Incremental kernel fuzzy c-means with optimizing cluster center initialization and delivery

Runhai Jiao, Shaolong Liu, Wu Wen and Biying Lin
School of Control and Computer Engineering,
North China Electric Power University, Beijing, China

Abstract

Purpose – The large volume of big data makes it impractical for traditional clustering algorithms which are usually designed for entire data set. The purpose of this paper is to focus on incremental clustering which divides data into series of data chunks and only a small amount of data need to be clustered at each time. Few researches on incremental clustering algorithm address the problem of optimizing cluster center initialization for each data chunk and selecting multiple passing points for each cluster.

Design/methodology/approach – Through optimizing initial cluster centers, quality of clustering results is improved for each data chunk and then quality of final clustering results is enhanced. Moreover, through selecting multiple passing points, more accurate information is passed down to improve the final clustering results. The method has been proposed to solve those two problems and is applied in the proposed algorithm based on streaming kernel fuzzy c-means (stKFCM) algorithm.

Findings – Experimental results show that the proposed algorithm demonstrates more accuracy and better performance than streaming kernel stKFCM algorithm.

Originality/value – This paper addresses the problem of improving the performance of increment clustering through optimizing cluster center initialization and selecting multiple passing points. The paper analyzed the performance of the proposed scheme and proved its effectiveness.

Keywords Big data, Incremental clustering, Initial cluster center, Multiple passing points

Paper type Research paper

1. Introduction

Clustering is an important method for data mining and unsupervised learning. It divides data into different clusters such that data points within the same cluster are more similar to each other than data points in other clusters. Many algorithms for clustering analysis have been developed, such as k-means (MacQueen, 1967), FCM (Rezaee *et al.*, 1998) and so on. They are capable of dealing with small amount of data effectively. However, traditional algorithms have been challenged by big data (Wu *et al.*, 2014). Generally, volume of big data is too large to fit processor's memory. Traditional algorithms often need to work on entire data set in each step, thus it will cost lots of time overhead on exchanging data between memory and external memory. This fact would have made traditional algorithms incapable of processing big data. Hence, algorithms which are fit to handle big data need to be studied.

Some strategies have been designed to address big data clustering in the literature. They can be divided into three different types including sampling clustering, distributed clustering and incremental clustering. In sampling clustering (Provost *et al.*, 1999; Kaufman and Rousseeuw, 2005; Havens *et al.*, 2012), only a small data set

This work was supported by the Fundamental Research Funds for the Central Universities (No. 2014MS29) and Natural Science Foundation of China (No. 61203100). These are gratefully acknowledged.

Kybernetes
Vol. 45 No. 8, 2016
pp. 1273-1291

0368-492X
DOI 10.1108/K-08-2015-0209

sampled from the entire data set needs to be clustered, and then the cluster results are extended to the entire data set without iteration. Distributed clustering divides the large scale data set into small chunks, each data chunk is clustered all alone and then results on each chunk are aggregated at a final run. For example, the online FCM aggregates the results by performing weighted FCM (wFCM) on the cluster centers form each chunk (Hore *et al.*, 2008).

Recently, some researchers have focused on incremental clustering. Literature review shows that incremental clustering is an effective method and has a good performance on big data. Algorithms in this category sequentially divide large scale data into small chunks, cluster each chunk together with points from the previous data chunk in a single pass and finally partition all the data points by results of last data chunk. Single pass FCM (Hore *et al.*, 2007) is a classical incremental clustering algorithm which is based on wFCM. It passes cluster centers from the previous chunk to the next chunk, and then clusters a union set consisting of current data and cluster centers passed from the previous step. Havens *et al.* think that it is not appropriate to pass cluster centers directly between adjacent data chunks (Zhang and Havens, 2013). The cluster centers of the previous chunk are projected into current data chunk as meta-vectors which are linear combinations of data points belonging to the current chunk. Then data chunk at current step is clustered together with meta-vectors. The algorithm might be the latest typical incremental algorithm which greatly improves performance and accuracy. Efforts are also done on learning an effective kernel function to improve clustering results (Baili and Frigui, 2011). Moreover, some researches show that multistage clustering can accelerate convergence and improve clustering quality. Dynamic incremental clustering is also studied (Aaron *et al.*, 2014a, b).

A fact should be noticed that the aim of these algorithms is to improve the quality of the passed cluster information from previous data chunks. Although they achieve this objective to some extent, these researches do not address two problems including optimizing initial cluster center and the number of passing points for each cluster. For the first problem, experiments show that clustering algorithm is sensitive to initial cluster center. Usually, improper initial cluster centers will lead to bad clustering results. For incremental algorithm, the bad clustering information from the previous chunk will be accumulated to the final chunk which may lead to worse results. Many literatures have studied the problem for non-incremental clustering (Katsavounidis *et al.*, 1994; Khan and Ahmad, 2004; Redmond and Heneghan, 2007), but little for incremental clustering. For the second problem, in most incremental clustering, only one passing point for each cluster is selected. However, one passing point may not be sufficient enough to capture the underlying structure of a cluster, which may lead to cluster information loss. Thus multiple passing points should be selected. Yangtao Wang and Lihui Chen propose a new algorithm to solve this problem (Wang *et al.*, 2014). They select multiple medoids for each cluster in a data chunk and establish a mechanism to make use of relationship among those identified medoids as side information to help the final data clustering process. But the algorithm is based on fuzzy clustering, not suitable kernel methods.

To address above two problems, an incremental kernel FCM method with optimizing cluster center initialization and delivery is proposed in this paper. Two strategies are included in the method. The first one is to select appropriate initial cluster centers for each data chunk. We design a simple method based on distance and the characteristic of incremental clustering to optimizing initial cluster center. The second

one is to choose multiple passing points for each cluster at a step to avoid cluster information from being missed. Experiments conducts on synthetic and real-world data sets demonstrate the effectiveness of the proposed method.

The rests of this paper are organized as follows. Section 2 briefly describes some related clustering algorithms, including wFCM, weighted Kernel FCM (wKFCM) and the incremental KFCM. The detail strategy and the procedure of the proposed method are described in Section 3. In order to evaluate the performance of the proposed method, experiments are conducted on several common used data sets, and the results and analysis are presented in Section 4. Finally, Section 5 concludes the paper.

2. Background

2.1 The wFCM algorithm

FCM is one of the most promising fuzzy clustering algorithms (Tamir and Kandel, 2010), which in most cases is more flexible than the hard clustering algorithm, like k-means algorithm. wFCM is an algorithm based on FCM and considers the influence of each data point on clustering results. Given a data set $X = \{x_1, x_2, \ldots, x_n\}$, where $x_i \in R^d$ and d is the dimension of data points. We assume that w_i represents the weight of data point x_i. The wFCM partitions X into C clusters by mining the following objective function:

$$\mathrm{J(U, V)} = \sum_{j=1}^{C} \sum_{i=1}^{n} w_i u_{ji}^m d_{ji}^2 \tag{1}$$

where C is the number of clusters, n the number of data points, $u_{ji} \in U (j = 1, \ldots, C, i = 1, \ldots, n)$ is a fuzzy partition satisfying $\sum_{i=1}^{n} u_{ji} = 1$ $(j = 1, \ldots, C)$ and $u_{ji} \in [0,1]$, m denotes the fuzzy exponent, $v_j \in V$ represents the cluster center, d_{ji}^2 is the distance between data point x_i and cluster center v_j and generally denotes Euclidean distance $\|x_i - v_j\|^2$.

In fact, the object of wFCM is seeking the best fuzzy partition matrix U which makes the Equation (1) minimum. And we can get the best solution in theory when:

$$u_{ji} = \sum_{l=1}^{C} \left(\frac{d_{ji}^2}{d_{li}^2} \right)^{-\frac{1}{m-1}} \tag{2}$$

and:

$$v_j = \frac{\sum_{i=1}^{n} w_i u_{ji}^m x_i}{\sum_{i=1}^{n} w_i u_{ji}^m} \tag{3}$$

Generally, we make use of the iterative method to get U and V. First of all, we initial C cluster centers randomly. And then we update U and V at each step until one of the following three conditions is satisfied:

(1) all elements of U change no longer;

(2) all elements of V change no longer; and

(3) $|J^{k+1}(U,V) - J^k(U,V)| < \varepsilon$, where ε is the given accuracy, k is the iterative step.

2.2 The wkFCM algorithm

When dealing with the linearly non separable data set, wFCM often has a bad performance and low accuracy. The kernel method is used to handle those linearly non

separable data sets. Given a data set $X = \{x_1, x_2, \ldots, x_n\}$ and a project function ϕ, data points in X can be projected into a high-dimensional space called Hilbert space.

Then the kernel function is defined as in the following equation:

$$k(x_i, x_j) = \phi(x_i)\phi(x_j) \tag{4}$$

Thus the distance between x_i and x_j in Hilbert space is:

$$\|\phi(x_i) - \phi(x_i)\|^2 = k_{ii} + k_{jj} - 2k_{ij} \tag{5}$$

where $k_{ij} = k(x_i, x_j)$.

In general, two kinds of kernel function are often used in the kernel method, including popular radial basic function and Gaussian kernel function.

Similar to wFCM, wKFCM can be described as the following optimization problem:

$$Min\, J_{KFCM} = \sum_{j=1}^{C} \sum_{i=1}^{n} w_i u_{ji}^m \|\phi(x_i) - \phi(v_j)\|^2$$

s.t.:

$$\sum_{i=1}^{n} u_{ji} = 1,\ \ j = 1, 2, \ldots, C$$

$$u_{ji} \geqslant 0,\ \ i = 1, 2, \ldots, n\ \ j = 1, 2, \ldots, C \tag{6}$$

where $u_{ji} \in U$ is a fuzzy partition, C is the number of clusters, m denotes the fuzzy exponent, w_i is the weight of the data point x_i.

Again we can solve the above optimization problem by updating fuzzy partition matrix U and cluster centers V in iteration. The initial cluster centers should be given at first. The termination of iteration is same to FCM:

$$u_{ji} = \sum_{l=1}^{C} \left(\frac{\|\phi(x_i) - \phi(v_j)\|}{\|\phi(x_i) - \phi(v_l)\|} \right)^{-\frac{2}{m-1}} \tag{7}$$

$$\emptyset(v_j) = \frac{\sum_{i=1}^{n} w_i u_{ji}^m \phi(x_i)}{\sum_{i=1}^{n} w_i u_{ji}^i} \tag{8}$$

2.3 The incremental KFCM algorithm

The incremental KFCM algorithm is designed for big data clustering (Havens *et al.*, 2012; Zhang and Havens, 2013). It partitions the entire data set X into N subsets, and each time it only needs to cluster a small data set consisting of current data chunk X_t and points passed from previous data chunk X_{t-1} by wKFCM. Hence the information from all previous data chunks is passed down through passing points. Let $A_t = \{a_1^t, \ldots, a_C^t\}$ represent the set of points passed from the previous chunk X_{t-1}. Thus, the incremental KFCM can be seen as a process of wKFCM on X_t and A_t repeatedly for every $t = 1, \ldots, N$.

For a specific t, the weights of data points in X_t are different from the weights of data points in A_t. Usually, the weight of each data point in X_t can be set 1. Let $w_i^{(t,a)}$ represent the weight of the point a_j^t, and it can be calculated as given in the following equation:

$$w_j^{(t,a)} = \sum_{i=1}^{n_t} u_{ji}^t w_i^t + \sum_{k=1}^{C} u_{jk}^a w_k^{(t-1,a)} \tag{9}$$

where $u_{ji}^t \in U^t$ and U^t is the fuzzy partition matrix for data chunk X_t, w_i^t is the weight of data point x_i^t in X_t and $w_i^t = 1$ generally, n_t is the number of data points in X_t, $u_{jk}^a \in U^a$ and U^a is the fuzzy partition matrix for A_t.

In incremental clustering, the key step is to select the set of passing points A_t. A simple method is selecting cluster centers directly as passing points. However, it is not appropriate to pass cluster centers directly between adjacent data chunks (Chitta *et al.*, 2011). Zhang and Havens propose the streaming kernel FCM (stKFCM) algorithm through using a set of meta-vectors to approximate cluster centers of previous data chunk as passing points (Zhang and Havens, 2013). Let $a_j^t = \sum_{i=1}^{n_t} \alpha_{ji}^t \phi(x_i^t)$, then it can be computed by the optimization:

$$min \left\| \phi\left(v_j^{t-1}\right) - a_j^t \right\| \tag{10}$$

where $\phi(v_j^{t-1})$ is the cluster center for data chunk X_{t-1}.

In the kernel clustering algorithm, cluster centers are usually expressed as linear combinations of feature vectors.

Let:

$$\phi\left(v_j^t\right) = \sum_{l=1}^{n_t} \mathrm{q}_{il}^t \phi\left(x_l^t\right) \tag{11}$$

where:

$$\mathrm{q}_{il}^t = \frac{w_l^t \left(u_{il}^t\right)^m + \sum_{k=1}^{q} \alpha_{kl}^t w_k^{(t,\alpha)} \left(u_{ik}^{\alpha}\right)^m}{\sum_{s=1}^{n_t} w_s^t \cdot \left(u_{is}^t\right)^m + \sum_{k=1}^{q} w_k^{(t,\alpha)} \cdot \left(u_{ik}^{\alpha}\right)^m} \tag{12}$$

Thus the solution of the optimization (10) is given in the following equation:

$$\alpha_i^t = \left(K^t\right)^{-1} K^{(t,t-1)} \mathrm{q}_i^t \tag{13}$$

where $K^t = k(x_i^t, x_j^t), i, j = 1, \ldots, n_t, K^{(t,t-1)} = k(x_i^t, x_k^{t-1}), i = 1, \ldots, n_t, k = 1, \ldots, n_{t-1}$ and q_j^t is a column vector.

So the Euclidean distance between the feature vector $\phi(x_i^t)$ and the cluster center $\phi(v_j^t)$ is calculated as given in in the following equation:

$$\left\| \phi\left(x_i^t\right) - \phi\left(v_j^t\right) \right\|^2 = \left(K^t\right)_{ii} + \left(q_j^t\right)^T K^t q_j^t - 2\left(K^t q_j^t\right)_i \tag{14}$$

And the Euclidean distance between a_k^t and the cluster center $\phi(v_j^t)$ is calculated using the following equation:

$$\left\| a_k^t - \phi\left(v_j^t\right) \right\|^2 = \left(\alpha_k^t\right)^T K^t \alpha_k^t + \left(q_j^t\right)^T K^t q_j^t - 2\left(\alpha_k^t\right)^T K^t q_j^t \tag{15}$$

The steps of the incremental KFCM are described as follows:

- Step 1: cluster the first data chunk X_1 by wKFCM.
- Step 2: calculate the set of passing points A_2 and their weights $w^{(2,a)}$.
- Step 3: cluster data chunk X_1 and A_2 by wKFCM.
- Step 4: repeat Steps 2 and 3 until the final data chunk.
- Step 5: finally, the partition of entire data set X can be computed in one single pass.

3. The proposed method

As mentioned in Section 1, in the incremental kernel clustering algorithm, little research has been focused on optimizing initial cluster center of each data chunk and the number of passing points. Generally, when a data chunk is clustered, the initial cluster center is selected randomly or from the cluster center of previous data chunk. Since clustering algorithm is sensitive to initial cluster centers, it is inappropriate to select initial cluster center randomly for the algorithm might converge at a local solution. Besides, it is also inappropriate to directly use cluster center of previous data chunk as the initial cluster center of current data chunk if the structure of the two adjacent data chunks differs largely. If we get bad clustering results when a data chunk is clustered, the bad information will be passed down to the next data chunk and accumulated to the final results, which worsen the final results. Moreover, good initial cluster center can also accelerate the iteration process of clustering. Thus, it is necessary to select optimal cluster center for each data chunk.

Inspired by the idea, we optimize the initial cluster center of each data chunk in our algorithm. Methods are proposed in many literatures to optimize initial cluster centers for non-incremental clustering, basing on density method, distance method (Gotoh, 1982) or optimization method. They can provide a reference for optimizing initial cluster center in incremental kernel clustering. The details of our method are presented in the next subsections.

The passing point plays an important role in the incremental clustering algorithm. It represents all the information of data points in the previous cluster to be passed down. In most incremental clustering algorithm, only one passing point is selected for each cluster. However, when different types of data points are clustered into one cluster, the passing point of this cluster may pass error information to next data chunk and is hard to be reclaimed. Besides, one passing point of each cluster may not be sufficient enough to capture the underlying structure of a cluster especially when data set is large. Thus, to pass more accurate clustering information from previous data chunk, multiple passing points for each cluster are selected in our algorithm.

3.1 Optimized initial cluster centers

The proposed method is based on two assumptions (Cao *et al.*, 2014): (1) the points in the same cluster have more similarity; and (2) the nearby points are likely to have the same label. In incremental clustering, cluster centers of the previous data chunk are usually used as initial cluster centers of current data chunk. It is improper when the structure of new data points differs from the previous data chunks. Thus, to contain the structure of new data points, we update the cluster centers of the previous data chunk by the new data points. Based on assumption (2), data points nearby the cluster centers will be assigned to update their nearest cluster centers. Data points away from

cluster centers will be merged into other clusters. Then all the cluster centers will be merged into k cluster centers based on distance.

Given a data chunk $X_t = \{x_1^t, \ldots, x_{n_t}^t\}$ and a set of cluster centers $V_{t-1} = \{v_1^{t-1}, \ldots, v_C^{t-1}\}$ from previous data chunk. Let D_v denote the distance between cluster centers in V_{t-1} and $G_{max} = \max(D_v)$ represent the maximum distance between clusters. $D_{xv} = (d_{ij}^{xv})_{n_t \times C}$ is a distance matrix whose elements represent the distance between data points in X_t and cluster centers in V_{t-1}. Let $dmin_i$ denote the minimum distance between the data point x_i^t and cluster centers and we can get:

$$dmin_i = \min\left\{d_{ij}^{xv} | j = 1, 2, \ldots, C\right\} \tag{16}$$

Here in our method a data point x_i^t is considered to be nearby the cluster centers if $dmin_i < G_{max}$. Let XN_t be the set of data points nearby the cluster centers and XA_t be the set of data points away from the cluster centers. So we can get:

$$XN_t = \left\{x_i^t | x_i^t \in X_t \text{ and } dmin_i < G_{max}, \quad i = 1, 2, \ldots, n_t\right\} \tag{17}$$

$$XA_t = \left\{x_i^t | x_i^t \in X_t \text{ and } dmin_i \geqslant G_{max}, \quad i = 1, 2, \ldots, n_t\right\} \tag{18}$$

Then the new initial cluster centers would be calculated based on XN_t, XA_t and V_{t-1}. Our method can be divided into three stages. In the first stage, data points in XN_t will be assigned to their nearest cluster center in V_{t-1}. Let $f_i(V_{t-1}, x_j^t)$ be the nearest center in V_{t-1} to x_j^t where $x_j^t \in XN_t$. Thus the cluster centers in this stage can be updated by the following equation:

$$v_i^{t-1} \leftarrow \frac{w_i f_i\left(V_{t-1}, x_j^t\right) + x_j^t}{w_i + 1} \tag{19}$$

$$w_i \leftarrow w_i + 1 \tag{20}$$

where w_i is the weight of center $f_i(V_{t-1}, x_j^t)$.

Then all the data points in XN_t are assigned by formula (19) and we can get a new set of cluster centers $VN_t = \left\{vn_i^t | i = 1, 2, \ldots, C\right\}$ where $vn_i^t = v_i^{t-1}$ and w_i is its weight.

In the second stage, data points in XA_t are processed to find out new structure of data. A simple method based on distance is used in this stage. First of all, we randomly select a data point x_i^t from XA_t and find out all the points nearby x_i^t whose distance to x_i^t within G_{max}. Let XNA_1^t represent the set of data points nearby x_i^t. Then we calculate the center of data points in XNA_1^t and remark the center as va_1^t. The weight of center va_1^t is equal to the number of data points in XNA_1^t. Let $VA_t = VA_t - XNA_1^t$ and repeat previous process to get new centers in VN_t until there is no point left in VN_t. Assuming that $VA_t = \left\{va_i^t | i = 1, 2, \ldots, h\right\}$ is the set of centers which we get in this stage which represents the structure of data set XA_t.

In the third stage, centers calculated in the first two stages will be merged based on distance. Let $V_t = VN_t \cup VA_t$. Based on assumption (2), the two closest centers in V_t are merged into a center and repeat it until the number of centers is reduced to k.

Let $V_t^{(0)}$ be the initial cluster centers of data chunk X_t with the passing points A_t. The steps of calculating the initial cluster centers $V_t^{(0)}$ are presented in Algorithm 1.

Then the initial fuzzy partition matrix can be calculated by the following equations:

$$u_{ji}^{(0,t)} = \left[\sum_{l=1}^{C}\left(\frac{\|\phi(x_i^t) - v_j^{(0,t)}\|}{\|\phi(x_i^t) - v_l^{(0,t)}\|}\right)^{\frac{2}{m-1}}\right]^{-1} \tag{21}$$

$$u_{jk}^{(0,\alpha)} = \left[\sum_{l=1}^{C}\left(\frac{\|a_k^t - v_j^{(0,t)}\|}{\|a_k^t - v_j^{(0,t)}\|}\right)^{\frac{2}{m-1}}\right]^{-1} \tag{22}$$

where $v_j^{(0,t)} \in V_t^{(0)}$and $a_k^t \in A_t$.

Algorithm 1. The method of initialing cluster centers.

Input: data chunk X_t, cluster centers V_{t-1}, the weight of cluster centers w
Output: the initial cluster centers $V_t^{(0)}$
1: calculate the distance matrix D_v and $G_{max} = \max(D_v)$
2: calculate the distance matrix D_{xv} and *dmin* by (16)
3: get the set of data points XN_t by (17) and the set XA_t by (18)
4: for $x_j^t \in XN_t$ do
find out the nearest center $f_i(V_{t-1}, x_j^t) = v_i^{t-1}$
update the ith cluster center v_i^{t-1} by (19)
update its weight w_i by (20)
end for
Let $vn_i^t = v_i^{t-1}$, and $VN_t = \{vn_i^t | i = 1, 2, \ldots, k\}$
5: initialize h = 1
6: while XA_t is not empty do
Pick a data point x from XA_t randomly
Get the set $XNA_h^t = \{$points in XA_t nearby $x\}$
Calculate the center of data points in XNA_h^t as va_h^t and its weight $wa_h^t = |XNA_h^t|$
h = h+1
$VA_t = VA_t - XNA_1^t$
end while
7: let $V_t = VN_t \cup VA_t$
8: while $|V_t| > k$ do
Merge the two closest centers in V_t
end while
9: let $V_t^{(0)} = V_t$

3.2 Multiple passing points

For each data chunk X_t and a set of passing points A_t, assuming that $X_t^{(j)}$ $(j = 1, \ldots, C)$ represents the set of data points in the same cluster in X_t and $A_t^{(j)}$ $(j = 1, \ldots, C)$ represents the set of passing points for cluster j in A_t. Since a data point is assigned to a cluster by a fuzzy partition in fuzzy clustering, the data point in $X_t^{(j)}$ and $A_t^{(j)}$ can be expressed, respectively, by the following equations:

$$x_i^{(j,t)} = u_{ji}^t x_i^t \tag{23}$$

$$a_l^{(j,t)} = u_{jl}^\alpha a_l^t \tag{24}$$

where $u_{ji}^t \in U^t$ is the fuzzy partition of X_t, $u_{jl}^\alpha \in U^\alpha$ is the fuzzy partition of A_t.

To select multiple passing points for each cluster, a data set consisting of $X_t^{(j)}$ and $A_t^{(j)}$ is clustered again by wKFCM. Let $V_t^{(j)} = \{v_1^{(j,t)}, \ldots, v_p^{(j,t)}\}$ be the set of cluster centers where p is the number of passing points for each cluster, $U^{(j,t)}$ is the fuzzy partition matrix of $X_t^{(j)}$, $U^{(j,t)}$ is the fuzzy partition matrix of $A_t^{(j)}$. And then $V_t^{(j)}$ is projected into data chunk X_{t+1} as passing points.

Let:

$$v_i^{(j,t)} = \sum_{l=1}^{n_t} \tilde{q}_{il}^{(j,t)} \phi\left(x_l^t\right) \tag{25}$$

where:

$$\tilde{q}_{il}^{(j,t)} = \frac{w_l^t \left(u_{il}^{(j,t)}\right)^m u_{jl}^t + \sum_{k=1}^{q} \alpha_{kl}^t w_k^{(t,\alpha)} \left(u_{ik}^{(j,t)}\right)^m u_{jk}^{\alpha}}{\sum_{s=1}^{n_t} w_s^t \cdot \left(u_{is}^{(j,t)}\right)^m + \sum_{k=1}^{q} w_k^{(t,\alpha)} \cdot \left(u_{ik}^{\alpha}\right)^m} \tag{26}$$

The passing points can be calculated as the following optimization:

$$min \quad \left\| v_i^{(j,t)} - a_i^{(j,t+1)} \right\| \tag{27}$$

where $a_i^{(j,t+1)} = \sum_{l=1}^{n_{t+1}} \alpha_{li}^{(j,t+1)} \phi(x_l^{(t+1)})$ and $x_l^{(t+1)} \in X_{t+1}$.

The solution of problem (27) is:

$$\alpha_i^{(j,t+1)} = \left(K^t\right)^{\dagger} K^{(t,t+1)} \tilde{q}_i^{(j,t)} \tag{28}$$

where $K^{(t,t+1)} = k(x_i^{t+1}, x_k^t), k = 1, \ldots, n_t,\ i = 1, \ldots, n_{t+1}$.

Thus the passing points are:

$$A_{t+1} = \left\{ a_i^{(j,t+1)} | i = 1, \cdots, p,\ j = 1, \cdots, C \right\} \tag{29}$$

And coefficient matrix of A_{t+1} is:

$$\alpha^{t+1} = \left(\alpha_1^{(1,t+1)}, \ldots, \alpha_p^{(1,t+1)}, \ldots, \alpha_1^{(C,t+1)}, \alpha_p^{(C,t+1)}\right) \tag{30}$$

here $\alpha_i^{(j,t+1)}$ is a column vector.

In our algorithm, passing points are reselected for each cluster, thus the weights of passing points should be recalculated.

The weight for each passing point $a_i^{(j,t+1)}$ is calculated as given in the following equation:

$$w_i^{(j,\alpha)} = \sum_{l=1}^{n_t} u_{il}^{(j,t)} w^t + \sum_{l=1}^{p} u_{il}^{(j,\alpha)} w^{(t,\alpha)} \tag{31}$$

3.3 The procedure of proposed algorithm

In this section, a new algorithm about the application of the above two strategies is proposed to improve the performance of stKFCM. The steps of our algorithm are fully described in Algorithm 1:

Algorithm 2

Input: number of clusters – C; fuzzy exponent – m;
X = {X_0, X_1, …, X_{N-1}};kernel function – k;
Total number of passing points – r,
number of passing points for each cluster – p

1: compute $K^0 = k(X_0, X_0)$
$U^0 = \mathrm{KFCM}(C, m, K^0)$
for j = 1 to C do
$K^{(j,0)} = k(u_j^0 X_0, u_j^0 X_0)$
$U^{(j,0)} = \mathrm{KFCM}(C, m, K^0)$
$\tilde{q}_j^{(j,0)} = u_j^{(j,0)} / |u_j^{(j,0)}|$
Compute $\alpha_i^{(j,1)} (i = 1, \ldots, p)$ by (28)
Compute $w_i^{(j,\alpha)} (i = 1, \ldots, p)$ by (31)
2: collect all $\alpha_i^{(j,1)}$ as element of α^1
3: collect all $w_i^{(j,\alpha)}$ as element of $w^{(1,\alpha)}$
4: for t = 1 to N−1 do
$K^t = k(X_t, X_t), K^{(t,t-1)} = k(X_t, X_{t-1})$
Optimizing initial cluster center $V_t^{(0)}$ by Algorithm 1
Compute $u_{ji}^{(0,\alpha)}$ by (22)
Compute $u_{ji}^{(0,t)}$ by (21)
Set $u_{ji}^{\alpha} = u_{ji}^{(0,\alpha)}$, $u_{ji}^{t} = u_{ji}^{(0,t)}$
while any u_{ji}^{α} or u_{ji}^{t} changes do
Compute q_j^t with (12)
for $i = 1, \ldots, n_t, j = 1, \ldots, C$ do
$u_{ji}^t = \sum_{l=1}^{C} \left(\frac{\|\phi(x_i^t) - \phi(v_j^t)\|^2}{\|\phi(x_i^t) - \phi(v_l^t)\|^2} \right)^{-\frac{1}{m-1}}$
where $\|\phi(x_i^t) - \phi(v_j^t)\|^2$ is computed by (14)
end for
for $i = 1, \ldots, r, j = 1, \ldots, C$ do
$u_{ji}^{\alpha} = \left[\sum_{l=1}^{C} \left(\frac{\|a_i^t - \phi(v_j^t)\|}{\|a_i^t - \phi(v_l^t)\|} \right)^{\frac{2}{m-1}} \right]^{-1}$
where $\|a_i^t - \phi(v_j^t)\|^2$ is computed by (15)
end for
for j = 1 to C do
$K^{(j,t)} = k(u_j^t X_0, u_j^t X_0)$
$U^{(j,t)} = \mathrm{KFCM}(C, m, K^{(j,\ t)})$
$\tilde{q}_j^{(j,t)} = u_j^{(j,t)} / |u_j^{(j,t)}|$
Compute $\alpha_i^{(j,t+1)} (i = 1, \ldots, p)$ by (28)
Compute $w_i^{(j,\alpha)} (i = 1, \ldots, p)$ by (31)
end for
Collect all $\alpha_i^{(j,t+1)}$ as element of α^{t+1}
Collect all $w_i^{(j,\alpha)}$ as element of $w^{(t+1,\alpha)}$
end while
Compute q_j^t with (12)
5: the fuzzy partition of the entire data set X is computed by the following steps:
for i = 1, …, n, $j = 1, \ldots, C$ do
Compute u_{ji} with (7)
end for

4. Experiments

In this section, experimental studies of the proposed methods are conducted on four typical data sets 2D15, 2D50, MNIST and Forest. The algorithm is implemented in

Matlab, and the experiments were run on a PC with four cores of Intel i7 and 8G of memory. Experiments are conducted, respectively, for the method of optimizing initial cluster centers and the method of selecting multiple passing points. To simplify the experiments, the size of each data chunk is the same and the value of fuzzy exponent m is fixed at 1.7 which is recommended by Zhang and Havens. The accuracy is evaluated by three popular criterions including purity, F-measure and run time.

4.1 Data sets

(1) 2D15: this is a synthetic data set which is composed of 5,000 two-dimensional points with 15 classes. The distribution of the points is showed in Figure 1. We use a Gaussian kernel function with width of 1 on this data set. And two passing points are selected for a cluster in each data chunk. The feature vectors are normalized the feature vectors by subtracting the minimum and then dividing by the subsequent maximum so that value of each dimension is between 0 and 1.

(2) 2D50: this is also a synthetic data set which is composed of 7,500 two-dimensional feature vectors with 50 classes. A Gaussian kernel function with width 1 is used on this data set. The feature vectors are normalized with the same way to 2D15. The number of passing points is two.

(3) MNIST: this data set is composed of 70,000 784-dimensional feature vectors with ten classes. It is collected from handwriting digits from 0 to 9 by the National Institute of Standards and Technology (NIST). A Gaussian kernel function with width 1.5 is used on this data set. Two passing points is selected when the data chunk size is small and four passing points is selected when the data chunk size is large.

(4) Forest: this data set is from US Geological Survey and US Forest Service (USFS). It is composed of 581,012 objects and has seven classes. Each object is represented as a 54 dimensional feature vector consisting of ten quantitative

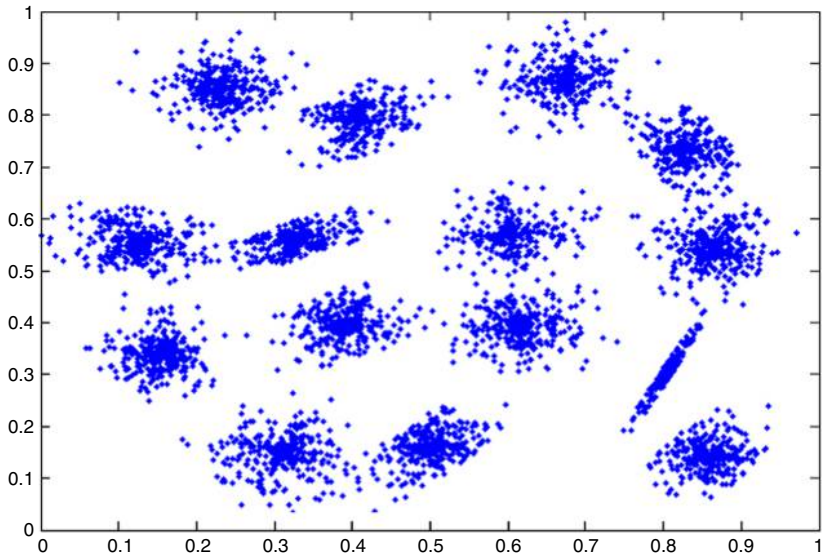

Figure 1.
Distribution of 2D15

variables and 44 binary variables. These features are collected from meter cells of forest by the USFS. A Gaussian kernel function with width of 1 is used on this data set. Two passing points is selected when the data chunk size is small and five passing points is selected when the data chunk size is large.

4.2 Data partition

For incremental clustering algorithm, a big data set is usually partitioned into a series of smaller data chunks that are manageable and thus one data chunk is handled at a time. Generally, there are three data partitioning methods, including sequential non-overlapping partitioning method, round robin partitioning method and sampling without replacement method (Farrash and Wang, 2013).

In our experiments, we use the sampling without replacement method to partition the data sets. The size of each data chunk is same. For data set 2D15 and 2D50, we chose 10, 20, 25 and 50 percent of the entire data set size, respectively, as each data chunk size. For data set MNIST, we chose 0.75, 1, 2, 2.5, 4, 5 and 10 percent of the entire data set size, respectively, as each data chunk size. For data set Forest, we chose 0.1, 0.2, 0.5, 1 and 2 percent of the entire data set size, respectively, as each data chunk size.

4.3 Data preprocessing

When processing image data sets, we hope that the label of image is the same even when the image is rotated. In our experiment, the MNIST is an image data set of handwriting digits. We find that the distance between objects of the same label may be larger than the distance between objects of different labels. For example, if you were to take a digit in MNIST and rotate it at a certain angle, the distance could be very large. That is to say, objects in the same label have less similarity because of the rotation of the digit and it will make the clustering algorithm based on distance invalid. Thus, a method of the feature extraction is used to extract rotation-invariant features for MNIST. The method of preprocessing used in our experiment is Sparse Auto Encoder and 100 features are extracted to be used in clustering.

4.4 Evaluation criterion

Three popular external evaluation criterions, Purity (Mei and Chen, 2012), F-measure (Larsen and Aone, 1999) and Run time are used to evaluate the clustering results in our experiments.

4.4.1 Purity. Purity, also called clustering accuracy, is a measure to evaluate the quality of clustering results. If we refer class as the ground truth and cluster as the results of the clustering algorithm, purity reflects the degree of a cluster only consisting of data points in one class. The value of purity for a cluster is given by the ratio between the amount of right assignments in this cluster and the size of this cluster. The value of purity for each cluster can be calculated as given in the following equation:

$$Purity(D_j) = \max\left(Pr_j(c_i)\right),\ i = 1, 2, \ldots, C \tag{32}$$

where C is the number of clusters or classes, D_j is the set of data points in cluster j, c_i is the class label i, $Pr_j(c_i)$ is the ratio between the number of data points belonging to c_i and number of data set D_j. $Pr_j(c_i)$ is calculated after obtaining one-to-one match between cluster and class.

Thus the total purity of whole clustering results is calculated using the following equation:

$$Purity_{total}(D) = \sum_{j=1}^{C} \frac{|D_j|}{|D|} Purity(D_j) \tag{33}$$

where D is the entire data set, $|D|$ represents cardinal number of data set D and $|D_j|$ denotes cardinal number of data set D_j.

From Equations (32) and (33), we can know that the value of purity is a real number between 0 and 1. The higher value the purity, the better the clustering results.

4.4.2 F-measure. F-measure, also called F-score, is based on precision and recall in information retrieval which is calculated using the following equation:

$$F - measure = 2 \cdot precision \cdot recall / (precision + recall) \tag{34}$$

For each cluster j and each class c_i:

$$precision(c_i, j) = n_{c_i,j} / n_j \tag{35}$$

$$recall(c_i, j) = n_{c_i,j} / n_{c_i} \tag{36}$$

Thus the total F-measure of the whole clustering results is:

$$F - measure_{total} = \sum_{i=1}^{C} \frac{n_{c_i}}{n} \max(F - measure(c_i, j)) \tag{37}$$

where *F-measure*(c_i, j) can be calculated by (34), n_j is the number of objects in cluster j, n_{c_i} is the number of objects in class c_i, and $n_{c_i,j}$ is the number of common objects in cluster j and in class c_i.

4.4.3 Run time. Run time is an important criterion especially for big data clustering. The time of complexity of our algorithm is not only compared with stKFCM but also with KFCM on the entire data set.

4.5 Results and analysis

4.5.1 Results of purity and F-measure and analysis. First of all, we compare our method with stKFCM using the four data sets 2D15, 2D50, MNIST and Forest through the two evaluation criterion purity and F-measure. Results of purity and F-measure are shown in Tables I-IV, respectively. Because the accuracy of clustering algorithm is affected by the initialization, it is difficult to guarantee the quality of an algorithm just through a single time experiment. Therefore, the mean, variance, minimum and maximum of purity and F-measure are calculated over 20 trials. The mean value reflects the average performance of the clustering algorithm and the standard deviation value reflects the robustness of the clustering algorithms. Table I describes the purity results on data set 2D15 with different data chunk sizes and Table I is the F-measure results. Table II describes the results of purity and F-measure on data set 2D50.

From Tables I and II, we can see that our algorithm always performs better than stKFCM with different data chunk sizes. In Table I, for data set 2D15, it shows

	Algorithm			
	Our algorithm		stKFCM	
Chunk size	Mean Min.	Var. Max.	Mean Min.	Var. Max.
(a) Purity on 2D15				
10%	0.9926	0.0004	0.9793	0.0009
	0.9288	0.9999	0.9270	0.9999
20%	0.9750	0.0015	0.9726	0.0019
	0.8951	0.9998	0.7924	0.9996
25%	0.9787	0.0013	0.9637	0.0018
	0.8934	0.9998	0.8382	0.9996
50%	0.9796	0.0013	0.9612	0.0019
	0.8656	0.9995	0.8504	0.9990
(b) F-measure on 2D15				
10%	0.9929	0.0003	0.9813	0.0007
	0.9290	0.9999	0.9246	0.9999
20%	0.9900	0.0005	0.9658	0.0009
	0.9286	0.9998	0.9218	0.9994
25%	0.9785	0.0011	0.9616	0.0014
	0.8743	0.9996	0.8688	0.9996
50%	0.9715	0.0011	0.9681	0.0011
	0.9156	0.9994	0.8806	0.9992

Table I.
Purity and F-measure on 2D15

	Algorithm			
	Our algorithm		stKFCM	
Chunk size	Mean Min.	Var. Max.	Mean Min.	Var. Max.
(a) Purity on 2D50				
10%	0.9593	0.0003	0.7876	0.0011
	0.9227	0.9832	0.7168	0.8413
20%	0.9446	0.0001	0.8113	0.0007
	0.9231	0.9632	0.7656	0.8665
25%	0.9410	0.0004	0.8016	0.0005
	0.9021	0.9793	0.7637	0.8416
50%	0.9231	0.0005	0.8311	0.0006
	0.8705	0.9715	0.7755	0.8747
(b) F-measure on 2D50				
10%	0.9577	0.0003	0.8002	0.0008
	0.9234	0.9832	0.7399	0.8449
20%	0.9393	0.0001	0.8178	0.0006
	0.9202	0.9629	0.7752	0.8568
25%	0.9363	0.0004	0.8087	0.0005
	0.9012	0.9794	0.7734	0.8483
50%	0.9198	0.0005	0.8303	0.0007
	0.8658	0.9714	0.7717	0.8714

Table II.
Purity and F-measure on 2D50

that, though stKFCM has gained very high accuracy on 2D15, our algorithm still obtains over 1 percent improvement for most kinds of data chunk sizes. From Table II, for data set 2D50, we can see that the improvements of purity are over 9 percent for different data chunk sizes, even up to 15 percent, which demonstrates

	Algorithm			
	Our algorithm		stKFCM	
Chunk size	Mean Min.	Var. Max.	Mean Min.	Var. Max.
(a) Purity on MNIST				
0.75%	0.1943	0.0001	0.1882	0.0001
	0.1808	0.2219	0.1662	0.2083
1%	0.2346	0.0001	0.2202	0.0002
	0.2113	0.2628	0.2086	0.2385
2%	0.2469	0.0002	0.2386	0.0002
	0.2299	0.2836	0.2173	0.2652
2.5%	0.2610	0.0003	0.2514	0.0004
	0.2368	0.2960	0.2208	0.2771
4%	0.2711	0.0001	0.2530	0.0001
	0.2438	0.2974	0.2372	0.2772
5%	0.2787	0.0003	0.2609	0.0003
	0.2480	0.3154	0.2408	0.2993
10%	0.2836	0.0002	0.2614	0.0003
	0.2512	0.3107	0.2223	0.2876
(b) F-measure on MNIST				
0.75%	0.3528	0.0003	0.2750	0.0007
	0.3270	0.3918	0.2411	0.3123
1%	0.2665	0.0001	0.2269	0.0001
	0.2486	0.2814	0.2163	0.2384
2%	0.3386	0.0001	0.2503	0.0002
	0.3062	0.3546	0.2362	0.2763
2.5%	0.3348	0.0003	0.2518	0.0001
	0.3004	0.3701	0.2296	0.2684
4%	0.3697	0.0007	0.2604	0.0008
	0.3501	0.4334	0.2403	0.2853
5%	0.2901	0.0004	0.2533	0.0001
	0.2372	0.3349	0.2344	0.2677
10%	0.2751	0.0005	0.2492	0.0002
	0.2293	0.3411	0.2378	0.2959

Table III.
Purity and F-measure on MNIST

good performance of our algorithm. Also, variance values are smaller than stKFCM which indicates that our algorithm has better robustness to different data partition. The results of F-measure given in Tables I and II show the similar pattern as purity.

Table III describes the results of purity and F-measure on data set MNIST. In Table III, results show that both our algorithm and stKFCM do not match well to the ground truth for the data set MNIST. From Table III, for the mean value, we can see that our algorithm outperforms stKFCM and the average improvement is only about 1.5 percent in terms of purity. In Table III, we can see that our algorithm has obvious improvement over the stKFCM, especially when the data chunk is 4 percent. The maximum improvement can be reached at 10 percent. For the variance value, we can see that there is almost no difference between our algorithm and the stKFCM both for the purity and F-measure.

The results of purity and F-measure on data set Forest are shown in Table IV. In Table IV, although the improvements are small, the purity of our algorithm is better

	Algorithm			
	Our algorithm		stKFCM	
Chunk size	Mean Min.	Var. Max.	Mean Min.	Var. Max.
(a) Purity on forest				
0.1%	0.5142	0.0003	0.4985	0.0002
	0.5107	0.5227	0.4975	0.5005
0.2%	0.4909	0.0001	0.4801	0.0005
	0.4867	0.5207	0.4786	0.5101
0.5%	0.4878	0.00001	0.4808	0.00001
	0.4860	0.4913	0.4800	0.4895
1%	0.4907	0.0001	0.4894	0.0001
	0.4874	0.5143	0.4874	0.5011
2%	0.4886	0.00003	0.4806	0.00006
	0.4879	0.4926	0.4790	0.4898
(b) F-measure on forest				
0.1%	0.3959	0.0005	0.3496	0.0025
	0.3622	0.4234	0.2802	0.4116
0.2%	0.3724	0.0012	0.3638	0.0015
	0.2768	0.3969	0.2965	0.3820
0.5%	0.3844	0.0002	0.3663	0.0006
	0.3533	0.4113	0.3047	0.3921
1%	0.3825	0.0008	0.2958	0.0009
	0.3595	0.4255	0.2596	0.3395
2%	0.3908	0.0001	0.3554	0.0018
	0.3794	0.4098	0.2886	0.3905

Table IV.
Purity and F-measure on forest

than the stKFCM under average meaning for different chunk sizes. In Table IV, we can see that the average F-measure of our algorithm is larger than the stKFCM and the average improvements of our algorithm are about 4 percent. Also, our algorithm outperforms stKFCM in terms of robustness.

In general, these results show that our algorithm performs well. The performance of our algorithm is always better than stKFCM on both synthetic data sets and real-world data sets. Moreover, our algorithm has a better robustness than stKFCM. In a word, the results demonstrate the effectivity of our algorithm.

4.5.2 Run time analysis. The run time of our algorithm and the stKFCM on the four data sets is shown in Figure 2. Figure 2 shows that more time is spent in our algorithm because of the time consumption of selecting multiple passing points. For 2D15 and 2D50, though the difference of our algorithm and stKFCM is obvious, the value of running time is very small. When the volume of the data is small, most of the time was spent on calculating the kernel matrix. Our algorithm adds some steps of calculating the kernel matrix in the process of selecting multiple passing points in the cost of more time. However, we should notice that the difference is not obvious between our algorithm and the stKFCM algorithm on the time for MNIST and Forest. That is because when the volume of the data is large, most of the time spent on computing the inverse matrix K^t both for the two kinds of algorithm. The time of computing the inverse matrix K^t is the same in our algorithm and the stKFCM algorithm. Thus it is reasonable to believe that the time difference between these two algorithms will be negligible when the data volume is big.

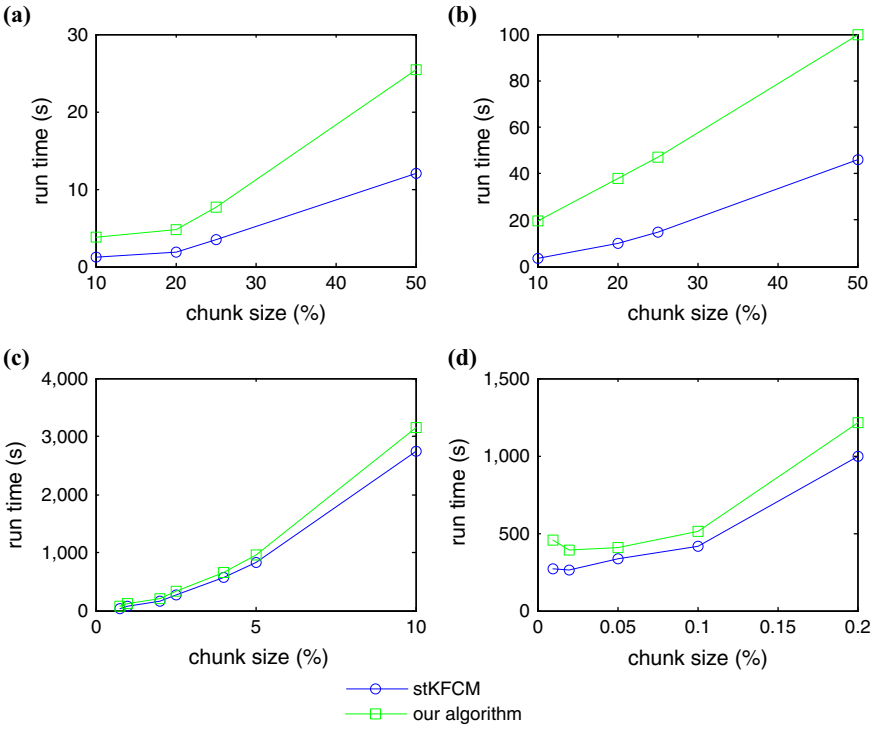

Figure 2.
The run time of data set

Notes: (a) 2D15; (b) 2D50; (c) MNIST; (d) Forest

5. Conclusion

The incremental clustering algorithm is effective for big data. In this paper, we propose two strategies to improve the performance of the incremental kernel clustering. First, a simple method is used to optimize initial cluster center for each data chunk. Then, multiple passing points are selected to pass more accurate clustering information to subsequent data chunk. The two strategies are used to improve the performance of the stKFCM algorithm and experimental results demonstrate its good performance and robustness by comparing with stKFCM on some data sets.

As a scope for future research, we may improve the method of optimizing initial cluster centers. We also notice that the number of multiple passing points for each cluster is the same and fixed in this paper. In the future, an adaptive method for selecting multiple passing points for each cluster may be studied. Also, larger data sets should be collected and tested to study those problems.

References

Aaron, B., Tamir, D.E., Rishe, N.D. and Kandel, A. (2014a), "Dynamic incremental K-means clustering", *Proceedings of the 2014 International Conference on Computational Science and Computational Intelligence(CSCI) in Las Vegas, NV, IEEE, Washington, DC*, pp. 308-313.

Aaron, B., Tamir, D., Rishe, N. and Kandel, A. (2014b), "Dynamic incremental fuzzy c-means clustering", *Proceedings of The Sixth International Conferences on Pervasive Patterns and Applications in Venice, IARIA XPS Press*, pp. 28-37.

Baili, N. and Frigui, H. (2011), "Fuzzy clustering with multiple kernels", *2011 IEEE International Conference on Fuzzy Systems (FUZZ) IEEE in Taipei, Washington, DC*, pp. 490-496.

Cao, J., Chen, P., Dai, Q.Y. and Ling, W.K. (2014), "Local information-based fast approximate spectral clustering", *Pattern Recognition Letters*, Vol. 38 No. 3, pp. 63-69.

Chitta, R., Jin, R., Hanens, T.C. and Jain, A. (2011), "Approximate kernel k-means: solution to large scale kernel clustering", *Proceedings of ACM SIGKDD Conference Knowledge Discovery and Data Mining in San Diego, CA, ACM, New York, NY*, pp. 895-903.

Farrash, M. and Wang, W. (2013), "How data partitioning strategies and subset size influence the performance of an ensemble", *2013 IEEE International Conference on Big Data, IEEE in Silicon Valley, CA, IEEE, Washington, DC*, pp. 42-49.

Gotoh, O. (1982), "An improved algorithm for matching biological sequences", *Journal of Molecular Biology*, Vol. 162 No. 3, pp. 705-708.

Havens, T.C., Bezdek, J.C., Leckie, C. and Palaniswami, M. (2012), "Fuzzy c-means algorithms for very large data", *IEEE Transactions on Fuzzy Systems*, Vol. 20 No. 6, pp. 1130-1146.

Hore, P., Hall, L.O. and Goldgof, D.B. (2007), "Single pass fuzzy c-means", *Proceedings of IEEE International Conference on Fuzzy System in London, IEEE, Washington, DC*, pp. 1-7.

Hore, P., Hall, L.O., Goldgof, D.B. and Cheng, W. (2008), "Online fuzzy C-means", *Fuzzy Information Processing Society, 2008. NAFIPS 2008. Annual Meeting of the North American in New York, USA, IEEE, Washington, DC*, pp. 1-5.

Katsavounidis, I., Jay Kuo, C.C. and Zhang, Z. (1994), "A new initialization technique for generalized Lloyd iteration", *IEEE Signal Processing Letters*, Vol. 1 No. 10, pp. 144-146.

Kaufman, L. and Rousseeuw, P. (2005), *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley-Blackwell, New York, NY.

Khan, S.S. and Ahmad, A. (2004), "Cluster center initialization algorithm for K-means clustering", *Pattern Recognition Letters*, Vol. 25 No. 11, pp. 1293-1302.

Larsen, B. and Aone, C. (1999), "Fast and effective text mining using linear time document clustering", *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining in San Diego, CA, ACM, New York, NY*, pp. 16-22.

MacQueen, J.B. (1967), "Some methods for classification and analysis of multivariate observations", *Proceedings of 5th Berkeley Symposium on Probability and Statistics in Berkeley, United States, University California Press, Berkeley, CA*, pp. 281-297.

Mei, J.P. and Chen, L. (2012), "A fuzzy approach for multitype relational data clustering", *IEEE Transactions on Fuzzy Systems*, Vol. 20 No. 2, pp. 358-371.

Provost, F., Jensen, D.D. and Oates, T. (1999), "Efficient progressive sampling", *Proceedings of The Fifth KDDM in San Diego, USA, ACM Press, New York, NY*, pp. 22-32.

Redmond, S.J. and Heneghan, C. (2007), "A method for initializing the K-means clustering algorithm using kd-trees", *Pattern Recognition Letters*, Vol. 28 No. 8, pp. 965-973.

Rezaee, M.R., Lelieveldt, B.P.F. and Reiber, J.H.C. (1998), "A new cluster validity index for the fuzzy c-means", *Pattern Recognition Letters*, Vol. 19 Nos 3-4, pp. 237-246.

Tamir, D.E. and Kandel, A. (2010), "The pyramid fuzzy c-means algorithm", *International Journal of Computational Intelligence in Control*, Vol. 2 No. 2, pp. 270-302.

Wang, Y.T., Chen, L.H. and Mei, J.P. (2014), "Incremental fuzzy clustering with multiple medoids for large data", *IEEE Transactions on Fuzzy Systems*, Vol. 22 No. 6, pp. 1557-1568.

Wu, X.D., Zhu, X.Q., Wu, G.Q. and Ding, W. (2014), "Data mining with big data", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 26 No. 1, pp. 97-107.

Zhang, Z. and Havens, T.C. (2013), "Scalable approximation of kernel fuzzy c-means", *2013 IEEE International Conference on Big Data IEEE in Silicon Valley, CA, IEEE, Washington, DC*, pp. 161-168.

About the authors

Runhai Jiao received his PhD Degree in Computer Science from the Beihang University in 2008, and now is an Associate Professor in the North China Electric Power University. His research interests include data mining, load forecasting, power transmission network planning and distribution network automation. He has published more than 20 papers in the journals and international meetings. Runhai Jiao is the corresponding author and can be contacted at: runhaijiao@ncepu.edu.cn

Shaolong Liu is a Postgraduate in the North China Electric Power University and will receive his Master's Degree in 2016. He majors in computer application technology and his research interests include data mining on big data and distributed computing.

Wu Wen is a Postgraduate in the North China Electric Power University and will receive his Master's Degree in 2017. His current research area focuses on data mining and machine learning.

Biying Lin is a Professor in the North China Electric Power University. Her interesting research fields include distribution network automation, distributed computing and computer network.