \mathbf{y}_i)$, among the centers examined so far. If a center $\mathbf{y}_j$ satisfies (6) we do not need to calculate $d(\mathbf{x}, \mathbf{y}_j)$ because $d(\mathbf{x}, \mathbf{y}_j) \geq d(\mathbf{x}, \mathbf{y}_i)$.

$$D\, d^2(\mathbf{x}, \mathbf{y}_i) \leq \left(\sum_{d=1}^{D} x_d - \sum_{d=1}^{D} y_{jd} \right)^2 \tag{6}$$

Consequently, (6) does not have to be checked for a center $\mathbf{y}_k$ that is higher, or lower, in the pre-ordered list than $\mathbf{y}_j$ as $d(\mathbf{x}, \mathbf{y}_k) \geq d(\mathbf{x}, \mathbf{y}_j) \geq d(\mathbf{x}, \mathbf{y}_i)$ (Choi and Chae 2000).

Poggi's (PGG) Method

This method takes advantage of distance estimations to reject potential centers from consideration as the nearest center for an input vector $\mathbf{x}$. Let $\mathbf{y}_l$ be the current nearest center

to $\mathbf{x}$ and let $\mathbf{y}_j$ be any other center. The center $\mathbf{y}_j$ can be rejected if the following condition is satisfied (Poggi 1993):

$$\frac{1}{|\mathbf{x}|}\sum_{d=1}^{D} x_d - \frac{1}{|\mathbf{y}_j|}\sum_{d=1}^{D} y_{jd} \geq \sqrt{\frac{d(\mathbf{x},\mathbf{y}_l)}{|\mathbf{x}|}} \qquad (7)$$

If the center $\mathbf{y}_j$ is not rejected due to (7), then $d(\mathbf{x},\mathbf{y}_j)$ must be computed. The rejection procedure for an input vector $\mathbf{x}$ is shown in Algo. 2.

for $j = 1$ **to** K **do**
  $r = \sqrt{\frac{d(\mathbf{x},\mathbf{y}_l)}{D}}$;
  $\alpha = \beta = 0$;
  for $d = 1$ **to** D **do**
    $\alpha = \alpha + x_d$;
    $\beta = \beta + y_{jd}$;
  end
  $\alpha = \alpha/D$;
  $\beta = \beta/D$;
  if $|\alpha - \beta| > r$ **then**
    Reject $\mathbf{y}_j$ as a potential nearest center for $\mathbf{x}$;
  else
    Calculate $d(\mathbf{x},\mathbf{y}_j)$;
  end
end

Algorithm 2: Rejection procedure for Poggi's method

Activity Detection (AD) Method

It can be observed that the only changes made by k-means are local changes in the set of centers, and the amount of change differs from center to center. Therefore, some centers will stabilize much faster than others. This information is useful because the *activity* of the centers can be monitored indicating whether the center was changed during the previous iteration. The centers can then be classified into two groups: *active* and *static*. The cluster of a static center is called a *static cluster*. The number of static centers increases as the iterations progress (Kaukoranta, Fränti, and Nevalainen 2000).

This activity information is useful for reducing the number of distance computations in the case of vectors in static clusters. Consider an input vector $\mathbf{x}$ in a static cluster $\mathcal{P}_i$, and denote the center of this cluster by $\mathbf{y}_i$. It is not difficult to see that if another static center $\mathbf{y}_j$ was not the nearest center for $\mathbf{x}$ in the previous iteration, it cannot be the nearest center in the current iteration either. The partition of the centers in a static cluster cannot therefore change to another static cluster. For these vectors, it is sufficient to calculate the distances only to the active centers because only they may become closer than the center of the current cluster.

This method is also useful for certain input vectors in an active cluster. Consider an input vector $\mathbf{x}$ in an active cluster α. If the center $\mathbf{c}_\alpha$ moves closer to $\mathbf{x}$ than it was before the update, then $\mathbf{x}$ can be treated as if it were in a static cluster. Note that for each input vector $\mathbf{x}$, its distance to the current nearest center $\mathbf{y}_j$ must be stored. For the remaining input vectors (whose distance to current center was increased), a full search must be performed by calculating the distances to all centers.

This method requires that a set of the active centers be maintained. This is done by comparing the new and the previous centers with at most $\mathcal{O}(KD)$ extra work and $\mathcal{O}(KD)$ extra space. For each input vector $\mathbf{x}$, the method checks to see if its current nearest center $\mathbf{y}_i$ is in the set of active centers. If this is not the case, we perform a search for the nearest center from the set of the active centers by using any fast search method. Otherwise, when $\mathbf{x}$ is mapped to active center $\mathbf{y}_i$, we calculate the distance $d(\mathbf{x},\mathbf{y}_i)$. If this distance is greater than the distance in previous iteration, a full search is necessary. If, however, the distance does not increase, only the set of the active centers is used. To store the distances of the previous iteration $\mathcal{O}(ND)$ extra space is needed.

3 Proposed Accelerated Nearest Neighbor Search Method

Many centers can be omitted from computations based on their geometrical relation to the input vector. Consider an input vector $\mathbf{x}$, its nearest center $\mathbf{y}_i$, and a new potential center $\mathbf{y}_j$. If $\mathbf{y}_j$ is located outside the circle centered at $\mathbf{x}$ with a radius equal to the Euclidean distance between $\mathbf{x}$ and $\mathbf{y}_i$, then it should not be considered as the nearest center, since the distance from $\mathbf{x}$ to $\mathbf{y}_j$ must be strictly larger than the current nearest center distance. For $\mathbf{y}_j$ to be a nearer center for $\mathbf{x}$, it must be inside the described circle. Evaluating whether or not a new center is inside this circle normally requires comparing the Euclidean distance between $\mathbf{x}$ and $\mathbf{y}_j$ with the radius. Such computations are expensive. However, they can be largely avoided if the circle is inscribed in a square whose sides are tangent to the circle. Any center that lies outside of the square is also outside of the circle and thus cannot be nearer to $\mathbf{x}$ than $\mathbf{y}_i$. Fig. 1 illustrates this in two dimensions.

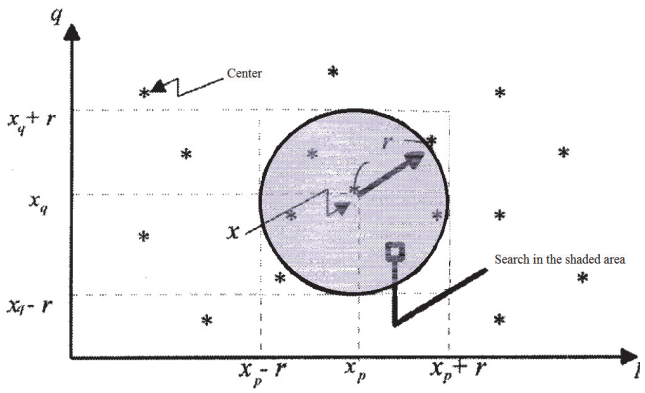

Figure 1: Geometrical relations among an input vector $\mathbf{x}$ and various cluster centers in two dimensions (Tai and Lin 1996)

To see this mathematically, let d_{min} be the Euclidean distance between the input vector $\mathbf{x}$ and its nearest center $\mathbf{y}_i$. Any center whose projection x_d on the axis d is outside the

range $[x_d - d_{min}, x_d + d_{min}]$ can be firmly discarded as it has no chance to be nearer to $\mathbf{x}$. This rule is applied to each axis (attribute) $d \in D$, eliminating any center that lies outside the square in Fig. 1 from further consideration. The focus can now be narrowed to only those centers that lie inside the square.

There exists an 'in-between' region near the corners of the square where it is possible to be inside the square and yet outside the circle. Consider a circle of radius 10, centered at the origin, O. The point A(9,9) is inside the square, however, $d(O, A) = \sqrt{9^2 + 9^2} = \sqrt{162}$, placing it outside of the circle as $\sqrt{162} > 10$. A potential center located in this region should be rejected. Determining whether a point is in the 'in-between' region in this manner requires a distance calculation. It is possible to make this decision with a less expensive operation by comparing the City-block distance (8) between O and A to the radius of the circle multiplied by the square root of D (number of dimensions). Using this approach, $T(O, A) = 18 \geq 10\sqrt{2}$, so A is rejected as expected.

$$T(\mathbf{x}, \mathbf{y}) = \sum_{d=1}^{D} |x_d - y_d| \tag{8}$$

Therefore, if $T(\mathbf{x}, \mathbf{y}_j) \geq d_{min}\sqrt{D}$, then $\mathbf{y}_j$ lies in the region between the circle and the enclosing square and thus can be firmly discarded. If, at this point, the potential center has not been rejected, it becomes the new nearest center, as it lies inside the circle with a radius equal to the distance from $\mathbf{x}$ to $\mathbf{y}_i$. The exact distance to the newly found nearest center, $\mathbf{y}_j$, must now be calculated.

The absolute differences calculated in (8) may be stored in array: $V[d] = |x_d - y_d| \quad \forall d \in \{1, 2, \ldots, D\}$. These values can then be reused in the event that the potential center is not rejected, resulting in the need for a full distance calculation. Eq. (3) then becomes (9), thereby not wasting any of the calculations preformed thus far.

$$d^2(\mathbf{x}, \mathbf{y}) = \sum_{d=1}^{D} (x_d - y_d)^2 = \sum_{d=1}^{D} V[d]^2 \tag{9}$$

Implementation

Implementation of the proposed method is fairly straightforward as shown in Algo. 3. Here, d_{min} is the current nearest neighbor distance found at any point in time, and $index$ is the index of $\mathbf{x}$'s current nearest center.

The `RejectionTest` procedure is given in Algo. 4. Note that *a*) it is necessary to set the variable $radius$ equal to the square root of the current minimum distance, i.e., $\sqrt{d_{min}}$, rather than simply d_{min} as we are using the squared Euclidean distance to measure the dissimilarity of two vectors, and *b*) the value of D does not change during the course of nearest neighbor search. Therefore, the square root of D is calculated only once outside of this method.

4 Experimental Results

Eight data sets from the UCI Machine Learning Repository (Frank and Asuncion 2012) were used in the experiments.

```
d_min = d(x, y_index);
for j = 1 to K do
    if RejectionTest (x, y_j, d_min) == false then
        dist = 0;
        for d = 1 to D do
            dist = dist + V[d]^2;
        end
        d_min = dist;
        index = j;
    end
end
```

Algorithm 3: Proposed accelerated nearest neighbor search method

```
radius = sqrt(d_min);
T = 0;
for d = 1 to D do
    V[d] = |x_d - y_d|;
    if V[d] ≥ radius then
        return true;
    end
    T = T + V[d];
end
if T ≥ radius sqrt(D) then
    return true;
else
    return false;
end
```

Algorithm 4: Rejection test for the proposed method

Table 1: Descriptions of the Data Sets (N: # points, D: # attributes)

ID	Data Set	N	D
1	Concrete Compressive Strength	1,030	9
2	Covertype	581,012	10
3	Landsat Satellite (Statlog)	6,435	36
4	Letter Recognition	20,000	16
5	Parkinsons	5,875	18
6	Person Activity	164,860	3
7	Shuttle (Statlog)	58,000	9
8	Steel Plates Faults	1,941	27

The data set descriptions are given in Table 1. Each method was executed a 100 times with each run initialized using a set of centers randomly chosen from the input vectors. Tables 2 and 4 give the average percentage of distance computations and average percentage of centers rejected per iteration for $K \in \{2, 4, 8, 16, 32\}$, respectively. Note that in the former table smaller values are better, whereas in the latter one larger values are better.

Tables 3 and 5 summarize Tables 2 and 4, respectively by averaging each method across the eight data sets for each K value. It can be seen that when compared to the other methods, the proposed method reduces the number of distance computations significantly and rejects a substantial number

Table 2: % of full distance computations per iteration

ID	Method	K=2	4	8	16	32
	PGG	71	44	40	33	30
	AD	100	88	89	82	83
1	MOS	100	100	98	94	87
	TIE	97	70	48	30	17
	Proposed	64	26	14	11	7
	PGG	53	46	41	35	30
	AD	98	95	95	83	83
2	MOS	53	45	41	34	30
	TIE	42	28	18	13	10
	Proposed	22	9	5	3	2
	PGG	83	30	29	26	24
	AD	100	93	85	82	63
3	MOS	83	30	29	26	24
	TIE	53	30	23	23	18
	Proposed	41	10	5	5	4
	PGG	86	83	80	73	68
	AD	100	99	95	89	84
4	MOS	86	84	84	74	70
	TIE	99	94	86	65	48
	Proposed	39	27	18	8	5
	PGG	61	43	40	32	26
	AD	100	92	86	76	57
5	MOS	62	43	41	33	26
	TIE	34	21	15	11	6
	Proposed	28	10	6	4	2
	PGG	57	47	39	33	26
	AD	100	80	77	73	72
6	MOS	60	48	37	29	24
	TIE	52	37	23	13	9
	Proposed	19	7	3	3	1
	PGG	70	42	39	33	29
	AD	100	88	89	82	84
7	MOS	75	40	41	32	34
	TIE	32	18	26	14	12
	Proposed	23	8	9	4	4
	PGG	52	39	34	21	18
	AD	100	89	88	76	76
8	MOS	46	37	31	18	13
	TIE	21	15	8	3	3
	Proposed	23	12	6	2	2

Table 3: Summary of Table 2

Method	K=2	4	8	16	32
PGG	67	47	43	36	31
AD	100	91	88	80	75
MOS	71	53	50	43	39
TIE	54	39	31	22	15
Proposed	32	14	8	5	3

of centers in each iteration. Furthermore, the advantage of our method is more prominent for larger K values.

Table 4: % of rejected centers per iteration

ID	Method	K=2	4	8	16	32
	PGG	2	5	8	13	20
	AD	0	7	17	25	50
1	MOS	0	0	1	5	13
	TIE	3	25	49	71	85
	Proposed	36	63	81	91	95
	PGG	30	45	55	65	72
	AD	0	0	0	0	0
2	MOS	29	43	52	62	68
	TIE	35	56	73	82	88
	Proposed	47	71	85	91	95
	PGG	14	55	63	71	75
	AD	0	5	13	17	36
3	MOS	14	55	63	71	75
	TIE	39	54	68	74	80
	Proposed	49	71	84	90	94
	PGG	9	18	20	32	37
	AD	0	0	3	10	13
4	MOS	8	12	14	24	29
	TIE	0	4	12	33	50
	Proposed	35	56	75	87	93
	PGG	27	50	60	72	81
	AD	0	4	14	23	45
5	MOS	24	45	53	64	72
	TIE	41	62	77	85	92
	Proposed	45	70	85	92	96
	PGG	26	45	62	74	81
	AD	0	0	0	0	2
6	MOS	23	40	56	67	74
	TIE	28	49	68	82	89
	Proposed	47	72	86	93	96
	PGG	18	51	60	73	74
	AD	0	9	10	17	15
7	MOS	15	46	53	65	65
	TIE	42	63	68	82	87
	Proposed	46	71	82	91	95
	PGG	32	51	63	79	88
	AD	0	3	12	30	39
8	MOS	32	50	62	77	85
	TIE	47	67	83	91	95
	Proposed	45	70	85	92	96

Table 5: Summary of Table 4

Method	K=2	4	8	16	32
PGG	20	40	49	60	66
AD	0	4	9	15	25
MOS	18	36	44	54	60
TIE	29	48	62	75	83
Proposed	44	68	83	91	95

5 Conclusions

In this paper, we presented a novel approach to accelerate the k-means clustering algorithm based on an efficient near-

est neighbor search method. The method exploits the geometrical relations among the input vectors and the cluster centers to reduce the number of necessary distance computations without compromising the clustering accuracy. Experiments on a diverse collection of data sets from the UCI Machine Learning Repository demonstrated the superiority of our method over four well-known methods.

6 Acknowledgments

This publication was made possible by a grant from the National Science Foundation (1117457).

References

Aloise, D.; Deshpande, A.; Hansen, P.; and Popat, P. 2009. NP-Hardness of Euclidean Sum-of-Squares Clustering. *Machine Learning* 75(2):245–248.

Celebi, M. E., and Kingravi, H. 2012. Deterministic Initialization of the K-Means Algorithm Using Hierarchical Clustering. *International Journal of Pattern Recognition and Artificial Intelligence* 26(7):1250018.

Celebi, M. E.; Kingravi, H.; and Vela, P. A. 2013. A Comparative Study of Efficient Initialization Methods for the K-Means Clustering Algorithm. *Expert Systems with Applications* 40(1):200–210.

Chen, S. H., and Hsieh, W. M. 1991. Fast Algorithm for VQ Codebook Design. *IEE Proceedings I: Communications, Speech and Vision* 138(5):357–362.

Choi, S. Y., and Chae, S. I. 2000. Extended Mean-Distance-Ordered Search Using Multiple l_1 and l_2 Inequalities for Fast Vector Quantization. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 47(4):349–352.

Frank, A., and Asuncion, A. 2012. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml. University of California, Irvine, School of Information and Computer Sciences.

Huang, S. H., and Chen, S. H. 1990. Fast Encoding Algorithm for VQ-based Image Coding. *Electronics Letters* 26(19):1618–1619.

Jain, A. K.; Murty, M. N.; and Flynn, P. J. 1999. Data Clustering: A Review. *ACM Computing Surveys* 31(3):264–323.

Jain, A. K. 2010. Data Clustering: 50 Years Beyond K-means. *Pattern Recognition Letters* 31(8):651–666.

Kaukoranta, T.; Fränti, P.; and Nevalainen, O. 2000. A Fast Exact GLA Based on Code Vector Activity Detection. *IEEE Transactions on Image Processing* 9(8):1337–1342.

Linde, Y.; Buzo, A.; and Gray, R. 1980. An Algorithm for Vector Quantizer Design. *IEEE Transactions on Communications* 28(1):84–95.

Lloyd, S. 1982. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory* 28(2):129–136.

Mahajan, M.; Nimbhorkar, P.; and Varadarajan, K. 2012. The Planar k-Means Problem is NP-hard. *Theoretical Computer Science* 442:13–21.

Orchard, M. T. 1991. A Fast Nearest-Neighbor Search Algorithm. In *Proceedings of the 1991 International Conference on Acoustics, Speech, and Signal Processing*, volume 4, 2297–2300.

Phillips, S. 2002. Acceleration of K-Means and Related Clustering Algorithms. In *Proceedings of the 4th International Workshop on Algorithm Engineering and Experiments*, 166–177.

Poggi, G. 1993. Fast Algorithm for Full-Search VQ Encoding. *Electronics Letters* 29(12):1141–1142.

Ra, S. W., and Kim, J. K. 1993. A Fast Mean-Distance-Ordered Partial Codebook Search Algorithm for Image Vector Quantization. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 40(9):576–579.

Tai, S. C.and Lai, C. C., and Lin, Y. C. 1996. Two Fast Nearest Neighbor Searching Algorithms for Image Vector Quantization. *IEEE Transactions on Communications* 44(12):1623–1628.

Tarsitano, A. 2003. A Computational Study of Several Relocation Methods for K-Means Algorithms. *Pattern Recognition* 36(12):2955–2966.