Information Sciences 418–419 (2017) 286–301

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

An efficient *k*-means clustering filtering algorithm using density based initial cluster centers

K. Mahesh Kumar*, A. Rama Mohan Reddy

Department of Computer Science and Engineering, SVU College of Engineering, Sri Venkateswara University, Tirupati, 517502, Andhra Pradesh, India

ARTICLE INFO

Article history:
Received 27 May 2016
Revised 27 July 2017
Accepted 31 July 2017
Available online 2 August 2017

Keywords:
k-means clustering
kd-tree
Initial cluster centers
Knowledge discovery

ABSTRACT

k-means is a preeminent partitional based clustering method that finds *k* clusters from the given dataset by computing distances from each point to *k* cluster centers iteratively. The filtering algorithm improves the performance of *k*-means by imposing an index structure on the dataset and reduces the number of cluster centers searched while finding the nearest center of a point. The performance of filtering algorithm is influenced by the degree of separation between initial cluster centers. In this paper, we propose an efficient initial seed selection method, RDBI, to improve the performance of *k*-means filtering method by locating the seed points at dense areas of the dataset and well separated. The dense areas are identified by representing the data points in a *kd*-tree. A comprehensive experimental analysis is performed to evaluate the performance efficiency of proposed method against state-of-the-art initialization methods and shown that the proposed method is efficient in terms of both running time and clustering accuracy.

1. Introduction

Data Clustering is the unsupervised classification of objects into disjoint groups such that the intra-cluster similarity of objects is maximized while minimizing the inter-cluster similarity of objects [19]. Cluster analysis plays an important role in different applications like vector quantization and data compression [15], pattern recognition [40], data mining and knowledge discovery [11], Gene similar expression grouping [27], document clustering [39] and image processing [20]. Clustering algorithms can be broadly classified into partitional and hierarchical clustering methods. Hierarchical clustering methods decompose the given dataset into several levels of nested partitions either in agglomerative (bottom-up) or divisive (top-down) manner. The different levels of partitions are represented by a dendogram for finding clusters. The single-link, complete-link and average-link are the most popular hierarchical clustering methods [21]. The Partitional clustering methods initially obtains a partition of given dataset and is fine-tuned iteratively to find out all clusters simultaneously without imposing any hierarchical structure on the dataset [38]. Density based clustering methods finds clusters by using density information of objects in its surrounding region [10]. Jain and Dubes [18] proposed a density based approach that partitions the data space into number of non-overlapping cells and computes histogram to find density estimates for obtaining clusters. DBSCAN (Density Based Spatial Clustering of Applications with Noise) [10] finds clusters by merging dense points or densely connected points separated by sparse regions. It has a time-complexity of $O(n^2)$ which can be reduced to $O(n \log n)$ using index-structures for region-query operations on low-dimensional data(less than 20). Recently, a graph-based indexing

* Corresponding author.
E-mail addresses: mahesh_cse@outlook.com (K.M. Kumar), ramamohansvu@yahoo.com (A.R.M. Reddy).

http://dx.doi.org/10.1016/j.ins.2017.07.036
0020-0255/

technique is proposed for DBSCAN region-query operation that is scalable for high-dimensional datasets [30]. Nock and Nielsen [34] proposed boosting techniques to improve the performance of clustering methods by reweighting the points in the dataset.

k-means is a partitional based clustering method that is widely used and well studied in the literature [12,19,28]. *k*-means algorithm finds *k* clusters, $C = \{c_1, \ldots c_k\}$ from a given set of n, d-dimensional data points $D = \{x_1, \ldots . x_n\}$ so as to minimize the squared distance from each data point to its nearest cluster center, Squared Error Distortion (SED).

$$SED = \sum_{i=1}^{n} \underset{j=1\ldots k}{\operatorname{argmin}} \left\| x_i - \overline{c_j} \right\|^2 \tag{1}$$

$\overline{c_j}$ is the mean of cluster c_j. Lloyd [25] proposed an iterative method based on the simple observation that the *k* centers are actually centroid of *k* clusters derived from the dataset. This method is often referred as generalized Lloyd's algorithm (GLA) [26]. Lloyd actually aimed it for scalar data, which was later extended to vector data and this extended version is called the *k*-means clustering algorithm [19]. Fuzzy c-means [4,9] is an extension to *k*-means clustering where each point is assigned to more than one cluster with a degree of membership. *k*-means is also used as one of the steps in different applications and clustering methods, usually as preprocessing or post processing step. Caiming et al. [6] employed *k*-means algorithm to reduce the computational cost of constructing a minimum spanning tree (MST) and theoretically shown that their method reduced computational complexity to $O(n^{1.5})$ compared to conventional MST algorithms with $O(n^2)$. Spectral clustering [8,41] is based on graph theory concepts and utilizes *k*-means as a post-processing step to obtain clusters. It computes pair-wise similarity matrix between all points in the dataset and finds clusters by running *k*-means on first *k* eigen vectors of similarity matrix. The cluster results of *k*-means is influenced by the position of initial cluster centers (Fig. 2). A poorly selected initial seed points not only converges the solution to a local optimum but also hampers the speed of convergence of the algorithm [14,19]. It may also produce an empty cluster i.e., the initial seed point does not qualify to be the nearest center for at least one point in the dataset. A common strategy to evade the local minima problem is to run the *k*-means algorithm multiple times with different set of initial seed points each time, and choose the partition with smallest squared error distortion (Eq. (1)) as final solution.

This paper presents an efficient implementation of *k*-means [26] by first computing a *kd*-tree [3] for the given dataset. Second, the *kd*-tree is utilized to select initial cluster centers located at dense areas and well separated. Third, the cluster centers are updated by recursively visiting the nodes in the *kd*-tree and eliminating distance computations towards cluster centers that are not nearest for at least one point in the node [23]. The rest of the paper is organized as follows: Section 2 reviews the *kd*-tree and filtering algorithm in detail. Section 3 presents the proposed RDBI-Filtering Algorithm (FA) and a comparison of RDBI with other density based initialization methods. Experimental results are presented in Section 4 and some of the concluding remarks are given in Section 5.

2. Related work

The performance of *k*-means can be improved by reducing distance computations involved in finding nearest cluster center of a point. Several methods are proposed in this direction by representing the data points in a *kd*-tree [3] and using a cluster center pruning process to eliminate the distance computations towards the cluster centers that are not reachable for a set of points stored in the node [1,23,32,36]. Alsabti et al. [1] stores the data points in a *kd-tree* and perform distance computations to the cluster centers that are reachable from at least one point in the node. The unreachable cluster centers are pruned by computing minimum and maximum distance of the node. Pelleg and Moore [36], proposed blacklisting algorithm, which also stores the data points in a *kd-tree*, but it uses a hyper-plane that bisects the *owner* of node and the candidate center for pruning unreachable cluster centers. The *owner* is taken as the center that minimizes the distance to the node. Kanungo et al. [23], improved this further by considering *owner* as the center that is nearest to the centroid of the node(filtering algorithm). Jim and Yiching [22] uses the displacement of cluster centers between iterations for pruning the candidate centers of the node. The concept of storing data points in *kd*-tree is also used for estimating parameters of a mixture of Gaussian clusters [32].

Several initial seed selection methods are proposed to improve the quality of cluster results of *k*-means. Bradley and Fayyad [5] proposed a refinement method that selects *S* small samples from dataset randomly and *k*-means is run independently on each subset. The subsets are initialized with same set of initial seed points. The final cluster centers of *S* subsets are combined into a superset and *k*-means is executed on this superset *S* number of times, each time initialized with one of the subset's centers. The centers of the subset that give minimum distortion score (Eq. (1)) are considered as finalized initial seeds. Katsavounidis et al. [25], proposed a parameter free method that chooses the first seed as the point with highest norm. The remaining cluster centers are computed as the point that is farthest from the nearest centers computed previously. Arthur and Vassilvitskii [2] proposed a stochastic procedure, *k*-means++ that selects the first cluster center randomly and the remaining cluster centers are computed as the point with a probability proportional to its distance from already selected seeds. Redmond and Heneghan [37], first constructs a *kd*-tree to obtain density estimation of points and uses a max-min method to find the seeds. Leaders-community [29] method first partitions the dataset into communities using modified leaders clustering method [16]. The centroids of the communities are selected as *k* initial seed points. The ROBIN (ROBust Initialization) [17] uses Local Outlier Factor (LOF) to avoid selecting outliers as seed point. The data points

are first sorted in descending order of their minimum-distance to the previously selected cluster centers and the point with LOF close to 1 is selected as the seed point. A comprehensive review and comparative analysis of several initialization methods is given in Ref. [7]. Next, *kd*-tree and *k*-means filtering algorithm are discussed in detail for the convenience of understanding the content of this paper.

2.1. kd-tree

kd-tree[1] [3] is a space-partitioning data structure for organizing data points in a *k*-dimensional space. It represents the data points in a top-down hierarchical partition scheme. The given set of points can be described in a *bounding box*, a smallest hyper-rectangle containing all the points. The size of the *box* is determined by the projection of minimum and maximum coordinate values of points in each dimension. There exists a closed box for each node in the *kd*-tree. The bounding box of root node is the entire data point set. Each node is split into two hyper-rectangles using an axis aligned hyperplane and assigned as its left and right nodes, thus leading to a binary tree. The process of splitting the nodes continues recursively until the number of points in the node is at most one or less than the specified number of minimum points. Such nodes are not split any further and declared as leaf nodes. There are a number of methods to select the splitting hyper-plane. A simple strategy is to split orthogonally to the longest side of cell using median coordinates of associated points. Given n number of points, *kd*-tree produces a hierarchical structure of $O(n)$ nodes and of depth $O(\log n)$. The number of points in the nodes at the same hierarchy level is roughly the same but with different volumes. This characteristic of nodes is useful to identify dense areas and non-dense areas in the dataset easily.

2.2. k-means filtering algorithm

The *k*-means filtering algorithm stores input data points in *kd*-tree [3] and selects k seed points randomly from the dataset as initial cluster centers. The cluster centers are updated by visiting all nodes in the *kd*-tree hierarchically from root node to leaf nodes. Each node maintains a set of candidate centers *ns*, which is a subset of cluster centers and act as nearest neighbor for at least one point in the node. The candidate centers of root node are the k cluster centers computed in the previous iteration or user specified cluster centers at the start of algorithm. The candidate centers of internal nodes are initially set as candidate centers of their respective parent node. Further, the candidate centers which are not nearest neighbors for at least one point in the node are deleted (or pruned) from the set of candidate centers. The root node simply propagates the candidate centers to its child nodes by neither performing cluster center pruning process nor distance computations to cluster centers. The cluster center pruning process is applied for all internal nodes. The point(s) in the leaf node is assigned to nearest cluster center by computing distances to its respective candidate centers. If the cardinality of candidate center of an internal node is equal to one (which should be c_n) then all points in such internal nodes are directly assigned to c_n and none of its subsequent child nodes are visited. The average of candidate centers (avg_{ns}) is computed as the sum of cardinality of candidate centers of leaf nodes to the number of leaf nodes.

$$avg_{ns} = \frac{\sum_{i=1}^{n} |l_i.ns|}{n}$$

where $|l_i.ns|$ denotes the cardinality of candidate centers of leaf node i and $avg_{ns} \in (1, k)$. When all nodes are complete avg_{ns} is the number of cluster centers k and when the solution is converged, $avg_{ns} = 1$ because all unreachable candidate centers are pruned by the internal nodes. Fig. 1 shows an internal node u of *kd*-tree where c_m is the centroid of u and c_n is candidate center of u nearest to c_m. For each internal node u, the number of data points $u.count$ and vector sum of all data points in the node $u.sum$ is computed. The centroid of the node can be obtained as $u.sum/u.count$. This additional information can be computed and stored at each node by suitably modifying the *kd*-tree construction without introducing any additional computational complexity. Let c_p be a candidate center in $u.ns$ and $x_{i_{max}}$ and $x_{i_{min}}$ denotes the maximum and minimum projections of u on dimension i respectively. The vertex point $H(h_1, \cdots h_d)$ of u is computed as

$$h_i = \left\{ \begin{array}{c} x_{i_{max}} \ if \ c_{pi} - c_{ni} \geq 0 \\ x_{i_{min}} \quad otherwise \end{array} \right\} \tag{2}$$

If $distance(H, c_p)$ is greater than or equal to $distance(H, c_n)$ then c_p can not be nearest center for any point in u and c_p can be deleted from $u.ns$.

The nodes can be classified based on the cardinality of its candidate centers(ns) as,

- **Complete node** A node is complete if the cardinality of candidate centers is equal to number of cluster centers. The root node is always a complete node.

$$u_{com} = \{u \in U \mid |u.ns| = k\}$$

[1] The variable k in *kd*-tree conflicts with variable k in *k*-means. The k in *k*-means represents number of clusters (or centers) whereas in *kd*-tree, k represents the dimensionality of the data structure.

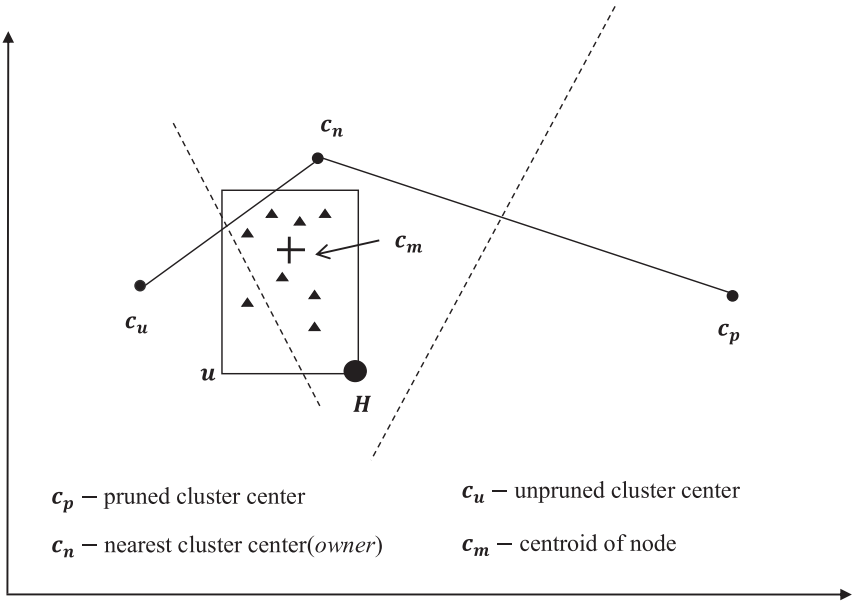

Fig. 1. Pruned and unpruned candidate centers of an internal node.

- **Effective node** A node is effective if the cardinality of candidate centers is less than the number of clusters.

$$u_{eff} = \{u \in U \mid 1 < |u.ns| < k\}$$

- **Ineffective node** A node is ineffective if a candidate center is the nearest center for all points in the node.

$$u_{inf} = \{u \in U \mid |u.ns| = 1\}$$

All points in the ineffective node are directly assigned to the candidate center(c_n) by simply adding vector sum of points and number of points in the node to the candidate center. The children of ineffective nodes are not visited further. The candidate centers of effective nodes are propagated to its children and the filtering process is applied recursively. If u is a leaf node then distance computations are performed from the data point to all the candidate centers $u.ns$ and is assigned to nearest cluster center.

3. Robust density based initialization *k*-means filtering method

In this section, we present the Robust Density Based Initialization(RDBI) method which aims to find well separated initial seed points located at dense areas and avoids outliers as seed points (Fig 2c). The proposed method is motivated from the observation that the number of nodes visited by filtering algorithm in each stage is inversely proportional to degree of separation between cluster centers. The total expected number of nodes visited by filtering algorithm in each stage is $O(2^d k \log n + dn/\rho^2)$ where ρ denotes the degree of cluster separation [24]. Since the maximum number of nodes visited is at leaf node level which is (n), the overall running time is $O(nk)$. However, the running time decreases as cluster separation ρ increases by $O(nk/\rho^2)$ since avg_{ns} (section 2.2) is minimized when clusters are well separated [23,24]. Also, when the initial clusters centers are well separated and located at dense areas (Fig. 2c) k-means could converge to global optimum with a large probability [18,19,31].

3.1. Robust density based initialization (RDBI) method

The RDBI approach determines initial seed points located at dense areas and avoids outliers as initial seeds. Table 1 defines the notations used to describe the method. The algorithm starts by computing *kd*-tree of the dataset, stipulating that a leaf node is that which contains less than a specified number of minimum points. Each leaf node maintains the count of number of points(N_i) it contains, minimum ($x_{i_{min}}$) and maximum($x_{i_{max}}$) coordinates in each dimension. The volume of leaf node is computed as,

$$V_j = \prod_{i=1}^{d} \left(x_{i_{max}} - x_{i_{min}}\right) \tag{3}$$

The density estimate of leaf node is defined using the volume and the number of points it contains, $\delta_i = N_i/V_i$. The density weight of leaf node l_i is computed using Eq. (4), where δ_{sum} is the sum of density estimates of leaf nodes,

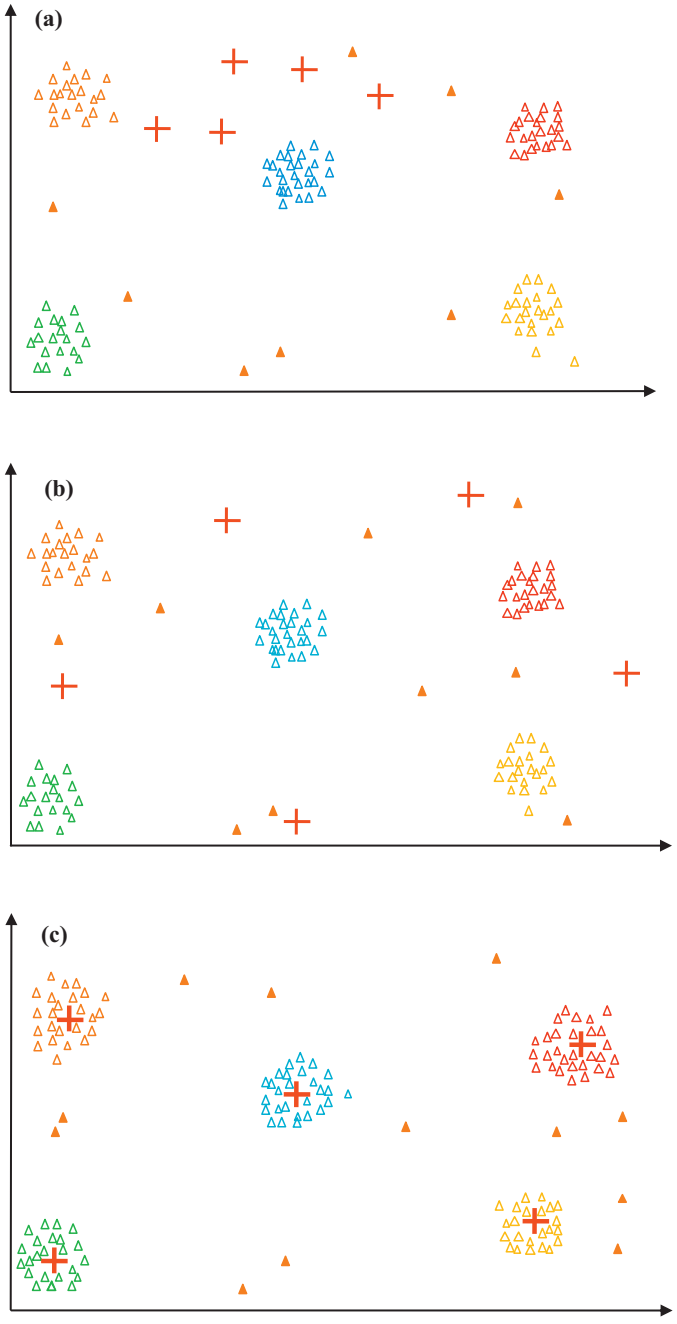

Fig. 2. **(a)** poor separated cluster centers at non-dense areas(worst-case) **(b)** isolated cluster centers at non-dense areas(avg. case) **(c)** isolated cluster centers at dense areas(best-case).

$\delta_{sum} = \sum_{i=1}^{q} \delta_i$.

$$\alpha_i = \frac{\delta_{sum}}{(\delta_{sum} - \delta_i)} \tag{4}$$

Given the density weights of leaf nodes, the first seed point is selected as the mean of leaf node having highest density estimate. The remaining seed points are computed by calculating distance estimate of leaf nodes. The distance estimate of leaf node is the distance from its nearest cluster center and is updated after computing each initial seed point. If t cluster centers are already selected, the distance estimate(γ_i) of leaf nodes in $t+1$iteration is computed as,

$$\gamma_i = \underset{k=1\ldots\ t}{\operatorname{argmin}} (\|m_i - c_k\|)$$

Table 1
Notations.

Symbol	Description
n	Size of dataset
d	Dimensionality of dataset
V	Volume of leaf node
N	Number of data points in leaf node
q	Number of leaf nodes
l	Leaf node
c	Cluster center
m	Mean of leaf node
δ	Density estimate of leaf node
α	Density weight of leaf node
γ	Distance estimate of leaf node
β	Distance weight of leaf node

The distance weight β_i of leaf node l_i is computed using Eq. (5), where γ_{sum} is the sum of density estimates of leaf nodes, $\gamma_{sum} = \sum_{i=1}^{q} \gamma_i$.

$$\beta_i = \frac{\gamma_{sum}}{(\gamma_{sum} - \gamma_i)} \tag{5}$$

The leaf node with highest distance weight and density weight is chosen as the seed point, $c_k = \{m_i \mid z = \arg\max_{i=1..q}(\alpha_i * \beta_i)\}$. The interior points of a cluster are more densely packed when compared to exterior points of cluster (Fig. 2). Hence, the high density weight of leaf node indicates that it is located near the centroid of the cluster. The noisy outliers are usually located far from centroid of clusters and occupy nodes of low density (Fig. 2). The *Noise-index* (Eq. (6)) of leaf node is used to avoid erroneously selecting an outlier as initial seed. The *Noise-index* of leaf node is computed using its distance weight and density weight as,

$$\textit{Noise-index} = \beta_i/\alpha_i \tag{6}$$

The leaf nodes in the top 10 percentage of highest *Noise-index* (or above specified threshold) are discarded and initial seeds are determined from the remaining leaf nodes only. The *k*-means [26] algorithm spends a considerable amount of time when the cluster centers are closely approaching to their final center locations and the solution is not converged while the filtering algorithm [23] runs faster by significantly reducing the distance computations to cluster centers. The cluster center pruning process is ineffective during the first few iterations of filtering method. However, as the number of iterations increases the cluster separation also increases since most of the points are assigned to their respective cluster centers, and eventually pruning process becomes effective. The cardinality of candidate centers of nodes decreases down the hierarchy and with the number of iterations. In the proposed method the cluster center pruning process is effective from the first iteration onwards since RDBI guarantees initial candidate centers as well separated seed points. Compared to *k*-means [26] and filtering algorithm [23], the proposed method takes less number of stages to converge the solution. The *k*-means and filtering algorithm run for same number of stages before the solution converges. RDBI is deterministic and insensitive to the input order of dataset as *kd*-tree is unique for the given dataset. The RDBI method is given in Algorithm 1. The step 5 in Algorithm 1, is essentially a noise or outlier handling step and is optional if not concerned of dealing noise. The *k*-means filtering method is executed in step 6 to obtain the final cluster centers.

3.2. Comparison with other density based initialization methods

This section presents a detailed comparative analysis of RDBI and state-of-the-art density based initialization methods. The density based initialization methods partition the dataset into set of hyper-rectangles and estimates density as the number of points enclosed in the hyper-rectangles. For example, the entire data space may be divided across each dimension into a given number of divisions, say g, producing g^d hyper-rectangles organized as a multidimensional grid. However, this method is clearly infeasible for high dimensional datasets since the number of hyper-rectangles grows exponentially with dimensionality(d). RDBI uses an elegant method using *kd*-tree to identify the dense areas in the dataset. A *kd*- tree partitions the data space more elegantly and guarantees that at most n hyper-rectangles(nodes) need to be dealt with for density estimation. Redmond and Heneghan [37] also proposed an initialization method that uses *kd*-tree for density estimation. The initial seeds are determined using density estimate(δ_i) and distance estimate(γ_i) of leaf nodes as $c_k = (m_i \mid z = \arg\max_{i=1..q}(\delta_i * \gamma_i))$. The value of z is dominated by higher density leaf nodes and lower density nodes are under estimated. As a result, the outcome is several seeds located at dense areas but not well separated. To combat this, density ranks are used instead of density estimate of the nodes [37]. The density rank of the leaf node is the chronological number, $\hat{\delta}_j$, in the sorted list of density estimates. Suppose, if there are 2048 leaf buckets, the highest density leaf node is assigned with highest density rank 2048 and lowest density leaf node with smallest rank 1. The initial seeds are determined using density rank of nodes as $c_k = (m_i \mid z = \arg\max_{i=1..q}(\hat{\delta}_i * \gamma_i))$. However, the density rank does not reflect the real

Algorithm 1 Robust density based initialization k-means filtering method(D, K).

Step 1: Construct a *kd-tree* representation for points in D and compute $u.count$, $u.sum$ and $(x_{i_{max}},\ x_{i_{\min}}), i = 1, 2 \cdots d$, for all internal nodes u
Step 2: Compute density weight α_i of each bucket using *Eq.* 4
$C \leftarrow \emptyset$
Step 3: Select initial seed $c_1 \leftarrow m_p$ where $p \leftarrow \underset{i}{argmax}(\alpha_i)$
$C \leftarrow c_1$
Step 4: **for** $i = 2 \cdots k$
Compute distance weights β_i for each bucket using Eq. (5)
$p \leftarrow \underset{i}{argmax}(\alpha_i * \beta_i)$
$c_i \leftarrow m_p$
$C \leftarrow C \cup^{c_i}$
end for
Step 5: Compute *Noise-index* for leaf buckets using Eq. (6) and remove leaf nodes in top 10% of highest *Noise-index*.
Repeat step 4 to obtain new initial cluster set.
$\mu^0 \leftarrow C$
$p \leftarrow 0$
Step 6:$u_{root}.ns \leftarrow \mu^p$
$p \leftarrow p + 1$
for each effective node u
$u.ns \leftarrow u_{pr}.ns$ $\quad /* \ u_{pr}$ is parent node of u $*/$
if u is leaf node
$c_n \leftarrow$ nearest candidate center of $u.x$
$c_n.sum \leftarrow c_n.sum + u.x$
$c_n.count \leftarrow c_n.count + 1$
else
for each $c_u \in u.ns$
compute vertex point H of u using *Eq.*2
if $\|H - c_u\| \geq \|H - c_n\|$
$u.ns \leftarrow u.ns - c_u$
if $|u.ns| = 1$
$c_n.sum \leftarrow c_n.sum + u.sum$
$c_n.count \leftarrow c_n.count + u.count$
end if
end for
Step 7: Repeat step 6 until $\mu^p \neq \mu^{p-1}$ where p is the Lloyd iteration number.

Table 2
Comparison of density weight and density rank of leaf nodes.

Density estimate(δ)				Density weight(α)				Density rank($\hat{\delta}$)
δ_1	δ_2	δ_3	δ_4	α_1	α_2	α_3	α_4	
8	8	8	8	1.0078	1.0324	1.0180	1.0333	1
12	12	9	12	1.0117	1.0494	1.0203	1.0508	2
14	14	11	16	1.0137	1.0581	1.0249	1.0690	3
22	22	12	22	1.0216	1.0944	1.0273	1.0973	4
27	27	15	27	1.0267	1.1184	1.0343	1.1222	5
135	35	22	35	1.1493	1.1591	1.0512	1.1643	6
221	37	35	37	1.2702	1.1697	1.0839	1.1754	7
245	45	145	45	1.3086	1.2143	1.4723	1.2217	8
355	55	195	46	1.5190	1.2750	1.7588	1.2277	9

density situation information of nodes (Table 2). RDBI uses an effective density weight approach by transforming the density estimate of leaf nodes into a relative uniform scale. The density weight of the leaf node is computed by considering the density estimates of all remaining leaf nodes(δ_{sum}). The density rank is simply the chronological number in the sorted list of density estimates and the variation in magnitude of density estimates of the nodes is discarded. If the density estimates of leaf nodes vary by a small amount then their corresponding density weights also differ by only small quantity, contrary to density ranks where the rank is always incremented in steps of one ignoring the magnitude of variation in density of leaf nodes (Table 2). Therefore, the density rank method fails to correctly represent the magnitude of variation of density estimates between leaf nodes. The density rank represents the density estimates of leaf nodes on a discrete uniform scale while density weight represents the density estimates on a continuous relative scale. A comparative analysis of density rank and density weight of four different sets of varied density estimates of leaf nodes is shown in Table 2. For the easy interpretation, the density estimates are presented in sorted order from lower to higher density estimates. The density rank 1 is assigned to the nodes with density estimate 8 in all the four cases while the corresponding density weight differs in each case because density weight is a relative value computed from the density estimates of remaining leaf nodes. In the fourth

Table 3
Comparison of avg. running time (s) and avg. distance computations per point of proposed method with *k*-means and filtering method.

Image	Size	Dimension	K	Average running time			Average distance computations		
				k-means	FA	Proposed	*k*-means	FA	Proposed
Lena	262,144	3	64	0.234	0.1427	0.0921	64	32.82	21.76
			256	0.935	0.5704	0.3717	256	131.31	87.04
			512	0.893	0.5447	0.3552	512	262.62	128.08
			1024	0.934	0.5697	0.3713	1024	525.24	285.16
Airplane	262,144	3	64	0.234	0.1427	0.0931	64	34.56	23.68
			256	0.851	0.5185	0.3379	256	138.24	74.72
			512	1.773	1.0815	0.7048	512	276.48	129.44
			1024	3.393	2.0697	1.3487	1024	552.96	218.88
Couple	65,536	3	64	0.058	0.0354	0.0231	64	40.60	22.16
			256	0.227	0.1385	0.0902	256	142.41	78.64
			512	0.448	0.2733	0.1781	512	294.83	187.28
			1024	0.869	0.5301	0.3454	1024	589.67	394.56
Boat	65,536	4	64	0.066	0.0387	0.0299	64	29.51	23.04
			256	0.261	0.1533	0.1182	256	118.06	89.16
			512	0.563	0.3307	0.2551	512	236.13	164.32
			1024	1.062	0.6226	0.4804	1024	472.26	318.64
Bridge	65,536	4	64	0.068	0.0399	0.0308	64	29.65	22.34
			256	0.262	0.1538	0.1187	256	140.61	79.36
			512	0.512	0.3007	0.2320	512	291.23	158.73
			1024	1.082	0.6355	0.4903	1024	542.47	327.47

case, the density ranks 8 and 9 are assigned to density estimates 45 and 46 respectively while the corresponding density weights are equal. In the third case, the density weight for density estimates 145 and 195 clearly reflects their superiority compared to density estimates of remaining nodes. The density weight computation do not require to sort density estimates of leaf nodes but it guarantees high density weights to higher density rank leaf nodes. The leaf node with highest density rank is assigned maximum density weight and minimum density weight is assigned to lowest density rank leaf node. When there are more number of leaf nodes having almost identical density estimates the density rank method gives worst performance since the lower density leaf nodes are also assigned higher ranks. The proposed method assigns similar density weights to nodes having almost same density estimates and in this case distance weights play a vital role in finding the initial seed points.

3.3. Performance analysis

The computational complexity of RDBI include the costs incurred in constructing a *kd*-tree of the dataset, computing density estimates of leaf nodes and finding distance computations from leaf nodes to clusters centers. The density estimate of leaf node involves computing volume of the node and its size. The *kd*-tree structure constructed by filtering algorithm is modified suitably to find and store the minimum and maximum coordinate values across each dimension $(x_{i_{max}}, x_{i_{\min}})$ and total number of points present in the node. The mean of leaf node is computed by finding center of mass of points in the bucket divided by the size of the node. The volume of bucket is computed by using the maximum and minimum co-ordinate values of points in the leaf node (Eq. (3)). The volume and centroid computation can be performed by modifying the *kd*-tree structure without introducing any additional computational complexity in terms of time and space. The $(x_{i_{max}}, x_{i_{\min}})$ values are used by filtering algorithm to find vertex point (Eq. (2)) of node during the cluster center pruning process while the proposed method utilizes them in finding the volume (Eq. (3)) of node during density estimate computation. Similarly, the parameter size of node is used by filtering algorithm for computing centroid of node whereas the proposed method uses it for computing density of node during density estimation. Therefore, our method can simply reuse the *kd*-tree structure constructed by filtering algorithm without any explicit modifications. Since, the modified *kd*-tree structure does not introduce any additional complexity in terms of time and space to the conventional *kd*-tree method, the time complexity of constructing modified *kd*-tree is $O(dn\ logn)$. The density rank initialization method [37] computes density ranks of leaf nodes by sorting the density estimates of q leaf nodes, which involves a time complexity of $O(q^2)$. RDBI computes the aggregate of density estimates of leaf nodes (δ_{sum}) in $O(q)$. The distance estimate computation overhead is same for density rank [37] and RDBI methods. The time complexity of proposed method is $O(qk)$ as the number of distance computations involved in finding the k^{th} cluster center dominates the total distance computations. The overall time complexity of RDBI including *kd*-tree computing time is $O(dn\log n + qk)$. RDBI can also be used to conventional *k*-means clustering [26] in which case the total time complexity is $O(dn\log n + qkd + nkdt)$. The number of distance computations involved in finding the cluster centers using RDBI is equal to one stage of *k*-means clustering but the cluster centers may be equal to many stages

of k-means over randomly selected initial seed points. The distance estimate computation involves finding distances from leaf nodes to the seed points that are already computed. To find the cluster center i, distance computations are made from each leaf node to the $i-1$ cluster centers that are computed previously. The total distance computations (TDC) performed to find the k initial cluster centers is given in Eq. (7).

$$\begin{aligned}\mathrm{TDC} &= \sum_{i=2}^{k}(q-(i-1))*(i-1)\\ &= q\sum_{i=2}^{k}(i-1)-\sum_{i=2}^{k}(i-1)^2\\ &= q\left(\frac{k(k+1)}{2}-k\right)-\left(\frac{(k(k+1)(2k+1))}{6}-k^2\right)\\ &= q\left(\frac{k(k-1)}{2}\right)-\left(\frac{((k-1)k(2k-1))}{6}\right)\\ &= \frac{k(k-1)(3q-(2k-1))}{6}\end{aligned} \tag{7}$$

4. Empirical analysis

The performance of proposed method is evaluated by performing extensive experiments on several sets of synthetic and real datasets. In the first experiment, the efficiency of proposed method is compared with k-means [26] and filtering algorithm [23] in terms of running time and distance computations. Second experiment, the running time and clustering accuracy of proposed method is compared with state-of-the-art initialization methods and three other clustering algorithms. Since kd-tree is a hierarchical data structure (multidimensional binary tree) the number of nodes at each level is 2^v, where v is the hierarchy level and $v=0$ for root node. The k initial seed points can be determined by navigating to any level v such that $2^v \geq k$. However, it is observed that well separated densely located cluster centers are obtained as we move down the hierarchy, for maximum value of v. The initial seed points are selected from leaf nodes and we set $minpts = 10$ for leaf nodes. All the experiments were performed on Intel core i3 processor with 4GB RAM and 1.7Ghz running Windows 7 ultimate service pack-1.All programs are compiled and executed as single-threaded java console applications using JDK 8u45 on Java Hotspot 64-bit server virtual machine.

4.1. Experiment 1

We ran three experiments to determine the efficiency of proposed method compared to k-means [26] and filtering algorithm [23] in terms of running time and distance computations. The first two experiments used synthetic data to determine the variations in running time as a function of data dimension and degree of separation between clusters. A series of synthetic datasets of sizes 50,000 and dimensions from 5 to 50 are generated. The 64 cluster centers are uniformly sampled over the hypercube $[-1,1]^d$ with d ranging from 5 to 50. The data points are generated around each center using Gaussian distribution, where each coordinate is generated independently with standard deviation $\sigma = 0.05$ along each coordinate. The running time and distance computations are calculated for k-means [26] and filtering algorithm [23] with four independent repetitions using different initial seed points. The initial cluster centers are selected randomly from the dataset. In each repetition, the k-means and filtering algorithm are initialized with same initial seed points. The average running time is calculated by dividing the running time of the algorithm with the number of stages the algorithm runs before convergence. A running time comparison of the proposed method, k-means [26] and filtering algorithm [23] with data dimension is shown in Fig 3. The proposed method can reduce the running time by a factor of 1.5 to 1.8 for data dimension ranging 25 to 50. For all effective internal nodes distance computations are involved in pruning the candidate centers. The leaf node involves distance computations for finding nearest cluster center of the point. All points in the ineffective internal nodes are directly assigned to the candidate center and are not visited any further. Hence, the sum of cardinality of candidate centers of effective nodes is accumulated as the total distance computations for filtering algorithm and proposed method. The average distance computation per stage per point is calculated by dividing total number of distance computations by the size of dataset and number of iterations. For k-means, in each stage the nearest center of a point is determined by computing distances from given point to all the cluster centers. Therefore, the average distance computations for k-means is fixed as k. From Fig. 4, we find that compared to filtering algorithm the proposed method can reduce the number of distance computations by a factor of 2.2 to 2.5.

The second experiment aims to study the variation in running time as a function of degree of cluster separation. We generated two dimensional datasets of sizes 50,000 around 64 cluster centers sampled from a uniform distribution over the hyper-cube $[-1,1]^d$. Each coordinate is generated independently using univariate Gaussian of given standard deviation. The clusters are well separated for smaller standard deviations. The datasets of varied degree of cluster separation ρ are generated around the same cluster centers by simply varying the standard deviation (σ) from 0.1 to 1.0. The clusters are

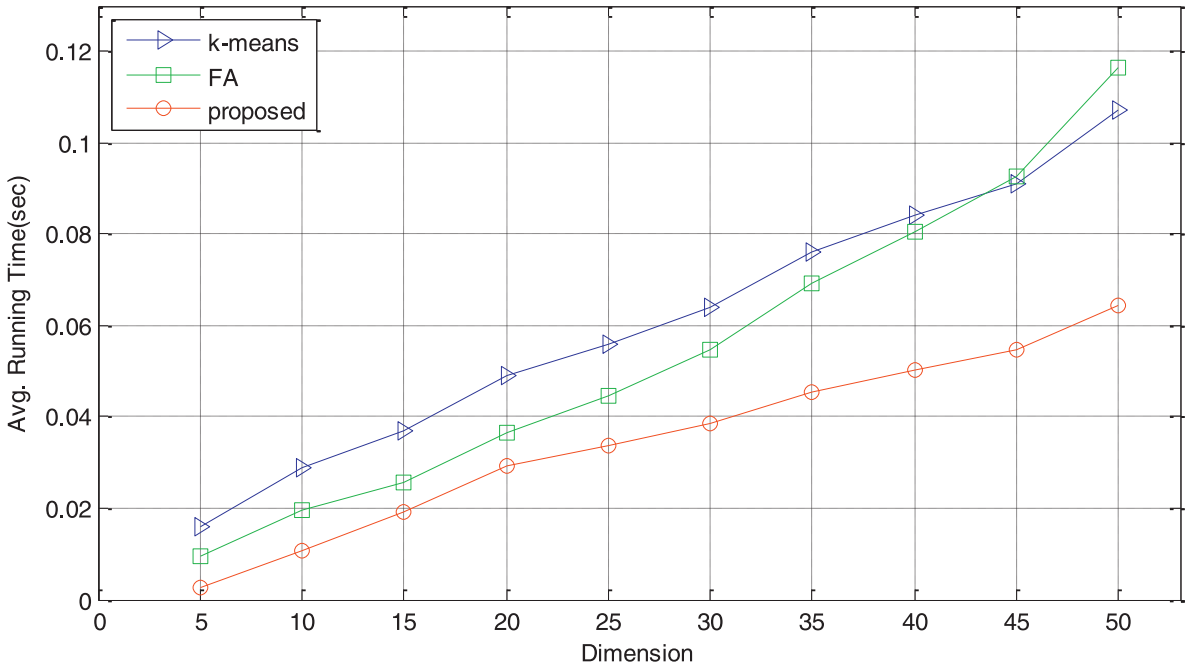

Fig. 3. The average computing time per stage of iteration for synthetic data sets with $n = 50,000$ and $k = 64$ and data dimension from 5 to 50.

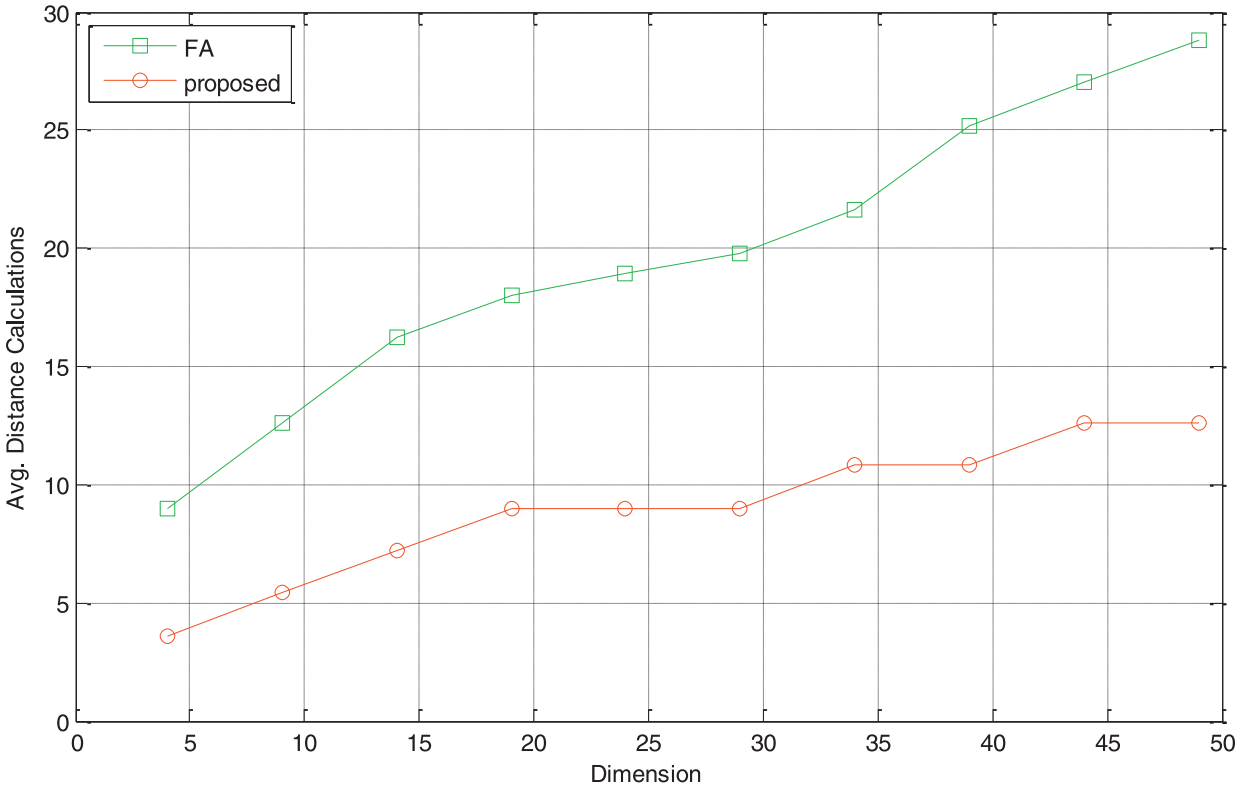

Fig. 4. The average number of distance calculations per data point and stage of iteration for data sets with $n = 50,000$ and $k = 64$ and d from 5 to 50.

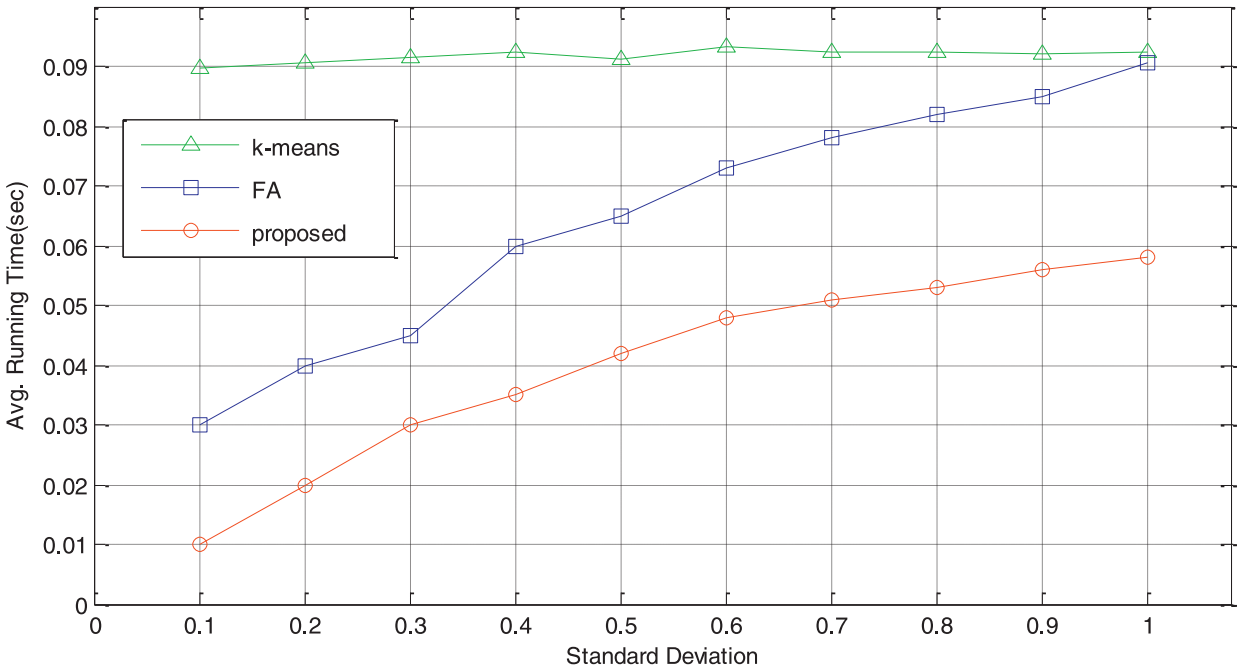

Fig. 5. The average computing time per stage of iteration for data sets with $n = 50,000$ and $k = 64$ and $d = 2$.

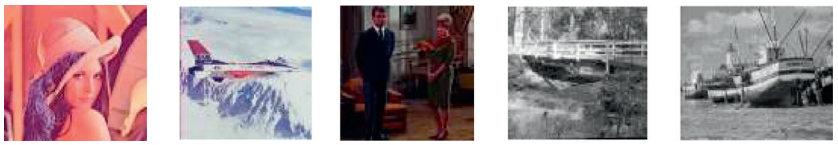

Fig. 6. Sample of images used in deriving datasets for experiment 3.

Table 4
Running time (s) comparison of *k*-means filtering method using RDBI and state-of-the-art initialization methods.

Dataset	$\|D\|$	d	RDBI	Density rank	Robin	Refinement	*k*-means++	Random	KKZ
Image Segmentation	2310	19	0.060	0.063	0.083	0.077	0.076	0.088	0.080
Pen digits	10,992	16	0.224	0.236	0.317	0.289	0.286	0.330	0.298
Letter recognition	20,000	16	1.239	1.431	1.706	1.602	1.584	1.828	2.089
Shuttle	58,000	9	1.349	1.425	1.914	1.743	1.724	1.990	1.933
Poker Hand	168,647	8	3.523	4.167	5.707	5.098	4.992	5.818	5.763

well separated when $\sigma = 0.1$ and almost overlapping clusters (unclustered) for $\sigma > 0.8$. A plot of average computing time of the three algorithms for varied range of standard deviations is shown in Fig. 5. Compared to filtering algorithm the proposed method reduced the computing time by a factor of 1.5 when the clusters are not well separated($\sigma > 0.6$). As the cluster separation decreases($\sigma > 0.8$), the running time of filtering algorithm approaches close to the running time of *k*-means because the candidate center pruning costs dominate the cluster center update costs. The running time of *k*-means is almost same in each case because the total distance computations is fixed for given n and k. From Figs. 4 and 5, we conclude that the proposed method scales better than filtering algorithm for higher dimensions (greater than 20) and when clusters are not clearly separated.

In the third experiment, the practical efficiency of proposed method is tested on five real data sets derived from benchmark images [42] in the context of image processing applications like image segmentation and compression Fig. 6. For color images each pixel is represented as one feature vector of red, green and blue components in the range $[0, 255]$ and hence, is a point in R^3. Each pixel in gray-scale images is a single gray-level component in the range $[0, 255]$. We chose a 2×2 block of pixels that represents a feature vector of four attributes. For all datasets the average running time and average distance calculations per point are computed for cluster centers k in $\{64, 256, 512, 1024\}$. The results in Table 4, reveals that the proposed method significantly outperforms the other two methods in terms of both computing time and distance computations

Table 5
Avg. distortion score comparison of k-means filtering method using RDBI and state-of-the-art initialization methods.

Dataset	k	RDBI	Density rank	Robin	Refinement		***k*-means++**		Random		KKZ
					min	avg	Min	avg	min	avg	
Image Segmentation	7	1.90 $\times 10^6$	1.93 $\times 10^6$	1.92 $\times 10^6$	1.99 $\times 10^6$	2.11 $\times 10^6$	1.98 $\times 10^6$	2.10 $\times 10^6$	2.02 $\times 10^6$	2.12 $\times 10^6$	2.27 $\times 10^6$
Pen digits	10	4.72 $\times 10^6$	4.77 $\times 10^6$	4.76 $\times 10^6$	4.94 $\times 10^6$	5.25 $\times 10^6$	4.91 $\times 10^6$	5.21 $\times 10^6$	5.01 $\times 10^6$	5.26 $\times 10^6$	5.61 $\times 10^6$
Letter recognition	26	22,772	23,019	22,949	23,814	25,338	23,694	25,144	24,177	25,386	27,078
Shuttle	7	9.40 $\times 10^7$	9.51 $\times 10^7$	9.48 $\times 10^7$	9.38 $\times 10^7$	1.04 $\times 10^8$	9.78 $\times 10^7$	1.03 $\times 10^8$	9.98 $\times 10^7$	1.04 $\times 10^8$	1.11 $\times 10^8$
Poker Hand	8	823,955	832,878	830,341	861,658	916,769	857,284	909,771	874,780	918,519	979,754

in all five cases. For Airplane dataset, compared to filtering algorithm the proposed method can reduce the running time by 34% and distance computations by 45–60%. The performance of proposed method is remarkable if n and k are larger.

4.2. Experiment 2

We performed experiments to demonstrate the efficiency of RDBI against state-of-the-art initialization methods using five benchmark datasets from UCI Machine learning repository [33]. Pen-based recognition of handwritten digits is a digit database by collecting 250 samples from 44 writers [33]. It contains 10,992 instances each of 16 attributes and a class label representing digits 0 to 9. Letter Recognition dataset [13] is a database of character image features of 26 uppercase alphabets in English alphabet. Shuttle dataset consists of 58,000 instances each of nine attributes. Image Segmentation dataset [33] consists of 19 attributes derived from seven different classes of images by processing each image in its 3 × 3 region. To determine the efficiency of proposed method for large datasets the poker hand dataset [33] is used. It consists of one million data points in ten different classes. It is an unbalanced dataset where approximately 92 percentage of total number of points belong to only two classes. We derived a subset of poker hand dataset by random sampling a subset of points from its two large classes and discarding the points in classes of size less than 200. RDBI is compared with the following initialization methods: Density Rank [37], ROBIN [17], k-means++ [2], refinement [5] and KKZ [25]. The clustering accuracy(CA) is compared using distortion score per cluster (Eq. (8)).

$$\text{Clustering Accuracy(CA)} = \frac{1}{k}\sum_{i=1}^{n} \underset{j=1\ldots k}{\operatorname{argmin}} \left\| x_i - c_j \right\|^2 \tag{8}$$

The random initialization method is essentially the conventional filtering algorithm [23]. For stochastic initialization methods, random, refinement [5] and k-means++ [2] the running time and clustering accuracy are considered as average of 25 independent repetitions. The clustering accuracy in the minimum and average cases is given separately. The kd-tree computing time is not included in the running time measurement for all the experiments. The minimum number of points for computing Local Outlier Factor (LOF) of ROBIN [17] is taken as 5. For the refinement method [5], the sub-sample is always taken as 3% of the dataset. The running time and distortion scores results are given in Tables 4 and 5 respectively. Compared to random initialization, the proposed method reduced the running time by 32–40% and improved clustering accuracy by 5–7%. For the large poker hand dataset, compared to state-of-the-art initialization methods RDBI can improve the clustering accuracy by 4–18% and reduce the running time by 18–63%.

We performed extensive experiments to demonstrate the efficiency of proposed method using a series of synthetic datasets composed with clusters of varied densities. The synthetic datasets are generated for a given number of centers k and data dimension d. Each Gaussian cluster is generated by a given mean (μ) and covariance matrix (Σ) where each component of mean is selected by a uniform random number in $[-1, 1]$. The size of each cluster is chosen uniformly between 500 and 5000 for $d = 10$, between 200 and 2000 for $d = 20$, between 100 and 1000 for $d = 50$. Therefore, the generated clusters will be of varied sizes in the three datasets and a cluster can be ten times larger than the smallest cluster. The covariance is a $d \times d$ diagonal matrix where each entry of the diagonal is chosen between $[0.8 * (s\sqrt{d}),\ s\sqrt{d},]$ where s is a scaling parameter to control the span of cluster width. Also, we introduced 3% noise in the dataset by random sampling uniformly in $[-2, 2]$. The variation in distortion score due to noise injected into the dataset is ignored and is considered as optimal distortion score. Experiments are performed on three synthetic datasets of dimensions(d): 10, 20 and 40, and for three different number of cluster centers (k) 10, 25 and 50 in each case. The stochastic procedures refinement, random and k-means++ are executed for 50 independent runs and the minimum and average case is given separately for distortion case. Compared to random initialization the proposed method can reduce the computing time by 32–35% and improve the clustering accuracy by 12–15%. From Tables 6 and 7, we find that the proposed method achieves an average distortion score close to the optimal score but in a reduced running time. Compared to state-of-the-art initialization methods the proposed method can reduce the running time by 5–35%. Compared to optimal score the distortion score of proposed method varied by only 1–2% while the state-of-the-art initialization methods by 3–18%.

Table 6
Running time (s) comparison of *k*-means filtering method using RDBI and state-of-the-art initialization methods.

d	K	RDBI	Density rank	Robin	Refinement	k-means++	random	KKZ
10	10	0.908	0.960	1.266	1.174	1.161	1.340	1.212
	25	8.237	8.703	11.478	10.645	10.530	12.150	10.992
	50	12.990	14.327	18.897	17.525	17.335	20.001	18.097
20	10	0.551	0.582	0.768	0.712	0.704	0.813	0.735
	25	4.853	5.127	6.762	6.272	6.203	7.158	6.476
	50	22.222	24.536	32.362	30.013	29.687	34.254	30.992
40	10	0.347	0.366	0.483	0.448	0.443	0.511	0.463
	25	1.725	1.822	2.404	2.229	2.205	2.544	2.302
	50	14.567	15.835	20.886	19.370	19.159	22.107	20.001

Table 7
Avg. distortion score comparison of *k*-means filtering method using RDBI and state-of-the-art initialization methods.

d	k	Optimal	RDBI	Density rank	Robin	Refinement		k-means++		Random		KKZ
						min	Avg	min	avg	min	avg	
10	10	17,936	18,363	18,562	18,505	19,203	20,432	19,106	20,276	19,496	20,471	21,835
	25	16,026	16,408	16,585	16,535	17,158	18,256	17,071	18,116	17,420	18,291	19,510
	50	7046	7214	7292	7269	7544	8026	7505	7965	7659	8041	8578
20	10	28,416	29,092	29,408	29,318	30,424	32,370	30,269	32,122	30,887	32,431	34,593
	25	24,728	25,317	25,591	25,513	26,475	28,169	26,341	27,954	26,879	28,223	30,104
	50	27,689	28,348	28,655	28,568	29,645	31,541	29,495	31,300	30,097	31,601	33,708
40	10	35,334	36,175	36,567	36,455	37,830	40,250	37,638	39,943	38,406	40,327	43,015
	25	29,332	30,030	30,356	30,263	31,404	33,413	31,245	33,158	31,883	33,477	35,709
	50	41,488	42,475	42,935	42,804	44,419	47,260	44,193	46,899	45,095	47,350	50,507

Table 8
Running time (s) and Avg. distortion score per cluster comparison of proposed clustering method and other clustering methods.

d	K	Running time				Clustering accuracy			
		Proposed	PAM	CVR	SC	Proposed	PAM	CVR	SC
5	10	0.825	2.056	1.826	2.412	7655	7449	7426	6907
	20	1.567	3.918	3.419	4.612	6304	6823	6635	6171
	40	11.851	29.125	26.227	34.654	3006	2773	2714	2981
15	10	0.517	1.342	1.151	1.465	11,108	11,177	11,765	10,942
	20	4.765	10.972	9.832	12.949	10,246	9927	10,238	9822
	40	20.514	50.218	44.957	58.218	11,308	11,253	10,891	10,962
25	10	0.879	1.929	1.723	2.279	13,128	12,862	13,328	12,796
	20	7.478	18.346	16.627	21.924	11,653	11,024	11,304	10,992
	40	13.312	32.725	29.423	35.812	15,312	15,230	15,678	14,909

The state-of-the-art initialization methods fail to meet at least one or more of the following desired characteristics: deterministic, parameter free, computationally inexpensive, robust to noise, insensitive to input order of data points and handle multi-scale clusters, while RDBI satisfies all the above properties. Experiments are also performed comparing the performance of proposed method with three other clustering methods in the literature. The Partitioning Around Medoids (PAM) clustering is similar to *k*-means except that the cluster is represented by medoid of points instead of the mean [40]. Medoid is the point of cluster that is nearest to all other points in the cluster. The spectral clustering(SC) method finds clusters by running *k*-means on eigen vectors of affinity matrix [38,41]. Clustering based on Voting Representatives with Local Maximization of Voting measure (CVR-LMV) [35] is based on the point similarity voting measure. It finds clusters by maximizing the sum of votes between points in the cluster. From Table 8, we find that the proposed method is faster than the PAM, CVR and SC methods by a factor of 1.9–2.9 with a small degradation in clustering accuracy by 2–9%.

5. Conclusions

This paper presented an efficient initial seed selection method RDBI, to improve the performance of *k*-means filtering method. The proposed work is motivated from the observation that the number of nodes visited by filtering algorithm is re-

duced by the degree of separation between cluster centers. RDBI determines well separated cluster centers located at dense areas using an elegant weight based approach for the density and distance estimation of cluster centers. The proposed method is scalable to large datasets since the running time complexity is linear in number of patterns. It uses a simple heuristic to evade erroneously selecting an outlier or noise point as initial cluster center. RDBI is deterministic, parameter free and insensitive to the input order of dataset. An experimental evaluation on five real image datasets shown that compared to filtering algorithm the proposed method can reduce the running time by 31%−35% and distance computations by 33%−60%. An experimental evaluation on synthetic and real datasets shown that compared to state-of-the-art initialization methods RDBI can reduce the running time of filtering method by 5%−35% and improve the accuracy of cluster results by 2%−15%.

Conflict of interest

None declared

Acknowledgements

The work of first author was supported by a monthly scholarship from the Department of Higher Education, Ministry of Human Resource Development (MHRD), Govt. of India, under Technical Education Quality Improvement Program, TEQIP-II-1.2. The authors thank the editors and anonymous reviewers for their constructive criticism in improving the quality of this paper.

References

[1] K. Alsabti, S. Ranka, V. Singh, An efficient k-means clustering algorithm, in: Proc. First Workshop on High Performance Data Mining, 1998, pp. 35–43.
[2] D. Arthur, S. Vassilvitskii, k-means++: the advantages of careful seeding., in: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, PA, USA, Society for Industrial and Applied Mathematics Philadelphia, 2007, pp. 1027–1035.
[3] J.L. Bentley, Multidimensional binary search trees used for associative searching, Comm. ACM 18 (1975) 509–517.
[4] J.C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum Press, 1981.
[5] P.S. Bradley, U.M. Fayyad, Refining initial points for K-means clustering, in: Proc. 15th International Conf. on Machine Learning, San Francisco, CA, Morgan Kaufmann, 1998, pp. 91–99.
[6] Z. Caiming, M. Mikko, M. Duoqian, F. Pasi, A fast minimum spanning tree algorithm based on K-means, Inf. Sci. 295 (2015) 1–17.
[7] M.E. Celebi, H.A. Kingravi, P.A. Vela, A comparative study of efficient initialization methods for the k-means clustering algorithm, Expert Syst. Appl. 40 (1) (2013) 200–210.
[8] W.Y. Chen, Y. Song, H. Bai, C.J. Lin, E.Y. Chang, Parallel spectral clustering in distributed Systems, IEEE Trans. PAMI. 33 (3) (2011) 568–586.
[9] J.C. Dunn, A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters., J. Cybernet 3 (1973) 32–57.
[10] M. Ester, H.P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proc. 2nd Int'l Conf. on Knowledge Discovery and Data Mining (KDD), Portland, Oregon, USA, 1996, pp. 226–231.
[11] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy, Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press, 1996.
[12] E. Forgy, Cluster analysis of multivariate data: efficiency vs. interpretabilityof classification, Biometrics 21 (1965) 768.
[13] P.W. Frey, D.J. Slate, Letter recognition using holland-style adaptive classifiers, Mach. Learn. 6 (1991) 125–134.
[14] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theoryof NP-Completeness, W.H. Freeman, New York, 1979.
[15] A. Gersho, R.M. Gray, Vector Quantization and Signal Compression, Kluwer Academic, Boston, 1992.
[16] J.A. Hartigan, Clustering Algorithms, John Wiley & Sons, New York, 1975.
[17] M.A. Hasan, V. Chaoji, S. Salem, M. Zaki, Robust partitional clustering by outlier and density insensitive seeding, Pattern Recognit. Lett. 30 (2009) 994–1002.
[18] A.K. Jain, R.C. Dubes, Algorithms for Clustering Data, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.
[19] A.K. Jain, Data clustering: 50 years beyond K-means, Pattern Recognit. Lett. 31 (2010) 651–666.
[20] A.K. Jain, P. Flynn, Image segmentation using clustering, Advances in Image Understanding, IEEE Computer Society Press, 1996.
[21] A.K. Jain, M.N. Murty, P.J. Flynn, Data clustering: a review, ACM Comput. Surv. 31 (3) (1999) 264–323.
[22] Z.C. Jim, L. Yi-Ching, Improvement of the K-means clustering filtering algorithm, Pattern Recognit. 41 (2008) 3677–3681.
[23] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, Y. Wu, An efficient k-means clustering algorithm: analysis and implementation, IEEE Trans. Pattern Analysis. Mach. Intell. 24 (7) (2002) 881–892.
[24] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, Y. Wu, The analysis of a simple k-means clustering algorithm, in: Proc. 16th Ann. ACM Symp. Computational Geometry, June, 2000, pp. 100–109.
[25] I. Katsavounidis, C.C.J Kuo, Z. Zhang, A new initialization technique for generalized Lloyd iteration, IEEE Signal Process. Lett. 1 (10) (1994) 144–146.
[26] S.P. Lloyd, Least squares quantization in PCM, IEEE Trans. Inf. Theory 28 (1982) 129–136.
[27] A.V. Lukashin, M.E. Lukashev, R. Fuchs, Topology of gene expression networks as revealed by data mining and modeling, Bioinformatics 19 (15) (2003) 1909–1916.
[28] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proc. Fifth Berkeley Symp. Math. Statistics and Probability, 1, 1967, pp. 281–296.
[29] K.K. Mahesh, A.R.M. Reddy, A fast k-means clustering using prototypes for initial cluster selection, IEEE Ninth International Conf. on Intelligent Systems and Control, 2015. doi: 10.1109/ISCO.2015.7282319.
[30] K.K. Mahesh, A.R.M Reddy, A fast DBSCAN clustering algorithm by accelerating neighbor searching using Groups method, Pattern Recognit. 58 (2016) 39–48.
[31] M. Meila, The uniqueness of a good optimum for k-means, in: Proc. 23rd Internat. Conf. Machine Learning, 2006, pp. 625–632.
[32] A. Moore, Very fast EM-based mixture model clustering using multi-resolution kd-trees, in: Proc. Conf. Neural Information Processing Systems, 1998.
[33] P.M. Murphy, UCI Repository of Machine Learning Databases (http://www.ics.uci.edu/mlearn/MLRepository.html), Department of Information and Computer Science, University of California, Irvine, 1994.
[34] R. Nock, F. Nielsen, On weighting clustering, IEEE Trans. Pattern Anal. Mach. Intell. 28 (8) (2006) 1–13.
[35] C. Panagiotakis, Point clustering via voting maximization, J. Classification 32 (2) (2015) 212–240.
[36] D. Pelleg, A. Moore, Accelerating exact k-means algorithms with geometric reasoning, in: Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1999, pp. 277–281.
[37] S.J. Redmond, C. Heneghan, A method for initializing the k-Means clustering algorithm using kd-trees, Pattern Recognit. Lett. 28 (2007) 965–973.

[38] A. Sami, K. Tommi, Introduction to Partitioning Based Clustering Methods with a Robust Example, Reports of the Department of Mathematical Information Technology Series C, Software and Computational Engineering No. C. 1/2006.
[39] M. Steinbach, G. Karypis, V. Kumar, A comparison of document clustering techniques, in: KDD Workshop on Text Mining, 2000, pp. 72–98.
[40] S. Theodoridis, K. Koutroumbas, Pattern Recognition, second ed., Academic Press, New York, 2003.
[41] Ulrike von Luxburg, Tech. Rep. TR-149, 2006 Max Planck Institute for Biological Cybernetics.
[42] University of southern California, Signal and Image Processing Institute, Ming Hsieh Department of electrical engineering Image database vol:3 miscellaneous., (http://sipi.usc.edu/database/database.php).

K. Mahesh Kumar completed his B. Tech and M. Tech in Computer Science and Engineering from JNT University, Hyderabad and SRM University, Chennai respectively. Currently, he is working towards his Ph.D. in the department of Computer Science and Engineering, SVU College of Engineering, Sri Venkateswara University, Tirupati, A.P., India. His areas of research interest include Data mining, Software Architecture and Language Processors.

A. Rama Mohan Reddy – He has born in 1958, received his B. Tech degree from JNT University Anantapur in 1986, Masters in Computer Science and Engineering from NIT, Warangal in 1991 and Ph.D. in Computer Science and Engineering from Sri Venkateswara University, Tirupati in 2007. He is currently working as a Professor of Computer Science and Engineering, SV University College of Engineering, Tirupati, India. His research interests are Software Engineering, Software Architecture, Cloud Computing, Operating System and Data Mining. He is life member of ISTE, IETE and CSI. He has more than 30 years of experience in teaching, 7 scholars completed Ph.D.'s under his guidance. He has 50 publications and presented 46 papers in National and International conferences.