Research Article

International Journal of Distributed Sensor Networks

International Journal of Distributed Sensor Networks
2017, Vol. 13(8)
© The Author(s) 2017
DOI: 10.1177/1550147717728627
journals.sagepub.com/home/ijdsn
SAGE

An effective and efficient hierarchical *K*-means clustering algorithm

Jianpeng Qi, Yanwei Yu, Lihong Wang, Jinglei Liu and Yingjie Wang

Abstract

K-means plays an important role in different fields of data mining. However, *k*-means often becomes sensitive due to its random seeds selecting. Motivated by this, this article proposes an optimized *k*-means clustering method, named *k**-means, along with three optimization principles. First, we propose a hierarchical optimization principle initialized by *k** seeds ($k^*>k$) to reduce the risk of random seeds selecting, and then use the proposed "top-*n* nearest clusters merging" to merge the nearest clusters in each round until the number of clusters reaches at *k*. Second, we propose an "optimized update principle" that leverages moved points updating incrementally instead of recalculating mean and *SSE* of cluster in *k*-means iteration to minimize computation cost. Third, we propose a strategy named "cluster pruning strategy" to improve efficiency of *k*-means. This strategy omits the farther clusters to shrink the adjustable space in each iteration. Experiments performed on real UCI and synthetic datasets verify the efficiency and effectiveness of our proposed algorithm.

Keywords

Data mining, clustering, *k*-means, top-*n* merging, cluster pruning

Date received: 19 May 2017; accepted: 13 July 2017

Academic Editor: Zhipeng Cai

Introduction

Clustering is to partition the data into different clusters with respect to similarity measures and is one of the most important tasks in data analysis, such as pattern discovery, pattern recognition, data summary, and image processing.[1] These fields now include many branches, namely partition-based methods, model-based methods,[2] hierarchical methods,[3] density-based methods,[4–6] graph-based methods, grid-base methods,[7] and clustering ensemble methods.[8] Partition-based method is the simplest and most important part of cluster analysis, and many algorithms are raised to facilitate its process such as *k*-means,[9]*k*-means++ ,[10] and *k*-medoids.[11]

K-means is a skillful and classic method in clustering. Given a dataset $D=\{\boldsymbol{p_i}|i=1,\ldots,n\}$, $\boldsymbol{p_i}$ in d-dimensional space $\Re^d$, *k*-means first selects *k* initial seeds, then assigns the set of points into *k* clusters by minimizing sum of squared error (*SSE*). Equation (1) shows the function of *k*-means, where $\| \boldsymbol{p_i} - m_j \|$ is the distance between point $\boldsymbol{p_i}$ and cluster C_j's center m_j that can be computed by equation (2)

$$SSE = \sum_{j=1}^{k}\sum_{i=1}^{n}\delta_{ij} \| \boldsymbol{p_i} - \boldsymbol{m_j}\|^2 \tag{1}$$

$$(\delta_{ij}=1 \text{ if } p_i \in C_j \text{ and 0 otherwise})$$

$$\boldsymbol{m_j} = \frac{\sum_{p_i \in C_j} \boldsymbol{p_i}}{|C_j|} \tag{2}$$

School of Computer and Control Engineering, Yantai University, Yantai, China

Corresponding author:
Yanwei Yu, School of Computer and Control Engineering, Yantai University, Yantai 264005, China.
Email: yuyanwei@ytu.edu.cn

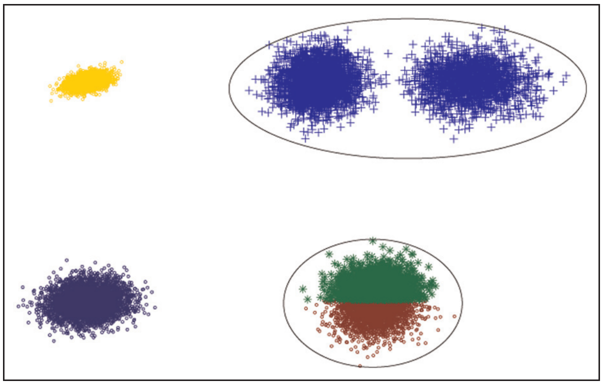


Figure 1. Example of clusters begins with a bad initialization.

Due to it is simple yet effective, k-means becomes a widely used clustering algorithm across different disciplines over the past 50 years.[12–15] However, k-means has an obvious limitation that it is sensitive to the initial seeds selecting. Random initial centers always lead to k-means trapped in the local optimum easily. A frequent observation that k-means usually merges small adjacent clusters into a larger one or divides a large cluster into several small partitions to let *SSE* reaching the convergence. Figure 1 shows an example of clusters obtained by k-means that starts with a bad initialization. We can see that one cluster is divided into two sub-clusters on the bottom right corner, and on the upper-right, two near clusters are mistakenly merged into one cluster.

Many work attempts to resolve the sensitivity of initialization for k-means.[1,16] Arthur and Vassilvitskii[17] propose a probability-based seeds selecting method called k-means++ . The probability of point p to be an initial center can be calculated by prior selected seeds. K-means++ first chooses a seed x_i randomly, and then selects the rest of initial seeds by probabilities. These probabilities can be calculated by computing the shortest squared distance D^2 between the prior chosen centers and the rest points. The more bigger D^2 means the probability more higher. However, for obtaining k centers, k-means++ needs to take k passes over the data that limit the applicability on massive dataset. Tzortzis and Likas[18] propose a method called MinMax k-means to tackle the initialization problem. In this method, a weighting factor is used to minimize the maximum intra-cluster sum of squared error (sse_{max}) instead of the global *SSE*. That is, MinMax k-means tries to minimize the largest intra-cluster variance to reduce the global *SSE* across a weighting factor with each cluster. However, MinMax does not obtain better clusters directly, especially for unbalanced cluster distribution. It also still needs to further optimize the result by performing k-means starting with MinMax k-means result. In addition, the weighting mechanism also leads to additional expensive computations.

Moreover, k-means suffers from the high computational complexity for processing high-volume datasets. Hence, there is another class of methods that attempt to improve the performance of k-means. To accelerate k-means, Hamerly[19] leverages the distance bounds and triangle inequality to avoid the redundant distance calculation. On the other hand, parallel solutions are also developed to further improve the performance of k-means. In Bahmani et al.,[10] they propose a scalable k-means++ to accelerate the initialization process in parallel on Hadoop platform. The parallel computational model reduces the number of passes over large datasets.

In this article, we propose k*-means algorithm, a novel hierarchical clustering, to improve both effectiveness and efficiency of k-means. In k*-means, we first start k-means with $k^*(k^*>k)$ initial centers that are selected randomly, and then iteratively use the "top-n nearest clusters merging" to merge the closest clusters and further refine clusters by k-means until the total number of clusters reaches at k. In this process, we propose three optimization principles to minimize the CPU cost. First, we reuse the clusters' features of last iteration to fast generate the features of new combined cluster in "top-n nearest clusters merging" process. Second, we formalize the "optimized update principle" based on the observation that just a few fractions of points are re-partitioned after several iterations. The principle uses an incremental method to update features of clusters instead of re-computation. Third, we propose the "cluster pruning strategy" to shrink the adjusting searching space. In the strategy, we provide a lower bound of searching distance between two different clusters along with theoretical analysis. In addition, we also extend experiments to explore the selecting bound of parameter k^* and n value.

While this work is based on a conference article,[20] the scope of the proposed work has been significantly extended.

(1) In this article, we now integrate these three optimized principles into a framework (section "Framework"). The framework helps us to better understand k*-means algorithm in a holistic approach.
(2) We now add and elaborate the proof of the proposed principles and strategies. These materials include "top-n nearest cluster merging" (Lemma 1), "optimized update principle" (Lemma 2 and Lemma 3), and "cluster pruning strategy" (Lemma 4).
(3) We give the pseudo-code of "top-n merging method" (Algorithm 2) and "optimized update method" (Algorithm 3) with further descriptions.
(4) We extend the experimental evaluation focusing on the effectiveness (section "Effectiveness

evaluation") and efficiency (section "Efficiency evaluation") of proposed k^*-means compared with other existing methods. In addition, we also use five additional datasets to extend the experiments with theoretical analysis.

(5) We now evaluate and discuss the contribution of three optimization principles on efficiency (section "Impact of optimization principles"). Experiments show that our proposed three optimizations reduce CPU time significantly.

(6) We also discuss the value selection of the parameter n in "top-n merging method" using additional experiments (Section "Determining of parameters k^* and n). From the experimental study, we find that we can obtain a better clustering result when n is set to 2.

Related work

Initialization methods for k-means

Forgey[21] first partition the data into k groups uniformly at random, and then choose the means of these clusters as initial seeds. MacQueen[9] proposes two different initialization methods. One method selects the first k points in the dataset as the initial seeds. And the other selects k seeds randomly. The former's result is different as the order of points, and the latter one may choose several seeds within a large cluster. A bad case that selected seeds is too close to each other, as shown in Figure 1. Bradley and Fayyad[11] present a refining initialization method. The method first selects J subsets of the data, then chooses k points randomly as the k-means initial seeds over each subset. Then, they perform k-means with MacQueen's second initialization method in each group. The obtained centers are combined into superset, which is then clustered by k-means J times. Finally, the group that reaches the best result is chosen as the initial seeds.

As mentioned above, k-means++ is a probabilistic-based seeds selected method. The point that has been far away from the already selected seeds has a high chance to be chosen as a center. They claim the complexity of k-means++ reaches at $O(\log k)$. However, the complexity would be much greater than $O(\log k)$[22] for clustering high-dimensional dataset.

Su and Dy[23] propose two methods, PCA-Part and Var-Part, to solve the initializing problem. PCA-Part first regards all the data as one cluster, and then cuts it into two partitions by the hyperplane that passes through the cluster centroid and is orthogonal to the principle eigenvector of cluster covariance matrix. After that the method repeatedly selects the cluster that has the greatest SSE to be split until k clusters are produced. Var-Part computes the variance in each dimension to find the largest variance of dimension and then uses a hyperplane to partition the cluster. This method assumes that the covariance matrix is diagonal. Lu et al.[24] propose a hierarchical initialization method. They treat the clustering problem as a weighted clustering problem to find better initial cluster centers based on the hierarchical approach. Redmond and Heneghan[25] use kd-trees to perform density estimation of the data at various locations and then use MaxMin method from densely populated leaf buckets to choose initial centers.

A recent study named MinMax k-means[18] is proposed to overcome the sensitivity of random initialization. MinMax k-means uses the objective of maximum sse_{max} of a single cluster instead of total SSE of all clusters to optimize the clusters. Since MinMax k-means aims to minimize the maximum intra-cluster variance, it tries to balance the different scale clusters. Thus, it suffers from the imbalanced dataset problems. In extreme cases, the empty or singleton cluster would cause the algorithm restarting many times. In addition, the weight mechanism in the objective of MinMax k-means also causes huge computation resources.

Another category of methods try to eliminate the dependence on initialization, such as global k-means[26] and kernel-based k-means.[27] Experimental results show that these methods achieve a better effectiveness than random initialization of k-means. However, these algorithms relieve the bad initialization at price of much more expensive computations.

Optimized variants of k-means on efficiency

Kanungo et al.[28] implement an efficient k-means method that first uses kd-tree to store instances and then adopts the index to prune or filter the candidate seeds to accelerate the procedure. Hung et al.[29] propose a Unit Block (UB)-based algorithm, which partitions the dataset into blocks and locates the centroids of the UBs. To improve the efficiency of k-means, they just examine the UBs that pass through the boundary of candidate clusters to compute the final converged centroids. Hamerly[19] uses distance bounds and the triangle inequality to accelerate k-means procedure. Another type of algorithm attempts to improve the efficiency of k-means from parallel perspective. Zhao et al.[30] propose a parallel k-means, named PKMeans, based on Hadoop platform. To accelerate the process of k-means++ , Bahmani et al.[10] propose a MapReduce-based algorithm, which focuses on the initialization stage of k-means++ to fast obtain the probabilistic initial seeds in parallel by reducing the number of passes. However, these methods also suffer from the sensitivity of initialization issues.

The algorithm

In this section, we present our proposed k^*-means method for optimizing both quality and efficiency of

k-means. First, we depict overall framework of our k*-means. We next propose three optimization principles to further cut down the CPU cost. These principles include "top-n nearest clusters merging", "optimized update principle", and "cluster pruning strategy".

Framework

Because k-means selects initial seeds randomly, it is difficult to avoid choosing noises or nearer points as the seeds. Figure 1 illustrates the situation happened where some close clusters are merged and a single cluster is divided due to a bad initialization. Especially when handle imbalanced or skewed dataset, the risk of random seeds selecting increases significantly.

Supposing cluster C_i contains n_i points, where $i = 1, 2, \ldots, k$. The probability p that the selected seeds fall into C_j in k-means initializing is $n_j/\Sigma n_i$. If cluster C_j contains a few points, the probability p would become small. And this situation further causes that some large clusters catch more than one seed. Apparently, an intuitive solution is how to improve the probability of a small cluster getting a seed to avoid a bad initialization. Thus, we extend the value of k. That is, we choose $k^*(k^*>k)$ centers instead of k. The probability of cluster C_j that owns a seed can be reached at $(n_j/\Sigma n_i) \cdot (k^*/k)$.

However, some additional steps need to be implemented to decrease k^* to k to obtain final k clusters. Next, we first introduce our k*-means framework in algorithm 1.

Top-n nearest clusters merging

As shown in algorithm 1, we first obtain k^* clusters by performing k-means with randomly chosen $k^*(k^*>k)$ initial centers. To obtain k clusters in final, we propose top-n nearest clusters merging strategy. In step 3, the algorithm would merge the clusters associated with top-n nearest distances. Before introduce top-n nearest clusters merging strategy, we first define the *edge* to describe the distance between two clusters.

Definition 1 (edge). Given two clusters C_1, C_2, and their centers c_1, c_2, the distance between C_1 and C_2 is defined as $d = |c_1 - c_2|$, we call the distance edge which connected c_1 and c_2, denoted $e_{1,2}$.

By Definition 1, we only need to find top-n shortest edge to get top-n nearest clusters. The algorithm of top-n nearest clusters merging strategy, we call Top-n method, is given in algorithm 2. We first use an ascending list structure to store the edges associated with corresponding clusters. Then, the top-n shortest edges are selected and clusters associated with the edges are merged.

However, top-n merging does not always reduce n clusters but at most n clusters (denoted as $clusters_{merge}$). For example, three edges $e_{i,j}$, $e_{i,k}$, and $e_{j,k}$ in shortlist are top-3 edges, then the clusters C_i, C_j, and C_k should be merged into one cluster at the same time, reducing two clusters. We still name the process top-n merging because we precisely merge n shortest edges. But we cannot guarantee to merge n clusters. Instead, the number of reduced cluster ranges from $\lceil(\sqrt{1 + 8 * n} - 1)/2\rceil$ to n. Therefore, we first need to determine the selection range of n value. If n is beyond the range, the number of clusters may be less than k after merging process. Here, we deduce that n should be smaller than $\lceil k^* - k\rceil$ in each round.

Proof. Suppose the top-n shortest edges involve m clusters. In extreme case of reducing minimum number of clusters, namely the shortest n edges (one edge corresponding to a combination of two clusters) are just the combinations of m clusters, then the number of involved clusters is the least, and all m clusters should be merged into one cluster. Hence, we have $n = C_m^2 = (m \cdot (m - 1))/2$, then we get $m = ((1 + \sqrt{1 + 8 \cdot n})/2)$. So, the number of reduced clusters is $m - 1 = ((\sqrt{1 + 8 \cdot n} - 1)/2)$. Therefore, if the n edges just are a subset of combinations of m clusters, then the number of reduced clusters is $\lceil((\sqrt{1 + 8 * n} - 1)/2)\rceil$.

Consider the cases of reducing maximum number of clusters, the first case is that the n edges involves $2n$ clusters, namely there are no duplicated clusters on the n edges. Intuitively, merging process would reduce n clusters in this case. The second case is that the m clusters associated with n edges are merged into one cluster but each of the m clusters appears at most two times on the n edges. Thus, we deduce $m = 2n - (n - 1) = n + 1$. So, the number of reduced clusters also is n in this case. Therefore, the number of clusters reduced by our top-n nearest clusters merging ranges from $\lceil((\sqrt{1 + 8 * n} - 1)/2)\rceil$ to n, and further we obtain $n \leq \lceil k^* - k\rceil$ to assure the number of clusters is not less than k after merging process.

Although top-n merging method reduces the number of cluster, we still need to re-compute the feature values of merged clusters for next round optimizing. To save CPU computation, we use a method to get the feature values of merged cluster directly by utilizing the feature values of previous clusters. As shown in Figure 2, suppose clusters C_i and C_j belong to top-n nearest clusters, and C is the merged cluster from C_i and C_j. Therefore, we can get Lemma 1.

Lemma 1. Given clusters C_i and C_j, m_i and m_j are means of C_i and C_j, respectively, and sse_i, sse_j are SSE of C_i

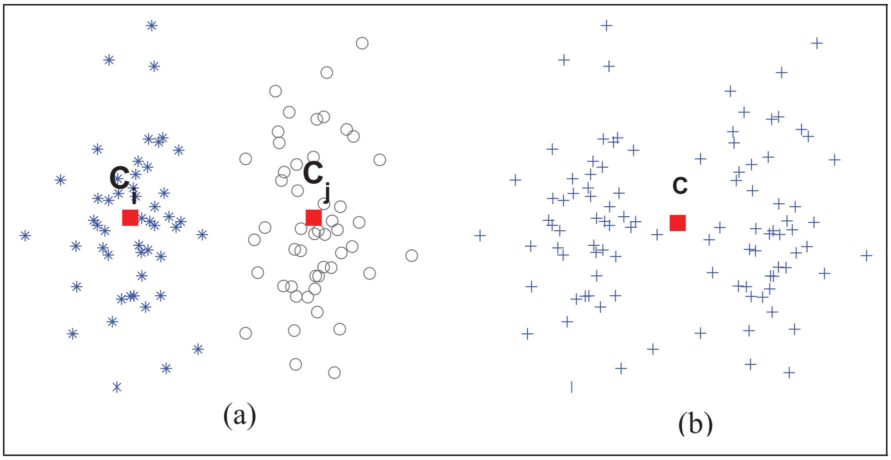


Figure 2. An example of cluster merging: (a) before merging and (b) after merging.

and C_j, respectively, if C is merged from C_i and C_j, then we have

$$\boldsymbol{m}_{\mathrm{c}} = \frac{\boldsymbol{m}_{\mathrm{i}} \cdot |C_i| + \boldsymbol{m}_{\mathrm{j}} \cdot |C_j|}{|C_i| + |C_j|} \tag{3}$$

$$\begin{aligned} sse_c = sse_i + sse_j + |C_i| \cdot \| \boldsymbol{m}_{\mathrm{i}} \|^2 + |C_j| \cdot \| \boldsymbol{m}_{\mathrm{j}} \|^2 \\ -(|C_i| + |C_j|) \cdot \| \boldsymbol{m}_{\mathrm{c}} \|^2 \end{aligned} \tag{4}$$

where $\boldsymbol{m}_{\mathrm{c}}$ is means of C, sse_c is SSE of C.

Proof. Without loss of generality, let $C_i = \{\boldsymbol{p}_u | u = 1, 2, \ldots |C_i|\}$ and $C_j = \{\boldsymbol{q}_v | v = 1, 2, \ldots |C_j|\}$.

First, since $\boldsymbol{m}_{\mathrm{i}}$ and $\boldsymbol{m}_{\mathrm{j}}$ are means of C_i and C_j, respectively, we have

$$\boldsymbol{m}_{\mathrm{i}} = \frac{\sum_{u=1}^{|C_i|} \boldsymbol{p}_u}{|C_i|} ; \boldsymbol{m}_{\mathrm{j}} = \frac{\sum_{v=1}^{|C_j|} \boldsymbol{q}_v}{|C_j|}$$

Then we get equations (5) and (6)

$$\sum_{u=1}^{|C_i|} \boldsymbol{p}_u = \boldsymbol{m}_i \cdot |C_i| \tag{5}$$

$$\sum_{v=1}^{|C_j|} \boldsymbol{q}_v = \boldsymbol{m}_j \cdot |C_j| \tag{6}$$

Note that $\boldsymbol{m}_{\mathrm{c}}$ is the means of merged cluster C, we have

$$\begin{aligned} \boldsymbol{m}_{\mathrm{c}} &= \frac{\sum_{o \in C} \boldsymbol{o}}{|C|} \\ &= \frac{\sum_{u=1}^{|C_i|} \boldsymbol{p}_u + \sum_{v=1}^{|C_j|} \boldsymbol{q}_v}{|C_i| + |C_j|} \\ &= \frac{\boldsymbol{m}_i \cdot |C_i| + \boldsymbol{m}_j \cdot |C_j|}{|C_i| + |C_j|} \end{aligned}$$

Second, sse_i and sse_j are SSE of C_i and C_j, respectively, thus

$$\begin{aligned} sse_i &= \sum_{u=1}^{|C_i|} \| \boldsymbol{p}_u - \boldsymbol{m}_i \|^2 \\ &= \sum \| \boldsymbol{p}_u \|^2 - 2\boldsymbol{m}_i \sum \boldsymbol{p}_u + \sum_{1}^{|C_i|} \| \boldsymbol{m}_i \|^2 \end{aligned}$$

$$\begin{aligned} sse_i + sse_j = \sum_{u=1}^{|C_i|} \| \boldsymbol{p}_u \|^2 + \sum_{v=1}^{|C_j|} \| \boldsymbol{q}_v \|^2 - 2\boldsymbol{m}_i \sum_{u=1}^{|C_i|} \boldsymbol{p}_u \\ - 2\boldsymbol{m}_j \sum_{v=1}^{|C_j|} \boldsymbol{q}_v + |C_i| \cdot \| \boldsymbol{m}_i \|^2 + |C_j| \cdot \| \boldsymbol{m}_j \|^2 \end{aligned} \tag{7}$$

And then, we get equation (8) by equation (7)

$$\begin{aligned} \sum_{u=1}^{|C_i|} \| \boldsymbol{p}_u \|^2 + \sum_{v=1}^{|C_j|} \| \boldsymbol{q}_v \|^2 = sse_i + sse_j \\ - |C_i| \cdot \| \boldsymbol{m}_i \|^2 - |C_j| \cdot \| \boldsymbol{m}_j \|^2 \\ + 2\boldsymbol{m}_i \sum_{u=1}^{|C_i|} \boldsymbol{p}_u + 2\boldsymbol{m}_j \sum_{v=1}^{|C_j|} \boldsymbol{q}_v \end{aligned} \tag{8}$$

By equations (5) and (6), we also get

$$\begin{aligned} 2\boldsymbol{m}_i \sum_{u=1}^{|C_i|} \boldsymbol{p}_u + 2\boldsymbol{m}_j \sum_{v=1}^{|C_j|} \boldsymbol{q}_v \\ = 2|C_i| \cdot \| \boldsymbol{m}_i \|^2 + 2|C_j| \cdot \| \boldsymbol{m}_j \|^2 \end{aligned} \tag{9}$$

Hence, we deduce equation (10) by equations (8) and (9)

$$\begin{aligned} \sum_{u=1}^{|C_i|} \| \boldsymbol{p}_u \|^2 + \sum_{v=1}^{|C_j|} \| \boldsymbol{q}_v \|^2 \\ = sse_i + sse_j + |C_i| \cdot \| \boldsymbol{m}_i \|^2 + |C_j| \cdot \| \boldsymbol{m}_j \|^2 \end{aligned} \tag{10}$$

Because all points in C_i and C_j form cluster C: $C = \{o_1, o_2, \ldots, o_{m+n}\}$, $\sum_{u=1}^{|C_i|+|C_j|} \| \boldsymbol{o_u}\|^2 = \sum_{u=1}^{|C_i|} \| \boldsymbol{p_u}\|^2 + \sum_{v=1}^{|C_j|} \| \boldsymbol{q_v}\|^2$

$$\begin{aligned} sse_c &= \sum_{u=1}^{|C_i|+|C_j|} \| \boldsymbol{o_u} - m_c\|^2 \\ &= \sum_{u=1}^{|C_i|+|C_j|} \| \boldsymbol{o_u}\|^2 - 2m_c \sum_{u=1}^{|C_i|+|C_j|} \boldsymbol{o_u} \\ &\quad + (|C_i| + |C_j|)\cdot \| \boldsymbol{m_c}\|^2 \\ &= \sum_{u=1}^{|C_i|} \| \boldsymbol{p_u}\|^2 + \sum_{v=1}^{|C_j|} \| \boldsymbol{q_v}\|^2 - 2\boldsymbol{m_c} \sum_{u=1}^{|C_i|+|C_j|} \boldsymbol{o_u} \\ &\quad + (|C_i| + |C_j|)\cdot \| \boldsymbol{m_c}\|^2 \end{aligned} \tag{11}$$

By equations (5) and (6) we have

$$2\boldsymbol{m_c} \sum_{u=1}^{|C_i|+|C_j|} \boldsymbol{o_u} = 2(|C_i| + |C_j|)\cdot \| \boldsymbol{m_c}\|^2 \tag{12}$$

Substituting equation (12) into equation (12), we obtain equation (13)

$$\begin{aligned} sse_c = &\sum_{u=1}^{|C_i|} \| \boldsymbol{p_u}\|^2 + \sum_{v=1}^{|C_j|} \| \boldsymbol{q_v}\|^2 \\ &- (|C_i| + |C_j|)\cdot \| \boldsymbol{m_c}\|^2 \end{aligned} \tag{13}$$

Therefore, we have equation (4) by equations (10) and (13), and we are done.

Lemma 1 implies that the feature values of merged cluster can be computed directly according to those of previous sub-clusters. Therefore, our top-n nearest clusters merging exactly leverages the optimization strategy to reduce merging cost.

Optimized update principle

From extending experiments on UCI data (see details in section "Experimental setup and methodologies"), we observe that the number of moved points declines sharply during the k-means iteration processes. Figure 3 shows two samples over two real-world datasets. As k-means iteration goes on, the points in clusters trend toward stability; thus, the number of moved points also goes toward zero. In particular, the number of points moved across clusters is very low after the forth iteration. And only average 3% of points are adjusted from fifth iteration to end. Motivated by this observation, we propose an optimized update principle, which updates feature values in each iteration using the moved points instead of re-computation from all points.

To understand our optimized update principle, we first state Lemma 2 and Lemma 3.

Lemma 2. Given a cluster C_i, and its center $\boldsymbol{m_i}$ and sum of squared error sse_i, if a point $\boldsymbol{p}$ is added into C_i, then we get the new center m_i' and sum of squared error $sse_{i'}$ as equations (14) and (15), where $|C_i|$ is number of points in C_i

$$\boldsymbol{m'_i} = \boldsymbol{m_i} + \frac{\boldsymbol{p}-\boldsymbol{m_i}}{|C_i|+1} \tag{14}$$

$$sse'_i = sse_i + \frac{|C_i|\cdot\|\boldsymbol{p}-\boldsymbol{m_i}\|^2}{|C_i|+1} \tag{15}$$

Proof. Lemma 2 can be easily proved by Lemma 1. Consider $\mathbf{p}$ as a singleton C_j that just has one point. Thus we have

$$\begin{aligned} \boldsymbol{m'_i} &= \frac{\boldsymbol{m_i}\cdot|C_i| + \boldsymbol{p}\cdot 1}{|C_i|+1} = \frac{\boldsymbol{m_i}\cdot(|C_i|+1) - \boldsymbol{m_i} + \boldsymbol{p}}{|C_i|+1} \\ &= \boldsymbol{m_i} + \frac{\boldsymbol{p}-\boldsymbol{m_i}}{|C_i|+1} \end{aligned}$$

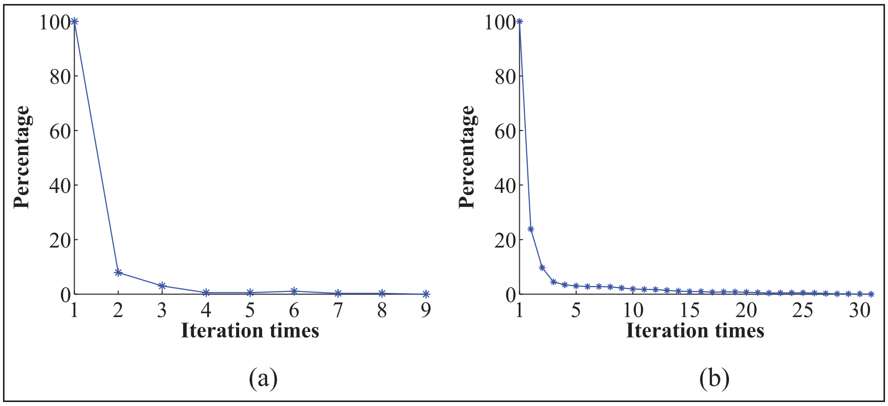


Figure 3. The number of moved points in k-means iterations: (a) dermatology and (b) optdigits.

and

$$
\begin{aligned}
sse'_i &= sse_i + 1 + |C_i|\cdot \parallel \boldsymbol{m}_{\mathrm{i}}\parallel^2 + 1\cdot \parallel \boldsymbol{p}\parallel^2 \\
&\quad - \frac{\parallel \boldsymbol{m}_{\mathrm{i}} \cdot |C_i| + \boldsymbol{p} \cdot 1\parallel^2}{|C_i| + 1} \\
&= sse_i + \frac{|C_i|\cdot \parallel \boldsymbol{p} - \boldsymbol{m}_{\mathrm{i}}\parallel^2}{|C_i| + 1}
\end{aligned} \tag{16}
$$

Lemma 3. Given a cluster C_i, and its center $\boldsymbol{m}_{\mathrm{i}}$ and sum of squared error sse_i, if a point $\boldsymbol{p}$ is moved out from C_i, then we get the new center $\boldsymbol{m}'_i$ and sum of squared error sse'_i as equations (17) and (18), where $|C_i|$ is the number of points

$$
\boldsymbol{m'}_{\boldsymbol{i}} = \boldsymbol{m_i} + \frac{\boldsymbol{m_i} - \boldsymbol{p}}{|C_i| - 1} \tag{17}
$$

$$
sse_{i'} = sse_i - \frac{|C_i|\cdot \parallel \boldsymbol{m_i} - \boldsymbol{p}\parallel^2}{|C_i| - 1} \tag{18}
$$

Proof. First, we show proof of equation (17).

Since

$$
\boldsymbol{m}'_i = \frac{\left(\sum_{o\in C} \boldsymbol{o}\right)}{|C_i| - 1} - \boldsymbol{p}
$$

By equation (5), we can easily get

$$
\begin{aligned}
\boldsymbol{m'}_{\boldsymbol{i}} &= \frac{\boldsymbol{m_i} \cdot |C_i| - \boldsymbol{p}}{|C_i| - 1} \\
&= \frac{\boldsymbol{m_i} \cdot |C_i| - \boldsymbol{m_i} + \boldsymbol{m_i} - \boldsymbol{p}}{|C_i| - 1} \\
&= \boldsymbol{m_i} + \frac{\boldsymbol{m_i} - \boldsymbol{p}}{|C_i| - 1}
\end{aligned} \tag{19}
$$

Next, we give proof of equation (18). Let sse_i denote the previous *SSE* value, $sse_{i'}$ be the current value. Let ∂sse be the difference between sse_i and $sse_{i'}$. Thus, $\partial sse = sse_i - sse_{i'}$ when move out $\boldsymbol{p}$ from C_i

$$
\begin{aligned}
\partial sse &= sse_i - sse'_i \\
&= \sum_{u=1}^{|C_i|} \parallel \boldsymbol{p_u} - \boldsymbol{m_i}\parallel^2 - \sum_{u=1}^{|C_i|-1} \parallel \boldsymbol{p_u} - \boldsymbol{m'}_{\boldsymbol{i}}\parallel^2 \\
&= \sum_{u=1}^{|C_i|} \parallel \boldsymbol{p_u}\parallel^2 - 2\boldsymbol{m_i} \sum_{u=1}^{|C_i|} \boldsymbol{p_u} + |C_i|\cdot \parallel \boldsymbol{m_i}\parallel^2 \\
&\quad - \sum_{u=1}^{|C_i|-1} \parallel \boldsymbol{p_u}\parallel^2 + 2\boldsymbol{m'}_{\boldsymbol{i}} \sum_{u=1}^{|C_i|-1} \mathbf{p_u} \\
&\quad -(|C_i| - 1)\cdot \parallel \mathbf{m'}_i\parallel^2
\end{aligned}
$$

Because

$$
\parallel \boldsymbol{p}\parallel^2 = \sum_{u=1}^{|C_i|} \parallel \boldsymbol{p_u}\parallel^2 - \sum_{u=1}^{|C_i|-1} \parallel \boldsymbol{p_u}\parallel^2
$$

Therefore, we have

$$
\begin{aligned}
\partial sse = {} & \parallel \boldsymbol{p}\parallel^2 - 2\boldsymbol{m_i} \sum_{u=1}^{|C_i|} \boldsymbol{p_u} + |C_i|\cdot \parallel \boldsymbol{m_i}\parallel^2 \\
& + 2\boldsymbol{m'}_{\boldsymbol{i}} \sum_{u=1}^{|C_i|-1} \boldsymbol{p_u} - (|C_i| - 1)\cdot \parallel \boldsymbol{m'}_{\boldsymbol{i}}\parallel^2
\end{aligned}
$$

According to equation (5), then we get

$$
\partial sse = \parallel \boldsymbol{p}\parallel^2 - |C_i|\cdot \parallel \boldsymbol{m_i}\parallel^2 + (|C_i| - 1)\cdot \parallel \boldsymbol{m}'_{\boldsymbol{i}}\parallel^2
$$

We further deduce that

$$
\begin{aligned}
\partial sse = {} & \parallel \boldsymbol{p}\parallel^2 - |C_i|\cdot \parallel \boldsymbol{m_i}\parallel^2 \\
& + (|C_i| - 1)\cdot \parallel \boldsymbol{m_i} + \frac{\boldsymbol{m_i} - \boldsymbol{p}}{|C_i| - 1}\parallel^2 \\
= {} & \frac{|C_i|\cdot \parallel \boldsymbol{m_i} - \boldsymbol{p}\parallel^2}{|C_i| - 1}
\end{aligned}
$$

by equation (17).

Because $sse_{i'} = sse_i - \partial sse$, hence equation (18) is obtained and we are done.

Lemma 2 and Lemma 3 depict updating method of *mean* and *SSE* of cluster when a point is moved in or out, respectively. Therefore, the feature values of a cluster can be maintained incrementally via the moved points during *k*-means iteration, rather than re-computed from all points in the cluster.

Armed with Lemma 2 and Lemma 3, we propose the optimized update method to reduce computation costs, as shown in algorithm 3. In each iteration, we maintain the copies of feature values incrementally according to moved points. Optimized update method obviously saves much computation resource when the number of moved points is very low. However, at beginning iterations of *k*-means, specially, the first round *k*-means, moved points amount is relatively larger, and the optimized update method may cost more time than completed method. Therefore, we use a threshold θ, the percentage of moved points in each iteration, to determine whether start to perform optimized update method in the first round *k*-means. In subsequent experiments, we set $\theta = 5\%$.

Cluster pruning strategy

To reduce the computational costs, we now introduce optimization principle, regarding to minimization of distance comparison, termed cluster pruning strategy.

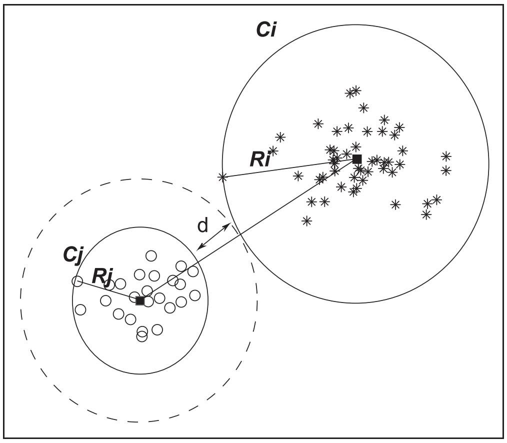


Figure 4. Pruning distance between two clusters.

In k-means iterations, each point needs to be examined if it is closer to its center than any other centers. Hence, each point has a larger searching space. For example, if there are k clusters, each point needs to calculate and compare distance $(k-1)$ times in each iteration. However, most of these computations are redundant. As exactly shown in Figure 3, only very few points would be moved among clusters in adjusting iterations. We also observe that if two clusters are distant from each other, the points in these two clusters are no need to compare with each other in adjusting steps.

Next, we introduce our pruning optimization method for each point at cluster level based on definition 2 and Lemma 4.

Definition 2 (radius). Given a cluster C_i, the radius of the cluster C_i is the maximum distance from the center of C_i to its points, denoted R_i.

Lemma 4. Given two clusters C_i, C_j, and their radius R_i, R_j, if $e_{i,j} \geq R_i + R_j + |R_i - R_j|$, then there is no point in $C_i(C_j)$ that is moved into $C_j(C_i)$ in adjusting iterations.

Proof. Suppose there are two clusters, C_i and C_j, their radius are R_j and R_i, respectively, as shown in Figure 4. Without loss of generality, we assume $R_i > R_j$, let d denote the gap between two clusters; thus, $d = e_{i,j} - R_i - R_j$. Obviously, all points in C_j are closer to $\boldsymbol{m_j}$ (the center of C_j) than $\boldsymbol{m_i}$ (the center of C_i).

Now, we consider the farthest point in C_i, denoted $\boldsymbol{p}$. By the definition of radius, the distance from $\boldsymbol{p}$ to the center of C_i is R_i, and the distance from $\boldsymbol{p}$ to the center of C_j must not be less than $d + R_j$, namely $distance(\boldsymbol{p}, \boldsymbol{m_j}) \geq d + R_j$.

If $distance(\boldsymbol{p}, \boldsymbol{m_j}) \geq R_i$, then all points in C_i are not moved into C_j certainly. Hence, if $d + R_j \geq R_i$, then $distance(p, \boldsymbol{m_j}) \geq R_i$ must hold.

Since

$$\begin{aligned} d + R_j \geq R_i &\Rightarrow e_{i,j} - R_i - R_j + R_j \geq R_i \\ &\Rightarrow e_{i,j} \geq R_i + R_j + |R_i - R_j| \end{aligned}$$

Therefore, $e_{i,j} \geq R_i + R_j + |R_i - R_j|$ guarantees that there is no point that be adjusted between C_i and C_j.

Lemma 4 guides our pruning rule for k*-means iteration. For each cluster C_i, if distances between C_i and other clusters, namely *edges* associated with C_i, are greater than the pruning distance of $R_i + R_j + |R_i - R_j|$, then we eliminate these farther clusters from searching space in k*-means iteration with respect to C_i.

It is essential to show the usage of *radius* in pruning. Each cluster C contains two additional indicators called *radius* and *radius*′. *radius* refers to the Definition 2 and *radius*′ represents a duplicate of *radius*. In k-means adjusting iterations, we first scan each point p in C_i, if p needs to be moved into another cluster C_j and the distance between p and $\boldsymbol{m_j}$ is larger than $C_j.radius'$, then we update $C_j.radius'$. Meanwhile, if a point p' in another cluster needs to be moved into C_i, we also update $C_i.radius'$ if necessary. After each iteration, we assign the value of $C_i.radius'$ and $C_j.radius'$ to their $C_i.radius$ and $C_j.radius$, respectively. These steps are shown in lines 7–21 of Algorithm 4.

k*-means algorithm

Next, we present our extending k*-means algorithm based on our proposed three optimization strategies. The detail pseudo-code of k*-means is shown in algorithm 4.

In k*-means, we also use the random initialization method to choose k^* starting centers, and first assign all points into k^* clusters. Then, we get feature values of *mean* for each cluster, as shown in lines 1–3. Next, k*-means performs hierarchical clustering along with k-means adjusting iteration in lines 4–22 and the top-n nearest clusters merging strategy in line 23–25. Line 6 describes the proposed cluster pruning strategy. We use collections *CS* to store the neighbor clusters of the specific cluster after prune remote clusters by Lemma 4. Then, we only need to verify the adjustable clusters in search space of *CS* for each point in C_i. Once the percentage of the moved points is lower than given θ in first round, k-means optimized update principle will be started and *radius*′ will be updated during this process (lines 8–19). For algorithm's efficiency, we maintain a value *radius*′ in each cluster to update its radius (lines 10–14). At the end of each iteration, each cluster mean m and its *radius* are replaced by m' and *radius*′ directly. Lines 23–25 show top-n nearest clusters merging which

Table 1. Descriptions of synthetic and real-world datasets.

No.	Dataset	No. of points	No. of attributes	No. of classes
D1	Smalldata	3851	2	30
D2	Unbaldata	18,000	2	5
D3	LargeUnbal	41,650	2	12
D4	Bigdata	120,000	2	50
D5	Ecoli	307	7	4
D6	Pendigits	10,992	16	10
D7	Optdigits	5620	64	10
D8	Dermatology	366	34	6
D9	Folio1	300	500	15
D10	Folio2	300	500	20
D11	CNAE-9	1080	857	9

reduces the number of clusters from $|C|$ to $|C| - [\lceil((\sqrt{1 + 8 * n} - 1)/2)\rceil, n]$. Therefore, parameter n is not fixed, but ranges from given n to 1. For each round refining of k*-means, we use a decrease strategy to determine value of n, and a top-1 may be performed at final round to make number of clusters reach at k.

Experiments

Experimental setup and methodologies

All algorithms are implemented by Java, and all experiments are performed on platform equipped with 2.4 GHz Intel Core i3 CPU, 4 GB memory, and Windows 7 operating system.

Datasets. We use 11 datasets including 4 synthetic datasets and 7 real-world datasets from the UCI Machine Learning Repository[31] to evaluate the effectiveness and efficiency of our k*-means. These datasets cover many fields such as spatial data, image, text, etc. As described in Table 1, D1–4 are generated by our data generator using mixture Gauss model for covering different situations, such as unbalanced clusters, clusters with arbitrary shapes.

As shown in Figure 5(a), D1 includes 30 clusters that composed of 3851 points. In this dataset, each cluster contains less than 200 points with Gauss distribution and number of clusters is balanced. We denote the small dataset as Smalldata.

D2 (Figure 5(b)) includes five clusters with clear gaps; number of points in the dataset reaches at 18,000 but is unbalanced in each cluster. We name the unbalanced dataset Unbaldata.

D3 (Figure 5(c)) contains 41,065 points, which consists of 12 clusters. Clusters of the dataset are also unbalanced but with clear shape. This dataset is denoted as LargeUnbal.

The forth dataset shown in Figure 5(d) comprises massive data points of 120,000, including 50 clusters with arbitrary shapes. Moreover, there are no obvious gaps for some clusters, and the number of points in clusters also is unbalance. We denote the big dataset with arbitrary shape clusters as Bigdata.

Ecoli[31] is calculated from the amino acid sequences, which includes 8 classes that comprised of 336 proteins. The data has eight attributes (includes one label), and the number of classes is different, thus it is an unbalanced dataset.

Pendigits[31] is collected from 44 writers, which contains 10,992 instances that represented as constant length feature vectors in 16-dimensional space.

Optdigits[31] is optical recognition of handwritten digits by extracting normalized bitmaps of handwritten digits from a preprinted form. Each preprinted form is represented as an 8 × 8 matrix, where every element is an integer. The dataset includes 5620 instances in 64_dimensional space.

Dermatology[31] contains 366 patients with 34 attributes, which is partitioned into 6 types of erythemato-squamous disease. The dataset is also unbalanced.

Folio1 & Folio2[31] are leaves images taken from different plants in the farm of the University of Mauritius and nearby locations. We first extract the SIFT descriptors from these images, and then use the bag of 500 visual words to represent the data. Two subsets of the data were generated, Folio1 (300 leaves from 15 different types of plants, balanced) and Folio2 (300 leaves but from 20 different types of plants and unbalanced).

CNAE-9[31] contains 1080 documents of free text business descriptions of Brazilian companies. These documents are categorized into a subset of nine categories. Each document is represented as a vector, where the weight of each word is its frequency in the document.

Experimental methodologies. For effectiveness evaluation of our algorithm, we use three metrics of average *SSE*, normalized mutual information (NMI), and Silhouette

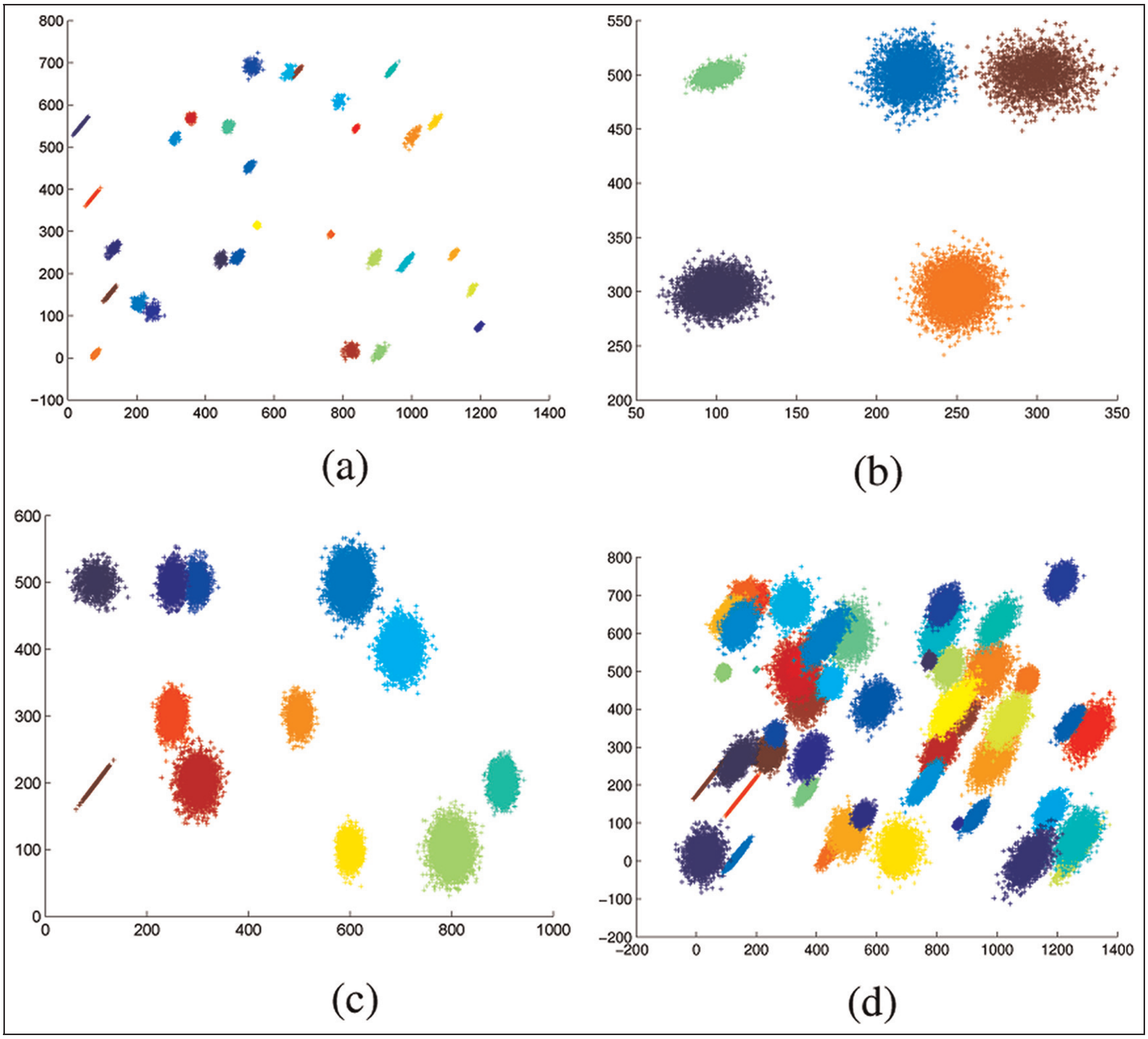


Figure 5. Demonstrations of four synthetic datasets: (a) D1, (b) D2, (c) D3, and (d) D4.

coefficient (SC)[32] to estimate the quality of clustering. *SSE* utilized is described as equation (1) in section "Introduction", and we take the average *SSE* that divides total *SSE* by the number of points. Equation (20) depicts computing method of NMI, where w_k is the obtained cluster k, c_j is the labeled class j, N is the size of dataset, and the $\mathcal{H}_M$ and $\mathcal{H}_C$ represent the entropy of clusters and classes, respectively

$$\text{NMI} = \frac{\sum_k \sum_j \frac{|w_k \cap c_j|}{N} \log \frac{N|w_k \cap c_j|}{|w_k| \cdot |c_j|}}{[\mathcal{H}_M + \mathcal{H}_C]/2} \tag{20}$$

$\mathcal{H}$ is defined as equation (21)

$$\mathcal{H}_M = -\sum_k \frac{|w_k|}{N} \log \frac{|w_k|}{N} \tag{21}$$

Therefore, NMI score ranges from 0 to 1.0, more higher value indicates more better quality of clustering.

As shown in equation (20), we need additional labeled class information for computing NMI score. But it is difficult to obtain data with annotations in many real-world applications. So, we also adopt another widely used metric SC, which describes both similarity among points in intra-cluster and differences among clusters without cluster labels. Let $a(i)$ be the average distance of point i to all other points within the same cluster. Let $b(i)$ be the minimum average distance of point i to other clusters. Equation (22) shows the SC for point i

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{22}$$

The average SC of all points $(\sum_{i=1}^{N} s(i)/N)$ is regarded as the SC value. Thus, SC $\in (-1, 1)$, similar with NMI, higher SC value indicates the obtained clusters are more distinguished clearly.

Table 2. Clustering quality on NMI.

Dataset	*k*-means	*k*-means++	MinMax *k*-means	*k**-means	*k**-means++
Smalldata	0.933 ± 0.01	0.967 ± 0.01	0.932 ± 0.02	**0.991** ± **0.00**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.996** ± **0.00**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
Unbaldata	0.869 ± 0.07	0.926 ± 0.07	0.957 ± 0.03	**0.974** ± **0.05**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.998** ± **0.00**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
LargeUnbal	0.915 ± 0.07	0.916 ± 0.02	0.923 ± 0.02	**0.969** ± **0.03**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.991** ± **0.02**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
Bigdata	0.889 ± 0.01	0.905 ± 0.01	0.904 ± 0.00	**0.913** ± **0.00**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.915** ± **0.00**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
Ecoli	0.616 ± 0.02	0.617 ± 0.03	0.550 ± 0.01	**0.635** ± **0.03**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.617** ± **0.01**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
Pendigits	**0.681** ± **0.01**	0.661 ± 0.01	**0.680** ± **0.01**	0.671 ± 0.01$^{\underline{*}\overline{\dagger}\underline{\ddagger}}$	0.672 ± 0.00$^{\underline{*}\overline{\dagger}\underline{\ddagger}}$
Optdigits	0.718 ± 0.02	**0.724** ± **0.02**	0.708 ± 0.00	**0.719** ± **0.01**$^{\overline{*}\underline{\dagger}\overline{\ddagger}}$	0.711 ± 0.01$^{\underline{*}\underline{\dagger}\overline{\ddagger}}$
Dermatology	0.861 ± 0.05	0.828 ± 0.04	0.775 ± 0.01	**0.863** ± **0.04**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.878** ± **0.02**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
Folio1	0.532 ± 0.02	0.534 ± 0.01	0.540 ± 0.00	**0.550** ± **0.02**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.555** ± **0.02**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
Folio2	0.606 ± 0.01	0.601 ± 0.01	**0.610** ± **0.01**	**0.612** ± **0.01**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	0.601 ± 0.01$^{\underline{*}\overline{\dagger}\underline{\ddagger}}$
CNAE-9	0.414 ± 0.04	0.429 ± 0.03	0.404 ± 0.03	**0.440** ± **0.07**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.451** ± **0.05**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$

NMI: normalized mutual information.

Table 3. Clustering quality on SC.

Dataset	*k*-means	*k*-means++	MinMax *k*-means	*k**-means	*k**-means++
Smalldata	0.741 ± 0.02	0.813 ± 0.02	0.748 ± 0.05	**0.841** ± **0.01**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.849** ± **0.01**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
Unbaldata	0.713 ± 0.10	0.736 ± 0.09	0.771 ± 0.06	**0.805** ± **0.07**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.836** ± **0.00**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
LargeUnbal	0.661 ± 0.10	0.643 ± 0.04	0.662 ± 0.04	**0.739** ± **0.04**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.769** ± **0.02**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
Bigdata	0.554 ± 0.03	0.559 ± 0.02	0.556 ± 0.01	**0.610** ± **0.01**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.577** ± **0.00**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
Ecoli	0.362 ± 0.02	0.365 ± 0.02	0.343 ± 0.01	**0.371** ± **0.01**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.378** ± **0.00**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
Pendigits	0.294 ± 0.02	0.289 ± 0.00	0.276 ± 0.00	**0.321** ± **0.00**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.323** ± **0.00**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
Optdigits	0.175 ± 0.02	**0.186** ± **0.00**	0.172 ± 0.00	**0.183** ± **0.01**$^{\overline{*}\underline{\dagger}\overline{\ddagger}}$	0.177 ± 0.01$^{\overline{*}\underline{\dagger}\overline{\ddagger}}$
Dermatology	0.254 ± 0.03	0.228 ± 0.04	0.204 ± 0.01	**0.269** ± **0.01**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.286** ± **0.02**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
Folio1	0.388 ± 0.04	0.417 ± 0.02	0.407 ± 0.01	**0.426** ± **0.02**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	**0.445** ± **0.01**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
Folio2	0.385 ± 0.02	**0.398** ± **0.02**	0.376 ± 0.04	0.390 ± 0.02$^{\overline{*}\underline{\dagger}\overline{\ddagger}}$	**0.421** ± **0.01**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$
CNAE-9	0.070 ± 0.01	**0.075** ± **0.01**	0.056 ± 0.01	**0.075** ± **0.00**$^{\overline{*}\overline{\dagger}\overline{\ddagger}}$	0.075 ± 0.01$^{\overline{*}\underline{\dagger}\overline{\ddagger}}$

SC: Silhouette coefficient.

For efficiency evaluation, we measure the common metrics of CPU consumption. All experiments are conducted 10 times on each dataset, and we take the average CPU time. Moreover, we discuss the selection of k^* value by analyzing the impact of k^* on effectiveness and efficiency in section "Determining of parameters k* and n."

Alternative algorithm. We compare our algorithm against k-means, k-means++ , and MinMax k-means. We also apply k-means++ initialization method into our k*-means, named k*-means++ algorithm. That is, k*-means++ chooses k^* initial centers using D^2 weighting probability instead of random selection.

Effectiveness evaluation

First, we evaluate the clustering quality of k*-means on 11 datasets when k^* is fixed to $2k$, $n = 2$, and $\theta = 0.05$. Note that we set memory lever $\beta = 0.3$ for MinMax k-means. Tables 2–4 show the experimental results. Asterisk ($*$), dagger ($\dagger$), and double dagger ($\ddagger$) superscripts denote that k*-means or k*-means++ has a statistically significant difference to k-means, k-means++ , and MinMax k-means, respectively, according to the standard deviation. A line above (below) these symbols stands for a higher (lower) average. To distinguish the best result, the top-2 best results on each dataset are shown in bold.

NMI. The results of NMI on 11 datasets are shown in Table 2, our k*-means is superior to k-means, k-means++ , MinMax k-means on all datasets but pendigits. In particular, NMI scores on four synthetic data demonstrate that k*-means has better ability of handling various scenarios compared to other tree algorithms. Even on real-world datasets, k-means also achieves robustly higher NMI score. In particular, it is worth noting that k*-means also works better than the competitors on the high-dimensional text dataset CANE-9. This clearly displays that k^* initial centers can better cover all of clusters and multiple-round merging improves the quality of clustering.

For k-means++ , its probability initialization process improves performance to a certain extend compared to randomly starting of k-means. MinMax

Table 4. Clustering quality on *SSE*.

Dataset	*k*-means	*k*-means++	MinMax *k*-means	*k**-means	*k**-means++
Smalldata	1.08E + 03 ± 383.47	2.77E + 02 ± 112.35	1.21E + 03 ± 878.40	**1.44E + 02 ± 35.14**∗†‡	**1.03E + 02 ± 2.21**∗†‡
Unbaldata	2.22E + 03 ± 1423.32	4.82E + 02 ± 217.85	1.21E + 03 ± 974.43	**3.21E + 02 ± 172.86**∗†‡	**2.44E + 02 ± 0.08**∗†‡
LargeUnbal	2.34E + 03 ± 2721.70	1.47E + 03 ± 555.40	1.39E + 03 ± 555.60	**7.31E + 02 ± 304.61**∗†‡	**5.06E + 02 ± 12.07**∗†‡
Bigdata	2.14E + 03 ± 576.16	9.73E + 02 ± 114.08	**9.02E + 02 ± 26.58**	1.22E + 03 ± 65.59∗†‡	**8.26E + 02 ± 10.22**∗†‡
Ecoli	5.06E−02 ± 0.00	**4.99E−02 ± 0.00**	5.30E−02 ± 0.00	5.07E−02 ± 0.00∗†‡	**4.96E−02 ± 0.00**∗†‡
Pendigits	4.68E + 03 ± 150.85	**4.09E + 03 ± 101.90**	4.72E + 03 ± 171.92	4.72E + 03 ± 190.52∗†‡	**4.60E + 03 ± 17.90**∗†‡
Optdigits	6.69E + 02 ± 18.54	**6.58E + 02 ± 3.55**	6.66E + 02 ± 0.11	**6.64E + 02 ± 11.70**∗†‡	6.73E + 02 ± 14.13∗†‡
Dermatology	9.79E + 00 ± 0.25	**9.65E + 00 ± 0.32**	1.01E + 01 ± 0.01	9.67E + 00 ± 0.19∗†‡	**9.30E + 00 ± 0.18**∗†‡
Folio1	2.69E−03 ± 0.00	**2.34E−03 ± 0.00**	2.50E−03 ± 0.00	2.37E−03 ± 0.00∗†‡	**2.21E−03 ± 0.00**∗†‡
Folio2	2.08E−03 ± 0.00	**1.86E−03 ± 0.00**	1.92E−03 ± 0.00	1.99E−03 ± 0.00∗†‡	**1.78E−03 ± 0.00**∗†‡
CNAE-9	5.77E + 00 ± 0.07	**5.69E + 00 ± 0.06**	**5.76E + 00 ± 0.15**	5.90E + 00 ± 0.19∗†‡	5.93E + 00 ± 0.15∗†‡

SSE: sum of squared error.

Algorithm 1. *K**-means framework

```
1: Randomly choose initial centers $c = \{c_1, c_2, \ldots, c_{k^*}\}$;
2: Perform k-means with parameter $k^*$, get $k^*$ clusters
     $C = \{C_1, C_2, \ldots, C_{k^*}\}$;
   update $c_i = (\Sigma_{X \in C_i} X)/(|C_i|)$;
3: Perform "top-n nearest clusters merging," set
     $k^* = k^* - [n, 2 * n - 1]$;
   update clusters as initial clusters;
4: Repeat steps 2 and 3 until $k^*$ equals to $k$.
```

Algorithm 2. Top-*n* merging method

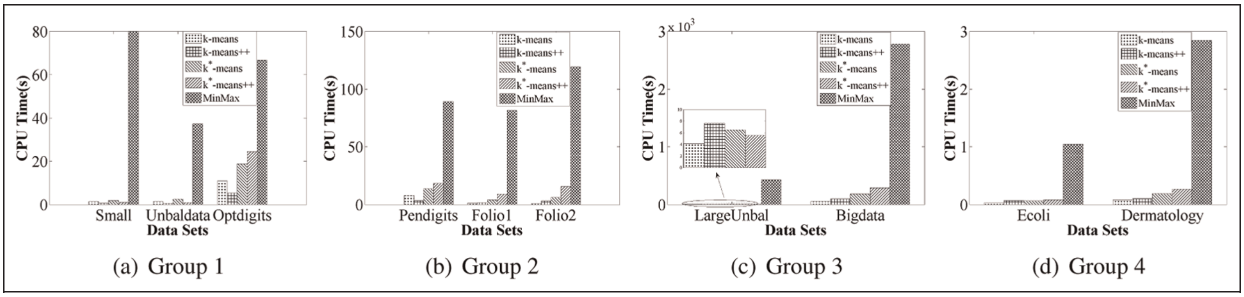
```
Input: $k^*$ clusters, number $N$ ($N \leq (k^* - k)/2$)
Output: $(k * -clusters_{merge})$ clusters
  Calculate distance between each clusters and sort by
    ascending;
  Select clusters associated with top $N$ edges to merge;
  return $(k * -clusters_{merge})$ clusters.
```

Algorithm 3. Optimized update method

```
Input: k clusters, dataset D
Output: k clusters
if # of moved objects $< \theta \cdot |D|$ then
    for each cluster $C_i$ do
        $m_{i}' \leftarrow m_i$;
        $sse_{i'} \leftarrow sse_i$;
        for each $p \in C_i$ do
            if $p$ is closest to $C_j$ then
                move $p$ into $C_j$;
                update $m_{i'}$ and $sse_{i'}$ using Lemma 3;
                update $m_{j'}$ and $sse_{j'}$ using Lemma 2;
            end if
        end for
    end for
    update m and sse of all clusters by their copies;
else perform k-means iteration; count # of moved points;
end if
```

k-means aims to get lower maximum sse_{max} of clusters but may not yield better clusters due to the randomly initial centers. However, k^* (more) centers improve the probability of obtaining best local optima, and our multi-round refining further approaches the optimum gradually. As expected, *k**-means++ also outperforms *k*-means, *k*-means++ , and MinMax *k*-means on most of datasets and achieves the best quality on seven datasets.

SC. As shown in Table 3, again, *k**-means outperforms *k*-means, *k*-means++ , and MinMax *k*-means in term of SC value. In particular, our algorithm exhibits better SC value than *k*-means and MinMax *k*-means on all datasets, and better than *k*-means++ on all datasets but *Optdigits* and *Folio2* data. However, SC values of



(a) Group 1 (b) Group 2 (c) Group 3 (d) Group 4

Figure 6. CPU time of algorithms on 10 datasets: (a) Group 1, (b) Group 2, (c) Group 3, and (d) Group 4.

Algorithm 4. *K**-means algorithm

Input: the number of initial centers K^*; the number of clusters K; dataset D; parameters of Top-*n* method N
Output: K clusters

```
1: arbitrary choose $K^*$ initial centers;
2: distribute $D$ into cluster sets $C$;
3: update mean of each cluster;
4: while not convergence do
5:     for each $C_i \in C$ do
6:         $CS \leftarrow$ pruned clusters of $C_i$;
7:         $C_i.radius' \leftarrow 0$;
8:         for each point $p \in C_i$ do
9:             $d \leftarrow \min distance(p, C_j.\mathbf{m_j})$ where $C_j \in CS$;
10:            if $d < |p - \mathbf{m_i}|$ then
11:                move p into cluster $C_j$;
12:                if $d > C_j.radius'$ then
13:                    $C_j.radius' \leftarrow d$;
14:                end if
15:                update $m'$ of $C_i$, $C_j$ by Lemma 2, 3;
16:            else if $|p - \mathbf{m_i}| > C_i.radius'$ then
17:                $C_i.radius' \leftarrow |p - \mathbf{m_i}|$;
18:            end if
19:        end for
20:    end for
21:    use $m', radius'$ instead of $m, radius$ in each cluster;
22: end while
23: if number of clusters $> K$ then
24:    Top-n(C, N); go to line 4;
25: end if
```

*k**-means are very close to those of *k*-means++ on this two datasets. *K**-means++ also improves clustering quality in 90% of tested cases. This is because careful selection of k^* initial centers further optimizes probability range of best local optima by avoiding centers that are too close to each other compared to random initialization.

SSE. Table 4 shows the *SSE* metrics of algorithms on 11 datasets. *K**-means obtains better (smaller) *SSE* on four datasets and *k*-means++ reaches better results on six datasets. This is because carefully seeding more easily guides *k*-means++ to converge to a better local minima of *SSE* than random initialization. MinMax *k*-means produces a better maximum sse_{max} of cluster but not a better *SSE* value. However, the optimized version *k**-means++ exhibits best *SSE* values on all datasets but optdigits and CNAE-9.

In summary, the above comprehensive experiments confirm the effectiveness of our proposed *k**-means in attaining clusters of better quality. Furthermore, our *k**-means integrated with *k*-means++ initialization robustly achieves best quality clusters in most real-world applications.

Efficiency evaluation

Comparison of efficiency with other algorithms. Next, we evaluate the efficiency of our *k**-means compared against *k*-means, *k*-means++ , and MinMax *k*-means on 10 datasets when $k^* = 2k$, $n = 2$, and $\theta = 0.05$. Figure 6 shows the results of running time on four groups of dataset. We can see that *k*-means costs lowest CPU time and MinMax consumes much more time compared to other algorithms. In addition, we also find that *k*-means++ spends fewer time than *k*-means on some datasets. This is because *k*-means++ carefully chooses initial seeds as its staring centers; thus, the better staring centers may reduce the number of iterations to fast reach the local optima. However, our *k**-means costs same order of magnitudes CPU time compared to *k*-means and *k*-means++ . This is because *k**-means utilizes merging optimization, cluster pruning strategy, and optimized update principle to reduce the computation cost significantly, while MinMax *k*-means adds a maximization step that updates objective with weighting mechanism in each iteration, which increases much more computation cost. Especially when dataset is unbalanced, MinMax *k*-means is more inefficient. In particular, *k**-means and *k**-means++ just cost two- or three-folds CPU time compared to *k*-means on most datasets (Groups 1–4).

Impact of optimization principles. We next analyze the contribution of three optimization principles ("top-*n* nearest clusters merging,""optimized update principle," and "cluster pruning strategy") on efficiency using four datasets (two real and two synthetic datasets). In this

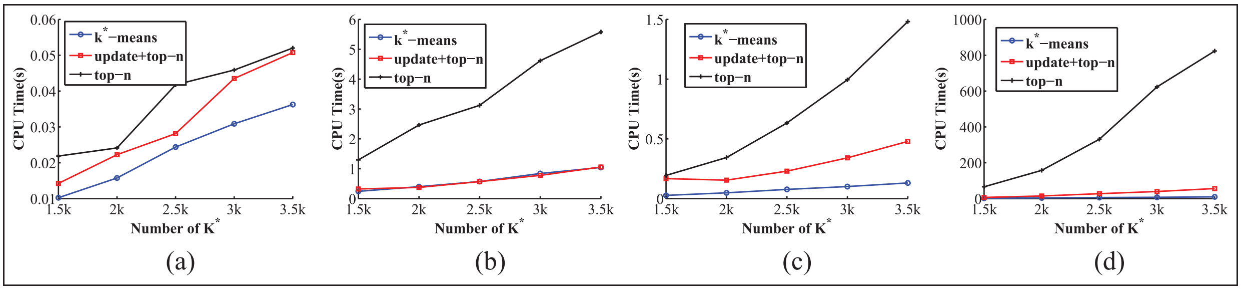


Figure 7. CPU costs in different part of k*-means: (a) dataset dermatology, (b) dataset optdigits, (c) dataset smalldata, and (d) dataset bigdata.

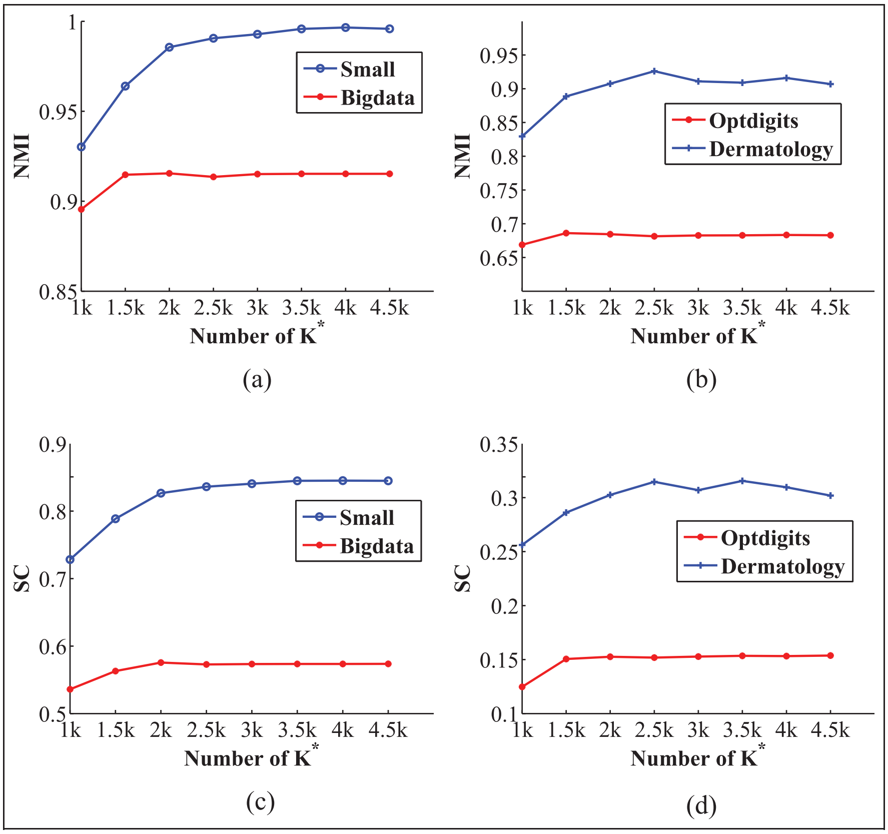


Figure 8. Impact of k^* on NMI and SC: (a) NMI on Group 1, (b) NMI on Group 2, (c) SC on Group 1, and (d) SC on Group 2.

experiment, we demote the k-means armed with "top-n nearest clusters merging" as *top-n*, armed with "optimized update principle," and "top-n nearest clusters merging" as *update* + *top-n* and the completed version as k*-*means*, respectively. Figure 7 shows the CPU time of optimization principles by varying number of k^* from 1.5k to 3.5k. Intuitively, k*-means costs less CPU time than *top-n* and *update* + *top-n* on all datasets. In particular, on *Optdigits* dataset (Figure 7(b)), we observe that the CPU cost of k*-means is approximately equal to *update* + *top-n*. We find that the NMI score (Table 2) and SC (Table 3) of *Optdigits* are smaller than other three datasets by analyzing the property of data. This is because different clusters have a little distance dissimilarity since the data contains high dimensions of 64 attributions, causing "cluster pruning strategy" not work well.

Determining of parameters k^* and n

We conduct experiments to explore the selection of k^* value on four datasets (two synthetic datasets and two real-world datasets). We use *Smalldata* and *Bigdata* to examine the effect of varying k^* unite different datasize.

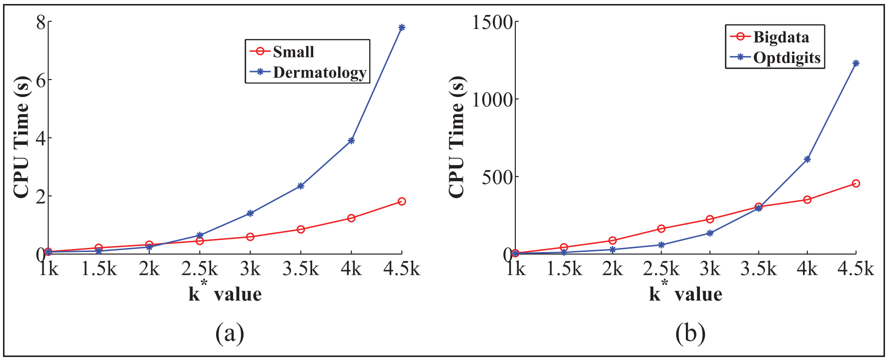


Figure 9. Impact of k^* on CPU time: (a) small and dermatology and (b) bigdata and optdigits.

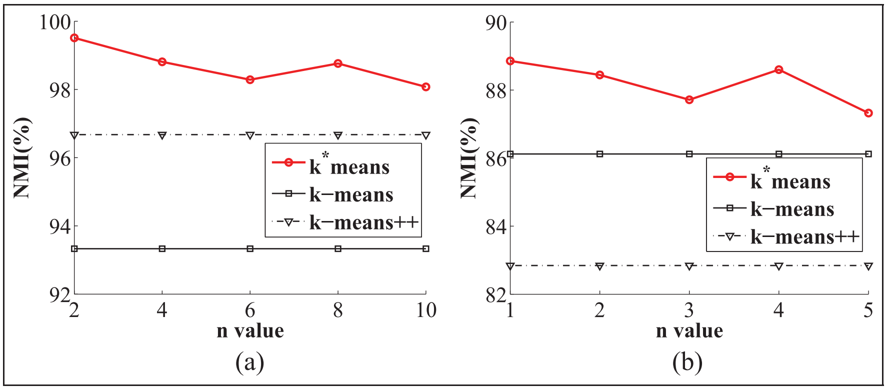


Figure 10. Impact of n on NMI: (a) smalldata and (b) dermatology.

And use *Optdigits* and *Dermatology* from *UCI* to examine the effect of varying k^* unite different amount of attributes. We measure the NMI score and SC value by varying k^* from $1k$ to $4.5k$ with $0.5k$ increment when k is fixed to the number of classes, $n = 2$ and $\theta = 0.05$. The results are shown in Figure 8, Figure 8(a) and (b) are NMI scores on the four datasets and Figure 8(c) and (d) are SC values on the four datasets, respectively. We observe that NMI scores of our proposed k*-means increase as k^* increases, but when $k^* \geq 2k$ the NMI score is no longer growth and remains stable. Similarly, SC value also holds stable when k^* reaches at $2k$ or $2.5k$ on tested datasets. This may be because range of initial seeds could cover all of cluster when k^* enlarges to $2k$, and it does not further improve the quality of clustering if continue to increase k^*. Namely we cannot obtain an better result if k^* is greater than $2k$, that is exactly why we set $k^* = 2k$ in above experiments.

Figure 9 shows the impact of k^* on efficiency. Obviously, k*-means costs more CPU time as k^* increases, this is because k*-means needs more times of merging round to reduce k^* to k when n is fixed. However, k*-means just increases CPU cost linearly before $k^* = 3k$. This also again validates the efficiency of our k*-means.

Finally, we evaluate the effect of parameter n on effectiveness and efficiency. As shown in Figure 10, the number of n has a little impact on clusters results. Moreover, k*-means have higher NMI score than k-means and k-means++ at all tested cases. However, the trend of NMI score decrease as the value of n quickly increases. Figure 11 shows the CPU time of k*-means as the value of parameter n increases. It is easy to see that CPU cost decreases as the n increases. This is because larger value of parameter n would significantly reduce the number of steps that reduce number of clusters from k^* to k. However, we can also achieve a high efficiency as well as good clusters. In addition, the higher value of n means the higher efficiency of k*-means, but n should be smaller than $\lceil k^* - k \rceil$ through our analysis in section "Top-n nearest clusters merging." Consider both of the effectiveness and efficiency, we fix $n = 2$ in evaluation experiments. For $n = 2$, we also can meet many requirements (some datasets only contain a few classes).

Conclusion

In this work, we propose a novel optimized hierarchical clustering method incorporated with three optimization

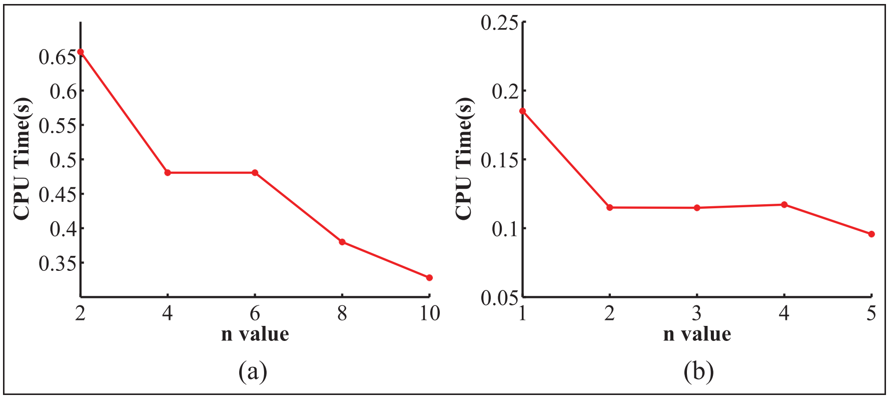


Figure 11. Impact of *n* on CPU time: (a) smalldata and (b) dermatology.

principles, namely "top-*n* nearest clusters merging," "optimized update principle," and "cluster pruning strategy" to achieve both effective and efficient clustering robustly. K^* initial centers effectively improve the probability of obtaining best local optima, and multiround top-*n* nearest clusters merging approaches the optimal result gradually. The first two optimizations update feature values of clusters by previous clusters or moved objects instead of re-computation from scratch. And the pruning strategy reduces significantly the adjusting searching space for each points in *k*-means iteration. Our experimental evaluation with four synthetic datasets and seven real datasets demonstrates that the proposed algorithm exhibits better performance than the state-of-the-art in terms of cluster quality and CPU time in different scenarios. Furthermore, our solution framework beginning with *k*-means++ initialization can further improve the performance, suggesting an alternative optimal solution.

An interesting direction for future work is to leverage modern distributed multi-core cluster of machines for further improving the scalability of our algorithm.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work is based on a conference "Proceedings of the IEEE international conferences on bigdata and cloud computing" and partially supported by the National Natural Science Foundation of China (nos 61403328, 61773331, 61572419, and 61502410), the Key Research & Development Project of Shandong Province (no. 2015GSF115009), and the Graduate Innovation Foundation of Yantai University (no. YDZD1712).

References

1. Emre Celebi M, Kingravi HA and Vela PA. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Syst Appl* 2013; 40(1): 200–210.
2. Dempster AP, Laird NM and Rubin DB. Maximum likelihood estimation from incomplete data via the EM algorithm (with discussion). *J Roy Stat Soc B Met* 1977; 39(1): 1–38.
3. Zhang T, Ramakrishnan R and Livny M. BIRCH: a new data clustering algorithm and its applications. *Data Min Knowl Disc* 1997; 1(2): 141–182.
4. Ester M, Kriegel HP, Sander J, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: *Proceedings of the 2nd international conference on knowledge discovery and data mining (KDD'96)*, Portland, OR, 2–4 August 1996, pp.226–231. New York: ACM.
5. Yu Y, Zhao J, Wang X, et al. Cludoop: an efficient distributed density-based clustering for big data using hadoop. *Int J Distrib Sens N* 2015; 11(6): 579391.
6. Yu Y, Wang H, Wang Q, et al. Density-based cluster structure mining algorithm for high-volume data streams. *J Software* 2015; 26(5): 1113–1128.
7. Wang W, Yang J and Muntz RR. STING: a statistical information grid approach to spatial data mining. In: *Proceedings of the 23rd international conference on very large data bases (VLDB'97)*, Athens, 25–29 August 1997, pp.186–195. San Francisco, CA: Morgan Kaufmann Publishers.
8. Strehl A and Ghosh J. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *J Mach Learn Res* 2002; 3(3): 583–617.
9. MacQueen J. Some methods for classification and analysis of multivariate observations. In: *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, pp.281–297, https://projecteuclid.org/euclid.bsmsp/1200512992
10. Bahmani B, Moseley B, Vattani A, et al. Scalable k-means++ . *Proc VLDB Endowm* 2012; 5(7): 622–633.
11. Bradley PS and Fayyad UM. Refining initial points for k-means clustering. In: *Proceedings of the 15th*

international conference on machine learning (ICML'98), Madison, WI, 24–27 July 1998, pp.91–99. San Francisco, CA: Morgan Kaufmann Publishers.

12. Jain AK. Data clustering: 50 years beyond K-means. *Pattern Recogn Lett* 2010; 31(8): 651–666.
13. Cai Z, Heydari M and Lin G. Clustering binary oligonucleotide fingerprint vectors for DNA clone classification analysis. *J Comb Optim* 2005; 9(2): 199–211.
14. Cai Z, Xu L, Shi Y, et al. Using gene clustering to identify discriminatory genes with higher classification accuracy. In: *Proceedings of the 6th IEEE symposium on bioinformatics and bioengineering (BIBE 2006)*, Arlington, VA, 16–18 October 2006, pp.235–242. New York: IEEE.
15. Cai Z, Goebel R, Salavatipour MR, et al. Selecting genes with dissimilar discrimination strength for sample class prediction. In: *Proceedings of the Asia-Pacific bioinformatics conference (APBC)*, Hong Kong, China, 15–17 January 2007, pp.81–90.
16. Pena JM, Lozano JA and Larranaga P. An empirical comparison of four initialization methods for the *K*-means algorithm. *Pattern Recogn Lett* 1999; 20(10): 1027–1040.
17. Arthur D and Vassilvitskii S. k-means++ : the advantages of careful seeding. In: *Proceedings of the 18th annual ACM-SIAM symposium on discrete algorithms*, New Orleans, LA, 7–9 January, pp.1027–1035. Philadelphia, PA: Society for Industrial and Applied Mathematics.
18. Tzortzis G and Likas A. The MinMax k-Means clustering algorithm. *Pattern Recogn* 2014; 47(7): 2505–2516.
19. Hamerly G. Making k-means even faster. In: *Proceedings of the SIAM international conference on data mining (SDM)*, Columbus, Ohio, USA, 29 April–1 May 2010, pp.130–140.
20. Qi J, Yu Y, Wang L, et al. K*-means: an effective and efficient k-means clustering algorithm. In: *Proceedings of the IEEE international conferences on big data and cloud computing*, Atlanta, GA, 8–10 October 2016, pp.242–249. New York: IEEE.
21. Forgy EW. Cluster analysis of multivariate data: efficiency versus interpretability of classification. *Biometrics* 1965; 21(3): 768–769.
22. Brunsch T and Röglin H. A bad instance for k-means++ . In: M Ogihara and J Tarui (eds) *Theory and applications of models of computation*. Berlin; Heidelberg: Springer, 2011, pp.344–352.
23. Su T and Dy JG. In search of deterministic methods for initializing K-means and Gaussian mixture clustering. *Intell Data Anal* 2007; 11(4): 319–338.
24. Lu JF, Tanga JB, Tanga ZM, et al. Hierarchical initialization approach for K-means clustering. *Pattern Recogn Lett* 2008; 29(6): 787–795.
25. Redmond SJ and Heneghan C. A method for initialising the *K*-means clustering algorithm using *kd*-trees. *Pattern Recogn Lett* 2007; 28(8): 965–973.
26. Bagirov AM, Ugon J and Webb D. Fast modified global *k*-means algorithm for incremental cluster construction. *Pattern Recogn* 2011; 44(4): 866–876.
27. Tzortzis GF and Likas AC. The global kernel *k*-means algorithm for clustering in feature space. *IEEE T Neural Networ* 2009; 20(7): 1181–1194.
28. Kanungo T, Mount DM, Netanyahu NS, et al. An efficient k-means clustering algorithm: analysis and implementation. *IEEE T Pattern Anal* 2002; 24(7): 881–892.
29. Hung MC, Wu J, Chang JH, et al. An efficient *k*-means clustering algorithm using simple partitioning. *J Inf Sci Eng* 2005; 21(6): 1157–1177.
30. Zhao W, Ma H and He Q. Parallel k-means clustering based on MapReduce. In: *Proceedings of the 1st international conference on cloud computing*, Beijing, China, 1–4 December 2009, pp.674–679. Berlin; Heidelberg: Springer.
31. University of California, Irvine. Machine learning repository, center for machine learning and intelligent systems, 2017, http://archive.ics.uci.edu/ml/
32. Rousseeuw PJ. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J Comput Appl Math* 1987; 20: 53–65.