Fuzzy c-Means Algorithms for Very Large Data

Timothy C. Havens, *Senior Member, IEEE*, James C. Bezdek, *Life Fellow, IEEE*, Christopher Leckie, Lawrence O. Hall, *Fellow, IEEE*, and Marimuthu Palaniswami, *Fellow, IEEE*

***Abstract*—*Very large* (VL) data or big data are any data that you cannot load into your computer's working memory. This is not an objective definition, but a definition that is easy to understand and one that is practical, because there is a dataset too big for any computer you might use; hence, this is VL data for you. Clustering is one of the primary tasks used in the pattern recognition and data mining communities to search VL databases (including VL images) in various applications, and so, clustering algorithms that scale well to VL data are important and useful. This paper compares the efficacy of three different implementations of techniques aimed to extend *fuzzy c-means* (FCM) clustering to VL data. Specifically, we compare methods that are based on 1) sampling followed by noniterative extension; 2) incremental techniques that make one sequential pass through subsets of the data; and 3) kernelized versions of FCM that provide approximations based on sampling, including three proposed algorithms. We use both loadable and VL datasets to conduct the numerical experiments that facilitate comparisons based on time and space complexity, speed, quality of approximations to batch FCM (for loadable data), and assessment of matches between partitions and ground truth. Empirical results show that random sampling plus extension FCM, bit-reduced FCM, and approximate kernel FCM are good choices to approximate FCM for VL data. We conclude by demonstrating the VL algorithms on a dataset with 5 billion objects and presenting a set of recommendations regarding the use of different VL FCM clustering schemes.**

***Index Terms*—Big data, fuzzy c-means (FCM), kernel methods, scalable clustering, very large (VL) data.**

I. Introduction

CLUSTERING or cluster analysis is a form of exploratory data analysis in which data are separated into groups or subsets such that the objects in each group share some similarity. Clustering has been used as a preprocessing step to separate data into manageable parts [1], [2], as a knowledge discovery tool [3], [4], for indexing and compression [5], etc., and there are many good books that describe its various uses [6]–[10]. The most popular use of clustering is to assign labels to unlabeled data—data for which no preexisting grouping is known. *Any field that uses or analyzes data can utilize clustering*; the problem domains and applications of clustering are innumerable.

TABLE I
Huber's Description of Dataset Sizes [11], [12]

Bytes	10^6	10^8	10^{10}	10^{12}	$10^{>12}$
"size"	medium	large	huge	monster	**VL**

The ubiquity of personal computing technology, especially mobile computing, has produced an abundance of staggeringly large datasets—Facebook alone logs over 25 terabytes (TB) of data per day. Hence, there is a great need to cluster algorithms that can address these gigantic datasets. In 1996, Huber [11] classified dataset sizes as in Table I.[1] Bezdek and Hathaway [12] added the *very large* (VL) category to this table in 2006. Interestingly, data with $10^{>12}$ objects are still unloadable on most current (circa 2011) computers. For example, a dataset representing 10^{12} objects, each with ten features, stored in short-integer (4 bytes) format would require 40 TB of storage (most high-performance computers have <1 TB of main memory). Hence, we believe that Table I will continue to be pertinent for many years.

There are two main approaches to clustering in VL data: distributed clustering which is based on various incremental styles, and clustering a sample found by either progressive or random sampling. Each has been applied in the context of FCM clustering of VL data; these ideas can also be used for *Gaussian-mixture-model* (GMM) clustering with the *expectation–maximization* (EM) algorithm. Both approaches provide useful ways to accomplish two objectives: acceleration for loadable data and approximation for unloadable data.

Consider a set of n objects $O = \{o_1, \ldots, o_n\}$, e.g., human subjects in a psychological experiment, jazz clubs in Melbourne, or wireless sensor network nodes. Each object is typically represented by numerical *feature-vector* data that have the form $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$, where the coordinates of $\mathbf{x}_i$ provide feature values (e.g., weight, length, cover charge, etc.) describing object o_i.

A *partition* of the objects is defined as a set of cn values $\{u_{ki}\}$, where each value represents the degree to which object o_i is in the kth cluster. The c-partition is often represented

Manuscript received September 6, 2011; revised January 27, 2012; accepted April 18, 2012. Date of publication May 25, 2012; date of current version November 27, 2012. This work was supported in part by Grant #1U01CA143062-01, Radiomics of Non-Small Cell Lung Cancer from the National Institutes of Health, and in part by the Michigan State University High Performance Computing Center and the Institute for Cyber Enabled Research. The work of T. C. Havens was supported by the National Science Foundation under Grant #1019343 to the Computing Research Association for the CI Fellows Project.

T. C. Havens is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824 USA (e-mail: havenst@gmail.com).

J. C. Bezdek was with the Department of Electrical and Electronic Engineering, University of Melbourne, Parkville, Vic. 3010, Australia (e-mail: jcbezdek@gmail.com).

C. Leckie is with the Department of Computer Science and Software Engineering, University of Melbourne, Parkville, Vic. 3010, Australia (e-mail: caleckie@csse.unimelb.edu.au).

L. O. Hall is with Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33630 USA (e-mail: hall@csee.usf.edu).

M. Palaniswami is with the Department of Electrical and Electronic Engineering, University of Melbourne, Parkville, Vic. 3010, Australia (e-mail: palani@unimelb.edu.au).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TFUZZ.2012.2201485

[1]Huber also defined *tiny* as 10^2 and *small* as 10^4.

as a $c \times n$ matrix $U = [u_{ki}]$. There are three main types of partitions: crisp, fuzzy (or probabilistic), and possibilistic [13], [14]. *Crisp* partitions of the unlabeled objects are non empty mutually disjoint subsets of O such that the union of the subsets equals O. The set of all nondegenerate (no zero rows) crisp c-partition matrices for the object set O is

$$M_{hcn} = \Big\{ U \in \mathbb{R}^{c\times n} | u_{ij} \in \{0,1\} \forall j, i \quad 0 < \sum_{j=1}^{n} u_{ij} < n, \forall i; \sum_{i=1}^{c} u_{ij} = 1, \forall j \Big\} \tag{1}$$

where u_{ki} is the *membership* of object o_i in cluster k; and the partition element $u_{ki} = 1$ if o_i is labeled k and is 0 otherwise. When the rows of U are considered as vectors in $\mathbb{R}^n$, we denote the kth row as $\mathbf{u}_k$; when the columns are considered as vectors in $\mathbb{R}^k$, we denote the ith column as $(U)_i$. Both $\mathbf{u_k}$ and $(U)_i$ are considered as column vectors.

Fuzzy (or probabilistic) partitions are more flexible than crisp partitions in that each object can have membership in more than one cluster. Note, if U is probabilistic, say $U = P = [p_{ki}]$, then p_{ki} is interpreted as a posterior probability $p(k|o_i)$ that o_i is in the kth class. Since this paper focuses primarily on FCM, we do not specifically address this difference. However, we stress that most, if not all, of the methods described here can be directly applied to the GMM/EM algorithm, which is the most popular way to find probabilistic clusters. The set of all fuzzy c-partitions is

$$M_{fcn} = \Big\{ U \in \mathbb{R}^{c\times n} | u_{ij} \in [0,1] \forall j, i \quad 0 < \sum_{j=1}^{n} u_{ij} < n, \forall i; \sum_{i=1}^{c} u_{ij} = 1, \forall j \Big\}. \tag{2}$$

Each column of the fuzzy partition U must sum to 1, thus ensuring that every object has unit total membership in a partition ($\sum_k u_{ki} = 1$).

Many algorithms have been proposed to cluster in VL data, but only a handful of them address the fuzzy clustering problem. Literal schemes simply cluster the entire dataset. In contrast, extended clustering schemes apply a clustering algorithm to a representative (and manageably sized) sample of the full dataset, and then noniteratively extend the sample result to obtain clusters for the remaining data in the full sample. Algorithms that include their own noniterative mechanisms for extension are referred to as extensible algorithms [15]. Perhaps the most well-known method for fuzzy clustering of VL data is the *generalized extensible fast* FCM (geFFCM) [12]. This algorithm uses statistics-based progressive sampling to produce a reduced dataset that is large enough to capture the overall nature of the data. It then clusters this reduced dataset and noniteratively extends the partition to the full dataset. However, the sampling method used in geFFCM can be inefficient and, in some cases, the data reduction is not sufficient for VL data. Hence, we will adapt geFFCM into a simple *random sampling plus extension* FCM (rseFCM) algorithm. Other leading algorithms include *single-pass* FCM (spFCM) [16] and *online* FCM (oFCM) [17], which are incremental algorithms to compute an approximate FCM solution. The *bit-reduced* FCM (brFCM) [18] algorithm uses a binning strategy for data reduction. A kernel-based strategy which is called *approximate kernel* FCM (akFCM) was developed in [19] and [20], which relies on a numerical approximation that uses sampled rows of the kernel matrix to estimate the solution to a c-means problem.

In this paper, we compare four leading algorithms to compute fuzzy partitions of VL vector data: rseFCM, spFCM, oFCM, and brFCM. Then, we will compare four *kernel* FCM (kFCM) algorithms to compute fuzzy partitions of VL data: rsekFCM, akFCM, spkFCM, and okFCM. The spkFCM and okFCM are proposed in this paper as kernelized extensions of the spFCM and oFCM algorithms.

Section II describes FCM algorithms for VL data, and Section III proposes new kernelized extensions of some of these algorithms. The complexity of the algorithms is compared in Section IV, and Section V presents empirical results. Section VI contains our conclusions and some ideas for further research. Next, we describe some of the related research.

A. *Background and Related Work*

Much research has been done on clustering in VL data. Methods can be roughly categorized into three types of algorithms. 1) *Sampling* methods compute cluster centers from a smaller sample of (often randomly) selected objects. Popular sampling-based methods include CLARA [7], CURE [21], and the coreset algorithms [22]. 2) *Single-pass* algorithms sequentially load small groups of the data, clustering these manageable chunks in a single pass, and then combining the results from each chunk. Representative algorithms include incremental clustering [23], [24] and divide-and-conquer approaches [25], [26]. 3) *Data transformation* algorithms alter the structure of the data itself so that it is more efficiently accessed. The data often take the form of graph-like structures. Well-known algorithms of this type include BIRCH [27] and CLARANS [28]. Other algorithms in this category include GARDEN$_{HD}$ [29] and CLUTO [30], which were designed to address high-dimensional data. While all the algorithms that are mentioned in this paragraph perform their job well, they all produce only crisp partitions.

Methods that generate fuzzy partitions include the *fast* FCM (FFCM) developed in [31], where FCM is applied to larger and larger nested samples until there is little change in the solution; and the *multistage random* FCM proposed in [32], which combines FFCM with a final literal run of FCM on the full dataset. Both these schemes are more in the spirit of acceleration, rather than scalability, as they both contain a final run on the full dataset. Other algorithms that are related, but were also developed for efficiency, include those described in [33] and [34]. In [35], *fast kernel* FCM (FKFCM) is proposed, which is designed to speedup the processing of quantized MRI images. This algorithm is similar to our brFCM algorithm in that it uses a weight in the FCM objective function to represent the multiple contributions of objects that have the same quantization (pixel) value. Unlike brFCM, the FKFCM algorithm is not appropriate for use with real-valued feature data.

The algorithms in this paper rely on extension to produce partitions of the full dataset. The objective of extension depends on the size of the data. When the dataset is VL, sampling

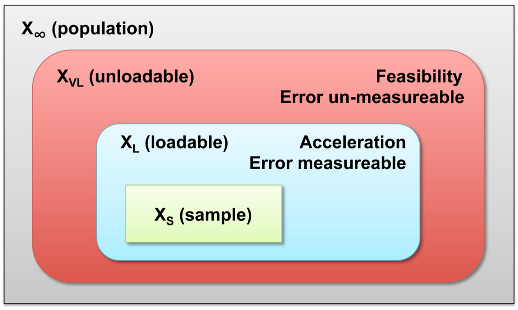


Fig. 1. Population X_∞ and samples X_{VL}, X_L, and X_s.

and extension offers a clustering solution (i.e., makes clustering feasible) for cases where it is not otherwise possible with the corresponding literal approach. If the dataset is merely *large* (L), but still loadable into memory, then an extended scheme may offer an approximate solution which is comparable to the literal solution at a significantly reduced computational cost—in other words, it accelerates the corresponding literal scheme. Therefore, the benefits for the two cases can be summarized as feasibility for VL data and acceleration for L data. Both situations are depicted in Fig. 1, where the dataset to be clustered is either X_L or X_{VL}. The set X_∞ denotes the source population, and X_s represents a sample of either X_{VL} or X_L. In a nutshell, extended schemes choose a sample X_s or collection of samples, cluster it (or them), and then extend the results to X_L or X_{VL}.

We do not specifically address how the sampled datasets are chosen; we simply use uniform random sampling. We believe that random sampling is sufficient for most VL datasets and is certainly the most efficient way to choose a sample. However, *progressive sampling* might provide improved performance for some datasets. Provost *et al.* [36] provide a very readable analysis and summary of progressive sampling schemes. Sampling methods for spectral and kernel methods are addressed in [37]–[39], and these papers also support our claim that uniform random sampling is preferred.

Another fundamental difference between X_L and X_{VL} involves the calculation of approximation error. For L datasets, we can assess the approximation error by measuring the difference between the clustering solutions obtained using the corresponding extended and literal schemes. On the other hand, the *only* solution generally available for a VL dataset is that obtained by the extended scheme, for which the approximation error cannot be measured. Thus, our confidence in the accuracy of extended clusters in the unverifiable case (X_{VL}) is necessarily derived from the verified good behavior that we can observe by conducting various X_L experiments. We do, however, perform a demonstration of the VL methods in Section V-D that compares the performance of the algorithms using a measure we devised specifically for this purpose. Now we move on to descriptions of the FCM algorithms.

II. Fuzzy *c*-Means Algorithms

The FCM algorithm is generally defined as the constrained optimization of the squared-error distortion

$$J_m(U,V) = \sum_{i=1}^{c}\sum_{j=1}^{n} u_{ij}^m \|\mathbf{x}_j - \mathbf{v}_i\|_A^2 \tag{3}$$

Algorithm 1: LFCM/AO

```
Input: $X$, $c$, $m$
Output: $U$, $V$
Initialize $V$
while $\max_{1\le k\le c}\{||v_{k,new} - v_{k,old}||^2\} > \epsilon$ do
    $u_{ij} = \left[\sum_{k=1}^{c}\left(\frac{||\mathbf{x}_j - \mathbf{v}_i||}{||\mathbf{x}_j - \mathbf{v}_k||}\right)^{\frac{2}{m-1}}\right]^{-1}, \forall i,j$   (4)
    $\mathbf{v}_i = \frac{\sum_{j=1}^{n}(u_{ij})^m \mathbf{x}_j}{\sum_{j=1}^{n}(u_{ij})^m}, \forall i$   (5)
```

where U is the $(c \times n)$ partition matrix, $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_c\}$ is the set of c cluster centers in $\mathbb{R}^d$, $m > 1$ is the fuzzification constant, and $\|\cdot\|_A$ is any inner product A-induced norm, i.e., $\|\mathbf{x}\|_A = \sqrt{\mathbf{x}^T A\mathbf{x}}$. We will only use the Euclidean norm $(A = I)$ in our examples, but there are many examples where the use of another norm-inducing matrix, e.g., using $A = S^{-1}$, the inverse of the sample covariance matrix, has been shown to be effective. The FCM/AO algorithm produces a solution to (3) using *alternating optimization* (AO) [40]. Other approaches to optimizing the FCM model include genetic algorithms, particle swarm optimization, etc. The FCM/AO approach is by far the most common, and the only algorithm used here. We may ease the notation at times by dropping the "/AO" notation unless clarity demands it. Algorithm 1 outlines the steps of the *literal* FCM/AO (LFCM/AO) algorithm. There are many ways to initialize LFCM; we choose c objects randomly from the dataset itself to serve as the initial cluster centers, which seems to work well in almost all cases. However, any initialization method that adequately covers the object space and does not produce any identical initial centers would work.

The alternating steps of LFCM in (4) and (5) are iterated until the algorithm terminates, where termination is declared when there are only negligible changes in the cluster center locations: more explicitly, $\max_{1\le k\le c}\{\|\mathbf{v}_{k,\text{new}} - \mathbf{v}_{k,\text{old}}\|^2\} < \epsilon$, where ϵ is a predetermined constant. We use $\epsilon = 10^{-3}$ in our experiments and the Euclidean norm for the termination test.

A. Sampling and Noniterative Extension

The most basic, and perhaps obvious, way to address VL data is to sample the dataset and then use FCM to compute cluster centers of the sampled data. If the data were sufficiently sampled, the error between the cluster center locations produced by clustering the entire dataset and the locations produced by clustering the sampled data should be small. Algorithm 2 outlines the *random sample* and *extend* approach, which we denote as rseFCM. Note that the extension step in line 3 of rseFCM is equivalent to (4) in FCM. Extension can be used to compute the full fuzzy data partition for any algorithm that produces (or approximates) cluster centers. In Section V, we will apply extension to other VL FCM algorithms in order to measure the relative accuracy of each clustering approach.

```
Algorithm 2: rseFCM to approximately minimize $J_m(U,V)$
Input: $X$, $c$, $m$
Output: $U$,$V$
1 Sample $n_s$ objects from $X$ without replacement, denoted $X_s$
2 $U_s, V$ = LFCM$(X_s, c, m)$
3 Extend the partition $(U_s, V)$ to $X$, $\forall x_i \notin X_s$, using Eq. (4), giving $(U,V)$
```

```
Algorithm 3: wFCM/AO to minimize $J_{m\mathbf{w}}(U,V)$ [13]
Input: $X$, $c$, $m$, $\mathbf{w}$, (initial $V$)
Output: $U$, $V$
If $V$ is not initialized, initialize $V$
while $\max_{1\leq i\leq c}\{||\mathbf{v}_{i,new} - \mathbf{v}_{i,old}||^2\} > \epsilon$ do
    Calculate $U$ with Eq. (4)
    $\mathbf{v}_i = \frac{\sum_{j=1}^n w_j (u_{ij})^m \mathbf{x}_j}{\sum_{j=1}^n w_j (u_{ij})^m}, \forall i$    (7)
```

Remark: Once the extension step is completed, so that a partition U on the full dataset is known, we can perform a *completion step* by using U to (re)compute the cluster centers V with (5). Extension, followed by completion, yields a pair (U,V) that satisfies the first-order necessary conditions to minimize J_m, and if desired, we can compute the value $J_m(U,V)$.

B. Algorithms Based on Weighted Fuzzy c-Means

In LFCM, each object is considered equally important in the clustering solution. The *weighted* FCM (wFCM) model introduces weights that define the relative importance of each object in the clustering solution. Hence, wFCM is defined as the constrained optimization of

$$J_{m\mathbf{w}}(U,V) = \sum_{i=1}^{c}\sum_{j=1}^{n} w_j u_{ij}^m \|\mathbf{x}_j - \mathbf{v}_i\|_A^2 \tag{6}$$

where $\mathbf{w} \in \mathbb{R}^n$, $w_j \geq 0$, is a set of predetermined weights that define the influence of each feature vector. Algorithm 3 outlines the wFCM algorithm. As (7) shows, objects with a higher weight w are more influential in defining the location of the cluster centers V. These weights are important in the spFCM, oFCM, and brFCM algorithms, which are outlined next.

1) Single-Pass Fuzzy c-Means: Algorithm 4 outlines the spFCM algorithm. Line 1 of the algorithm sets the weight vector $\mathbf{w}$ to the n_s-length vector of 1s. Line 2 calculates the wFCM partition of the first sample set of data, returning the c cluster centers V (note that wFCM in Line 2 is initialized using your chosen initialization method). spFCM then iterates over the remaining subsets of data in X. At each iteration, wFCM is used to cluster an augmented set of data that is composed of the union of the cluster centers from the previous iteration and the sample subset X_l, which we denote as $\{V \cup X_l\}$. Hence, there are

```
Algorithm 4: spFCM/AO to approximately minimize $J_{m\mathbf{w}}(U,V)$ [16]
Input: $X$, $c$, $m$, $n_s$
Output: $V$
Load $X$ as $n_s$-sized randomly chosen subsets, $X = \{X_1, X_2, \ldots, X_s\}$
1 $\mathbf{w} = \mathbf{1}_{n_s}$
2 $U, V$ = wFCM$(X_1, c, m, \mathbf{w})$
for $l = 2$ to $s$ do
3   $w'_i = \sum_{j=1}^{n_s} (u_{ij}) w_j,\ i = 1, \ldots, c$    (8)
4   $\mathbf{w} = \{\mathbf{w}' \cup \mathbf{1}_{n_s}\}$
5   $U, V$ = wFCM$(\{V \cup X_l\}, c, m, \mathbf{w}, V)$
```

```
Algorithm 5: oFCM/AO to approximately minimize $J_{m\mathbf{w}}(U,V)$ [17]
Input: $X$, $c$, $m$, $n_s$
Output: $V$
Load $X$ as $n_s$-sized subsets, $X = \{X_1, X_2, \ldots, X_s\}$, where $X_i = \{\mathbf{x}_{(i-1)n_s+1}, \ldots, \mathbf{x}_{in_s}\}$
1 $U_1, V_1$ = wFCM$(X_1, c, m, \mathbf{1}_{n_s})$
for $l = 2$ to $s$ do
2   $U_l, V_l$ = wFCM$(X_l, c, m, \mathbf{1}_{n_s}, V_{l-1})$
3 $\mathbf{w}_l = \sum_{j=1}^{n_s} (U_l)_j,\ l = 1, \ldots, s$
4 $V$ = wFCM$(\{V_1 \cup \ldots \cup V_s\}, c, m, \{\mathbf{w}_1 \cup \ldots \cup \mathbf{w}_s\})$
```

$(c + n_s)$ objects that are clustered at each iteration. Lines 3 and 4 determine the weights that are used in wFCM for the augmented input dataset $\{V \cup X_l\}$. Line 3 calculates the weights for the c (next step) cluster centers V from the current cluster memberships, and Line 4 creates the $(c + n_s)$-length weight vector for the next step, composed of the c weights that correspond to the cluster centers and n_s 1s, which correspond to the objects in the sample subset X_l. Line 5 shows that wFCM takes as input the augmented subset of data $\{V \cup X_l\}$, the number of clusters c, and the weight vector $\mathbf{w}$. In addition, wFCM is initialized using the cluster centers V from the previous iteration (shown as the fifth argument in Line 5), which speeds up convergence.

2) Online Fuzzy c-Means: Algorithm 5 outlines the oFCM algorithm. Unlike spFCM, which computes the new cluster centers by feeding forward the cluster centers from the previous iteration into the data being clustered, oFCM clusters all s subsets of objects separately and then aggregates the s sets of cluster centers at the end. In Line 2 of oFCM, a partition and set of cluster centers, denoted as U_1 and V_1, are produced for the first subset of objects X_1. At each iteration of Line 2, wFCM partitions X_l, producing U_l and V_l. Note that we use the cluster centers from the previous iteration to initialize wFCM to speed up convergence (this feedforward initialization could be ignored for the case where each iteration of Line 2 is performed separately, such as on a distributed architecture). Line 3 computes the weights for each of the c cluster centers in each of the s sets of Vs. Note

Algorithm 6: brFCM/AO to approximately minimize $J_{m\mathbf{w}}(U,V)$ [18]

```
Input: $X$, $c$, $m$
Output: $V$
Bin $X$ into $s$ quantization bins, where
$X' = \{\mathbf{x}'_k\},\ k = 1, \ldots, s$, is the set of bin prototypes
(typically the means of the binned feature vectors or the
bit-reduced binary values)
Set $w_k$ to the number of feature vectors aggregated into
$\mathbf{x}'_k$, $\forall k$
$V = $ wFCM$(X', c, m, \mathbf{w})$
```

that $(U_l)_j$ indicates the jth column of U_l. Finally, wFCM is used in Line 4 to cluster the (cs) centers $\{V_1 \cup \ldots \cup V_s\}$ using the corresponding weights $\{\mathbf{w}_1 \cup \ldots \cup \mathbf{w}_s\}$. The final output is c cluster centers V.

The size of the dataset that is composed of the cluster centers (which are processed at Line 4) could be large in cases where X is extremely large and must be broken up into many chunks (s is large). If the cluster center data at Line 4 are too large to be processed, it could be processed by another run of oFCM or a streaming algorithm, such as proposed in [41].

3) Bit-Reduced Fuzzy c-Means: brFCM was designed to address the problem of clustering in large images. Algorithm 6 outlines the steps of this algorithm. The brFCM algorithm begins by binning the input data X into a reduced set X', where X' is the set of bin prototypes (i.e., the bin centers). This reduced set X' is then clustered using wFCM, where the weights are the number of objects in each bin. There are many ways to bin data; e.g., image data can be bit-reduced by removing the least significant bits and binning the identical objects. An easy way to bin the data is to compute the s-bin histogram; the weights are the number of pixels in each bin, and the data are the bin centers (the means of the data in each bin). This is the method that we will use in this paper as it provides us with an easy way to directly compare the results with the other VL FCM algorithms as we can specify the number of bins (which is equivalent to specifying the sample size).

Note that the spFCM, oFCM, and brFCM algorithms all produce cluster centers V by approximately minimizing J_{mw} and not the full $c \times n$ data partition. However, this partition can be easily computed by the extension step (Line 3) of Algorithm 2 and then the completion step, computing a final V with the full U, yields a necessary pair (U, V) that can be evaluated by J_m if desired.

It was shown in [42] that the spFCM and oFCM algorithms converge—[42, Ths. 1 and 2] also can be used to show that rseFCM converges. Now we move to kernel versions of the algorithms that are presented.

C. Kernel Fuzzy c-Means Algorithms

Consider some nonlinear mapping function ϕ: $\mathbf{x} \rightarrow \phi(\mathbf{x}) \in \mathbb{R}^{D_K}$, where D_K is the dimensionality of the transformed feature vector $\mathbf{x}$. With kernel clustering, we do not need to explicitly transform $\mathbf{x}$; we simply need to represent the dot product $\phi(\mathbf{x}) \cdot \phi(\mathbf{x}) = \kappa(\mathbf{x}, \mathbf{x})$. The kernel function κ can take many forms, with the polynomial $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^p$ and *radial basis function* (RBF) $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(\sigma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$ being two of the most well known. Given a set of n object vectors X, we can, thus, construct an $n \times n$ kernel matrix $K = [K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)]_{n \times n}$. This kernel matrix K represents all pairwise dot products of the feature vectors in the transformed high-dimensional space—called the *reproducing kernel Hilbert space* (RKHS).

Given a kernel function κ, *kernel* FCM (kFCM) can be generally defined as the constrained minimization of a reformulation—by elimination of V by substitution of (5) into (3)—of the objective in (3):

$$J_m(U;\kappa) = \sum_{j=1}^{c} \left(\sum_{i=1}^{n} \sum_{k=1}^{n} \left(u_{ij}^m u_{kj}^m d_\kappa(\mathbf{x}_i, \mathbf{x}_k) \right) / 2 \sum_{l=1}^{n} u_{lj}^m \right) \tag{9}$$

where $U \in M_{fcn}$, $m > 1$ is the fuzzification parameter, and $d_\kappa(\mathbf{x}_i, \mathbf{x}_k) = \kappa(\mathbf{x}_i, \mathbf{x}_i) + \kappa(\mathbf{x}_k, \mathbf{x}_k) - 2\kappa(\mathbf{x}_i, \mathbf{x}_k)$ is the kernel-based distance between the ith and kth feature vectors. Note that in (9) we use a partition matrix that is $(n \times c)$ to stay consistent with the existing kernel clustering literature. We will stick to this convention for our discussion of kernel algorithms.

kFCM solves the optimization problem $\min\{J_m(U;\kappa)\}$ by computing iterated updates of

$$u_{ij} = \left[\sum_{k=1}^{c} \left(\frac{d_\kappa(\mathbf{x}_i, \mathbf{v}_j)}{d_\kappa(\mathbf{x}_i, \mathbf{v}_k)} \right)^{\frac{1}{m-1}} \right]^{-1} \quad \forall i, j. \tag{10}$$

The kernel distance between input datum $\mathbf{x}_i$ and cluster center $\mathbf{v}_j$ is

$$d_\kappa(\mathbf{x}_i, \mathbf{v}_j) = \|\phi(\mathbf{x}_i) - \phi(\mathbf{v}_j)\|^2 \tag{11}$$

where, like LFCM, the cluster centers are linear combinations of the feature vectors

$$\phi(\mathbf{v}_j) = \frac{\sum_{l=1}^{n} u_{lj}^m \phi(\mathbf{x}_l)}{\sum_{l=1}^{n} u_{lj}^m}. \tag{12}$$

Equation (11) cannot by computed directly, but by using the identity $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$, denoting $\tilde{\mathbf{u}}_j = \mathbf{u}_j^m / \sum_i |u_{ij}^m|$ where $\mathbf{u}_j^m = (u_{1j}^m, u_{2j}^m, \ldots, u_{nj}^m)^T$, and substituting (12) into (11), we get

$$\begin{aligned} d_\kappa(\mathbf{x}_i, \mathbf{v}_j) &= \frac{\sum_{l=1}^{n} \sum_{s=1}^{n} u_{lj}^m u_{sj}^m \phi(\mathbf{x}_l) \cdot \phi(\mathbf{x}_s)}{\sum_{l=1}^{n} u_{lj}^{2m}} \\ &\quad + \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_i) - 2 \frac{\sum_{l=1}^{n} u_{lj}^m \phi(\mathbf{x}_l) \cdot \phi(\mathbf{x}_i)}{\sum_{l=1}^{n} u_{lj}^m} \\ &= \tilde{\mathbf{u}}_j^T K \tilde{\mathbf{u}}_j + \mathbf{e}_i^T K \mathbf{e}_i - 2\tilde{\mathbf{u}}_j^T K \mathbf{e}_i \\ &= \tilde{\mathbf{u}}_j^T K \tilde{\mathbf{u}}_j + K_{ii} - 2\left(\tilde{\mathbf{u}}_j^T K\right)_i \end{aligned} \tag{13}$$

where $\mathbf{e}_i$ is the n-length unit vector with the ith element equal to 1. This formulation of kFCM is equivalent to that proposed in [43] and, furthermore, is identical to *relational* FCM [44] if the kernel $\kappa(\mathbf{x}_i, \mathbf{x}_k) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ is used [45].

Equation (13) shows the obvious problem which arises when using kernel clustering with VL data; the distance equation's complexity is quadratic with the number of objects. Furthermore, the memory requirement to store K is also quadratic with the number of objects.

1) Approximate Kernel Fuzzy c-Means: The akFCM [20] algorithm approximates a solution to kFCM by using a numerical approximation of the cluster center to object kernel distance in (13) that is based on the assumption that the cluster centers can be accurately represented as a linear sum of a subset of the feature vectors. First, s feature vectors are drawn randomly. The cluster centers are approximated by a linear sum of the selected objects

$$\hat{\phi}(\mathbf{v}_j) = \sum_{i=1}^{n} a_{ij}\xi_i\phi(\mathbf{x}_i) = \sum_{i=1}^{s}\alpha_{ij}\phi(\mathbf{x}_{(i)}) \tag{14}$$

where a_{ij} is the weight of the ith feature vector in the jth cluster center, ξ_i is a binary-selection variable ($\xi_i = 1$ if $\mathbf{x}_i$ is drawn and $\xi_i = 0$ if $\mathbf{x}_i$ is not), and $\alpha_{ij} = a_{(i)j}$ are the weights of the chosen vectors $\phi(\mathbf{x}_{(i)})$, $i = 1, \ldots, s$. Although this notation seems cumbersome, it dramatically simplifies the description of akFCM, as we can ignore the weights a_{ij} for the objects that are not sampled ($\xi_i = 0$).

After sampling, two kernel sub-matrices are computed:

$$K_{\xi\xi} = [K_{ij}]_{s\times s}\ \forall i,j|\xi_i = 1, \xi_j = 1 \tag{15}$$

$$K_{\xi} = [(K)_i]_{n\times s}\ \forall i|\xi_i = 1 \tag{16}$$

where $(K)_i$ is the ith column of K. Notice that $K_{\xi\xi}$ is the $s \times s$ square matrix that corresponds to the pairwise elements of K for the ξ-selected objects. K_ξ is the $n \times s$ rectangular matrix that corresponds to the columns of K for the ξ-selected objects; also, notice that $K_{\xi\xi}$ is a submatrix of K_ξ.

Using these two submatrices, the weights $\boldsymbol{\alpha}_j$ (where this is the s-length vector, $\boldsymbol{\alpha}_j = (\alpha_{1j}, \ldots, \alpha_{sj})^T$) are computed as

$$\boldsymbol{\alpha}_j = -(K_{\xi\xi}^{-1}K_\xi^T\tilde{\mathbf{u}}_j)^T. \tag{17}$$

See [19] and [20] for a detailed derivation. In practice, $K_{\xi\xi}$ could be rank deficient (have a number of relatively small eigenvalues); hence, it is advisable to use a pseudoinverse when computing $K_{\xi\xi}^{-1}$, which we denote as $K_{\xi\xi}^{\dagger}$. In addition, the calculation of $K_{\xi\xi}^{-1}K_\xi^T$ only needs to be performed once, at the beginning of the algorithm, with the result stored.

The key element of akFCM is the approximation of (13) using $\boldsymbol{\alpha}_j$, i.e.,

$$\hat{d}_\kappa(\mathbf{x}_i, \mathbf{v}_j) = \boldsymbol{\alpha}_j^T K_{\xi\xi}\boldsymbol{\alpha}_j + K_{ii} - 2(K_\xi\boldsymbol{\alpha}_j)_i. \tag{18}$$

Notice that the full kernel matrix is no longer required to compute the distance $\hat{d}_\kappa(\mathbf{x}_i, \mathbf{v}_j)$; the only required kernel matrix elements are the diagonal elements (which for the RBF kernel are all equal to 1) and the s columns of K corresponding to the selected objects. Algorithm 7 outlines the steps of akFCM.

One theoretical advantage of the akFCM algorithm is that it has a bounded squared-error distortion, $\mathcal{L}_m(U, \Xi)$, of[2]

$$\frac{\mathrm{E}_\Xi[\mathcal{L}_m(U,\Xi)]}{\mathcal{L}_m(U,\mathbf{1})} \le \mathrm{tr}\left((U^m)^T\left[K^{-1} + \frac{s}{n}[\mathrm{diag}(K)]^{-1}\right]^{-1}\tilde{U}^m\right)$$

[2]See [19] for a detailed derivation of this error for hard kernel c-means. The error we present here for akFCM is a simple extension of that derivation.

Algorithm 7: akFCM/AO to approximately minimize $J_m(U;\kappa)$ [20]

```
Input: $K_{\xi\xi}$, $K_\xi$, diag($K$), $c$, $m$
Output: $U$
Initialize $U \in M_{fcn}$
$\hat{K} = K_{\xi\xi}^{\dagger}K_\xi^T$
while $\max_{1\le k\le c}\{||\mathbf{u}_{k,new} - \mathbf{u}_{k,old}||^2\} > \epsilon$ do
    $\boldsymbol{\alpha}_j = \left(\hat{K}\tilde{\mathbf{u}}_j\right)^T$
    $\hat{d}_\kappa(\mathbf{x}_i,\mathbf{v}_j) = \boldsymbol{\alpha}_j^T K_{\xi\xi}\boldsymbol{\alpha}_j + K_{ii} - 2\left(K_\xi\boldsymbol{\alpha}_j\right)_i,\ \forall i,j$
    $u_{ij} = \left[\sum_{k=1}^{c}\left(\frac{\hat{d}_\kappa(\mathbf{x}_i,\mathbf{v}_j)}{\hat{d}_\kappa(\mathbf{x}_i,\mathbf{v}_k)}\right)^{\frac{1}{m-1}}\right]^{-1},\ \forall i,j$
```

where E_Ξ is the expectation with respect to the selection variable $\Xi = (\xi_1, \ldots, \xi_n)$, $[\tilde{U}^m]_{ik} = u_{ik}^m / \sum_j |u_{jk}^m|$ is the (column)-normalized membership matrix, and

$$\mathcal{L}_m(U,\mathbf{1}) = \sum_{i=1}^{n} K_{ii}\sum_{j=1}^{c} u_{ij}^m - \mathrm{tr}((U^m)^T K\tilde{U}^m)$$

is the squared-error distortion of the literal kFCM solution (i.e., $\Xi = \mathbf{1}_n$, the vector of all 1s). In [19], it was shown that this error, for $m = 1$, is bounded by cn/s for kernels where $K_{ij} \le 1$, which is an intuitively pleasing result. This bound also holds for $m > 1$, as $u_{ij} \le 1$. Next, we describe novel kernelized extensions of the wFCM, spFCM, and oFCM algorithms.

III. New Kernel Fuzzy c-Means Algorithms for Very Large Data

A. Weighted Kernel Fuzzy c-Means

The extension of the kFCM model $J_m(U;\kappa)$ to the *weighted* kFCM (wkFCM) model $J_{m\mathbf{w}}(U;\kappa)$ follows the same pattern as the extension of (3) to (6). The cluster center $\phi(\mathbf{v}_j)$ is a weighted sum of the feature vectors, as shown in (12). Now assume that each object $\phi(\mathbf{x}_i)$ has a different predetermined influence, given by a respective weight w_i. This leads to

$$\phi(\mathbf{v}_j) = \frac{\sum_{l=1}^{n} w_l u_{lj}^m\phi(\mathbf{x}_l)}{\sum_{l=1}^{n} w_l u_{lj}^m}. \tag{19}$$

Substituting (19) into (13) gives

$$\begin{aligned} d_\kappa^{\mathbf{w}}(\mathbf{x}_i,\mathbf{v}_j) = &\frac{1}{\|\mathbf{w}\circ\mathbf{u}_j\|^2}(\mathbf{w}\circ\mathbf{u}_j)^T K(\mathbf{w}\circ\mathbf{u}_j) + K_{ii} \\ &- \frac{2}{\|\mathbf{w}\circ\mathbf{u}_j\|}\left((\mathbf{w}\circ\mathbf{u}_j)^T K\right)_i \end{aligned} \tag{20}$$

where $\mathbf{w}$ is the vector of predetermined weights, and $\circ$ indicates the Hadamard product. This leads to the wkFCM algorithm shown in Algorithm 8. Notice that wkFCM also outputs the index of the object closest to each cluster center, which is called the cluster prototype. The vector of indices $\mathbf{p}$ is important in the VL data schemes that are now proposed.

Algorithm 8: wkFCM/AO to minimize $J_{m\mathbf{w}}(U;\kappa)$

```
Input: $K$, $c$, $m$, $\mathbf{w}$
Output: $U$, $\mathbf{p} = \{p_1, \ldots, p_c\}$
Initialize $U \in M_{fcn_s}$
while $\max_{1\leq k\leq c}\{||\mathbf{u}_{k,new} - \mathbf{u}_{k,old}||^2\} > \epsilon$ do
    Compute $d_\kappa^{\mathbf{w}}(\mathbf{x}_i, \mathbf{v}_j)$ using (20)
    $u_{ij} = \left[\sum_{k=1}^{c}\left(\frac{d_\kappa^{\mathbf{w}}(\mathbf{x}_i,\mathbf{v}_j)}{d_\kappa^{\mathbf{w}}(\mathbf{x}_i,\mathbf{v}_k)}\right)^{\frac{1}{m-1}}\right]^{-1}, \forall i, j$
$p_j = \arg\min_i\{d_\kappa(\mathbf{x}_i, \mathbf{v}_j)\}, \ \forall j$
```

Algorithm 9: rsekFCM/AO to approximately minimize $J_{m\mathbf{w}}(U;\kappa)$

```
Input: Kernel function $\kappa$, $X$, $c$, $m$, $n_s$
Output: $U$
1 Sample $n_s$ vectors from $X$, denoted $X_s$
2 $K = [\kappa(\mathbf{x}_i, \mathbf{x}_j)], \ \forall \mathbf{x}_i, \mathbf{x}_j \in X_s$
3 $U, \mathbf{p} =$ wkFCM$(K, c, m, \mathbf{1}_{n_s})$
4 Extend partition to $X$:
  $d_\kappa(\mathbf{x}_i, \mathbf{v}_j) = \kappa(\mathbf{x}_i, \mathbf{x}_i) + \kappa(\mathbf{x}_{p_j}, \mathbf{x}_{p_j}) - 2\kappa(\mathbf{x}_i, \mathbf{x}_{p_j}), \ \forall i, j$
  Update $U$ using (10).
```

B. Random Sample and Extend Kernel Fuzzy c-Means

The *random sample and extend* kernel FCM (rsekFCM) follows the same idea as rseFCM in Algorithm 2. A sample X_s of X is chosen, and this sample is clustered using wkFCM. The cluster prototypes that are returned by wkFCM are then used to extend the partition to the entire dataset. Algorithm 9 outlines the rsekFCM procedure. Like the feature vector case, the extension steps, starting at Line 4, can be used to extend the partition for any algorithm that returns cluster prototypes (rather than a full partition).

C. Single-Pass Kernel Fuzzy c-Means

The *single-pass* kFCM (spkFCM) which is outlined in Algorithm 10 performs the same basic steps for kernel data as spFCM does for feature vectors. At Line 1, s (n/s)-sized sets of indices are drawn, without replacement, from the set $\{1, \ldots, n\}$. We call these sets of indices $E = \{\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_s\}$. The indices in $\boldsymbol{\xi}_l$ are the object indices that are clustered in the lth step of the algorithm. At Line 3, the kernel matrix for the first subset of objects is calculated, and at Line 4 these objects are clustered with unity-valued weights. At Line 5, the weights for the c cluster prototypes are computed. Lines 6–10 comprise the main loop of spkFCM. At Line 6, an $(n/s + c)$ weight vector is created, which includes the c weights of the cluster prototypes returned by the previous iteration and n/s 1s. At Lines 7 and 8, the $(n/s + c) \times (n/s + c)$ kernel matrix is calculated, and at Line 9 the objects are clustered. Finally, at Line 9 the weights of the c new prototypes are computed. After each subset of X is operated on, Line 11 returns the indices of the c objects that are the cluster protoypes.

Algorithm 10: spkFCM to approximately minimize $J_{m\mathbf{w}}(U;\kappa)$

```
Input: Kernel function $\kappa$, $X$, $c$, $m$, $s$
Output: $\mathbf{p}$
1 Randomly, without replacement, draw $s$ (approximately)
  equal-sized subsets of the integers $\{1, \ldots, n\}$, denoted
  $E = \{\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_s\}$. $n_l$ is the cardinality of $\boldsymbol{\xi}_l$.
2 $\boldsymbol{\xi}' = \boldsymbol{\xi}_1$
3 $K = [\kappa(\mathbf{x}_i, \mathbf{x}_j)], \ i, j = \boldsymbol{\xi}'$
4 $U, \mathbf{p} =$ wkFCM$(K, c, m, \mathbf{1}_{n_1})$
5 $w'_j = \sum_{i=1}^{n_1} u_{ij}, \ \forall j$
  for $l = 2$ to $s$ do
6     $\mathbf{w} = \{\mathbf{w}', \mathbf{1}_{n_l}\}$
7     $\boldsymbol{\xi}' = \{\boldsymbol{\xi}'(\mathbf{p}), \boldsymbol{\xi}_l\}$
8     $K = [\kappa(\mathbf{x}_i, \mathbf{x}_j)], \ i, j = \boldsymbol{\xi}'$
9     $U, \mathbf{p} =$ wkFCM$(K, c, m, \mathbf{w})$
10    $w'_j = \sum_{i=1}^{n_l+c} u_{ij}, \ \forall j$
11 $\mathbf{p} = \boldsymbol{\xi}'(\mathbf{p})$
```

Algorithm 11: okFCM to approximately minimize $J_{m/w}(U;\kappa)$

```
Input: Kernel function $\kappa$, $X$, $c$, $m$, $s$
Output: $U$, $\mathbf{p}$
1 Randomly draw $s$ (approximately) equal-sized subsets of
  the integers $\{1, \ldots, n\}$, denoted $E = \{\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_s\}$. $n_l$ is
  the cardinality of $\boldsymbol{\xi}_l$.
2 $K = [\kappa(\mathbf{x}_i, \mathbf{x}_j)], \ i, j = \boldsymbol{\xi}_1$
3 $U_1, \mathbf{p}_1 =$ wkFCM$(K, c, m, \mathbf{1}_{n_1})$
  for $l = 2$ to $s$ do
4     $K = [\kappa(\mathbf{x}_i, \mathbf{x}_j)], \ i, j = \mathbf{x}'_l$
5     $U_l, \mathbf{p}' =$ wkFCM$(K, c, m, \mathbf{1}_{n_l})$
6     $\mathbf{p}_l = \boldsymbol{\xi}_l(\mathbf{p}')$
7 $\mathbf{p}_{all} = \{\mathbf{p}_1, \ldots, \mathbf{p}_s\}$
8 $K = [\kappa(\mathbf{x}_i, \mathbf{x}_j)], \ i, j = \mathbf{p}_{all}$
9 $\mathbf{w}_l = \sum_{j=1}^{n_s} (U_l)_j, \ \forall l$
10 $U, \mathbf{p}' =$ wkFCM$(K, c, m, \mathbf{w})$
11 $\mathbf{p} = \mathbf{p}_{all}(\mathbf{p}')$
```

D. Online Kernel FCM

Online kFCM (okFCM), which is outlined in Algorithm 11, is built on the ideas used in the oFCM algorithm. First, in Line 1, the dataset X is split into s roughly equal-sized datasets by randomly drawing a set of selection vectors E. Each subset of X is then individually clustered using the wkFCM algorithm, where each object is weighted equally (see Lines 2–6). The objects that are indexed by the cluster prototypes $\mathbf{p}_l$—returned by each run of wkFCM—are then clustered together in one final step, where the weights for each prototype are the sum of the respective column of the partition matrix U_l (see Lines 7–10). The final output is a set of c cluster prototypes $\mathbf{p}$, represented by the c indices of the corresponding objects in X.

TABLE II
TIME AND SPACE COMPLEXITY OF FCM/AO VL ALGORITHMS

(a) Vector Data Algorithms

Algorithm	**Time**	**Space**
wFCM, LFCM	$O(tc^2dn)$	$O((d+c)n)$
rseFCM	$O(tc^2dn/s)$	$O((d+c)(n/s))$
spFCM	$O(tc^2dn)$	$O((d+c)(n/s))$
oFCM	$O(tc^2dn)$	$O((d+c)(n/s)+cs)$
brFCM	$O(tc^2sd)$ + bin	$O((d+c)s)$
Extension	$O(c^2dn)$	$O(cn)$

(b) Kernel Algorithms

Algorithm	**Time**	**Space**
wkFCM, kFCM	$O(tcn^2)$	$O(n^2)$
akFCM	$O(n^3/s^2+tcn^2/s)$	$O(n^2/s)$
rsekFCM	$O(tcn^2/s^2)$	$O(n^2/s^2)$
spkFCM	$O(tcn^2/s)$	$O(n^2/s^2)$
okFCM	$O(tcn^2/s+tc^3s^2)$	$O(n^2/s^2+s^2)$
Extension	$O(cn)$	$O(cn)$

It is clear that all the VL implementations of FCM reduce the amount of data that are simultaneously required. The next section analyzes in detail the time and space requirements for each of these algorithms.

IV. COMPLEXITY

We estimate the time and space complexity of each of the proposed VL variants of FCM/AO. All operations and storage space are counted as unit costs. We do not assume economies that might be realized by special programming tricks or properties of the equations involved. For example, we do not make use of the fact that the kernel matrices are symmetric matrices to reduce various counts from n^2 to $n(n-1)/2$, and we do not assume space economies that might be realized by overwriting of arrays, etc. Therefore, our "exact" estimates of time and space complexity are exact only with the assumptions we have used to make them. Importantly, however, the asymptotic estimates that are shown in Table II for the growth in time and space with n, which is the number of objects in X, are unaffected by changes in counting procedures.

It is easy to let asymptotic estimates lull you into believing that methods are "equivalent" (and they are, in the limit). However, we never reach infinity in real computations, so empirical comparisons of speedup are useful and are presented later. Table II(a) shows the complexities of the vector data algorithms in terms of problem variables: n is the number of objects in the d-dimensional data, $X \in \mathbb{R}^d$; c is the number of clusters; t is the number of iterations required for termination; and s is the number of subsets that X is divided into by random sampling without replacement or the number of bins for brFCM. Table II(b) provides complexities of the kernel algorithms.

As Table II(a) shows, the wFCM, LFCM, spFCM, and oFCM all share the same big-O time complexity, as all these algorithms operate on every object in X. However, as we will see in Section V, the run-times of these algorithms differ significantly. Essentially, each subpart of spFCM and oFCM converges in fewer iterations, which results in reduced overall run-time. rseFCM and brFCM have reduced big-O time complexity, compared with the other algorithms, because they cluster a reduced set of data. The extension algorithm uses the cluster centers that are returned by any of these VL algorithms to produce full data partitions. We have not estimated complexities for the completion step, which is not used in our experiments.

The space complexity of the VL vector data algorithms is less when compared with LFCM and wFCM, and in each case it is proportional to the number of objects that are in each chunk, i.e., n/s. oFCM has a slightly greater space complexity as it must store the c prototypes for all s chunks (one could easily imagine a scheme where the c prototypes are processed incrementally by oFCM or another streaming algorithm, which would reduce the space complexity of oFCM).

The time and space complexities of the kernel-based algorithms in Table II(b) show similar trends to their corresponding vector data counterparts. The main drawback of kernel clustering is the $O(n^2)$ memory requirement for the storage of the kernel matrix. The rsekFCM algorithm combats this by only computing the $(n/s) \times (n/s)$ kernel matrix for the sampled data, resulting in an $O(n^2/s^2)$ space complexity.[3] The akFCM algorithm requires $O(n^2/s)$ units of memory to store the rectangular kernel matrix K_ξ. The spFCM and okFCM algorithms operate on $O(n^2/s^2)$-sized kernel matrices, and the final step of okFCM requires $O(s^2)$ units of storage (resulting in $O(n^2/s^2+s^2)$ space complexity for okFCM).[4] Like the extension step for vector data, the extension step for kFCM requires $O(cn)$ to store the partition matrix.

The time complexity of the kernel algorithms is greater than that of their vector data counterparts and is dominated by the the computation of (13). This calculation requires $O(n^2)$ operations per cluster, resulting in a total time complexity of $O(tcn^2)$ for kFCM and wkFCM. The akFCM algorithm has a time complexity of $O(n^3/s^2+tcn^2/s)$, where $O(n^3/s^2)$ is required for the one-time calculation of $K^\dagger_{\xi\xi}K^T_\xi$ [20]. The rsekFCM algorithm is equivalent to kFCM or wkFCM for an (n/s)-sized dataset, resulting in a time complexity of $O(tcn^2/s^2)$. The proposed spkFCM and okFCM algorithms run wkFCM on s chunks of approximately (n/s)-sized data, which results in $O(tcn^2/s)$ time complexity. Note that if $O(s) = O(n/s)$, then okFCM has a time complexity of $O(tcn^2/s+tc^3s^2)$ because the last step of okFCM clusters—using wkFCM—the (cs) prototypes that are produced from the s data chunks.

Overall, the main strength of the VL FCM algorithms is the reduced space complexity, compared with the literal implementations. However, the time complexity for many of these algorithms is also reduced. In the next section, we will see that even in the cases where the literal and VL implementations share the same asymptotic time complexity, the VL implementation often produces a faster run-time.

[3]Note that because the size of the memory required to store the kernel matrix dominates that of the partition matrix, we disregard the space complexity of storing the partition matrix, as is appropriate in big-O calculations.

[4]The space complexity of okFCM could be dominated by either $O(n^2/s^2)$ or $O(s^2)$, depending on the relationship of n and s. In typical VL applications, $n \gg s$; hence, the space complexity of okFCM will go to $O(n^2/s^2)$.

V. Experiments

We performed two sets of experiments. The first experiments compare the performance of the VL FCM algorithms on data for which there exists ground truth (or known object labels). The second set of experiments applies the proposed algorithms to datasets for which there exists no ground truth. For these data, we compared the partitions from the VL FCM algorithms to the LFCM partitions.

For all vector data algorithms, we initialize V by randomly choosing c objects as the initial cluster centers. For the kernel algorithms, we initialize U by choosing c objects as the initial cluster centers and setting the corresponding entry in U equal to 1 for each of those objects, all other entries are set to 0. We ensure that each algorithm is started with the same initialization (and same data sampling for the single-pass and online variants) at the start of a run, with a different initialization drawn for each run. We fix the values of $\epsilon = 10^{-3}$ and the fuzzifier $m = 1.7$.[5] The termination criterion for the vector data algorithms is $\max_{1 \le k \le c}\{\|\mathbf{v}_{k,new} - \mathbf{v}_{k,old}\|^2\} < \epsilon$, and for the kernel algorithms it is $\max_{1 \le k \le c}\{\|\mathbf{u}_{k,new} - \mathbf{u}_{k,old}\|^2\} < \epsilon$. The experiments were performed on a dedicated single core of an AMD Opteron in a Sun Fire X4600 M2 server with 256 GB of memory. All code was written in the MATLAB computing environment.

A. Evaluation Criteria

We judge the performance of the VL FCM algorithms using two criteria. They are computed for 50 independent runs with random initializations and samplings. We present statistical comparisons of the algorithms' performance over the 50 experiments.

1) Speedup Factor or Run-Time: This criterion represents an actual run-time comparison. When the LFCM or kFCM solution is available, speedup is defined as $t_{\text{full}}/t_{\text{samp}}$, where these values are times in seconds to compute the cluster centers V for the vector data algorithms and the membership matrix U for the kernel algorithms. In the cases where LFCM and kFCM solutions cannot be computed, we present run-time in seconds for the various VL algorithms. Although in our experiments we loaded the data into memory all at once at the beginning, we expect the speedup factors for truly unloadable data to be similar because the same amount of data are read into memory whether it is done in chunks or all at once. If data need to be broken into a large number of chunks, say >100, then speedup factor could be slightly degraded (although, at that point, the feasibility to produce a solution with an *extremely* large dataset would be more important than speedup).

2) Adjusted Rand Index: The Rand index [49] is a measure of agreement between two crisp partitions of a set of objects. One of the two partitions is usually a crisp reference partition U', which represents the ground truth labels for the objects in the data. In this case, the value $R(U, U')$ measures the degree to which a candidate partition U matches U'. A Rand index of 1 indicates perfect agreement, while a Rand index of 0 indicates perfect disagreement. The version that we use here, the *adjusted Rand index*, $\text{ARI}(U, U')$, is a bias-adjusted formulation developed by Hubert and Arabie [50]. To compute the ARI, we first harden the fuzzy partitions by setting the maximum element in each column of U to 1, and all else to 0. We use ARI to compare the clustering solutions with ground-truth labels (when available), as well as to compare the VL data algorithms with the literal FCM solutions.

Note that the rseFCM, spFCM, oFCM, and brFCM algorithms—and the analogous kernel variants—do not produce full data partitions; they produce cluster centers as output. Hence, we cannot directly compute the ARI for these algorithms. To complete the calculations, we used the extension step to produce full data partitions from the output cluster centers. The extension step was *not* included in the speedup factor or run-time calculations for these algorithms as these algorithms were originally designed to return cluster centers, not full data partitions. However, we observed in our experiments that the extension step comprised <1% of the overall run-time of the algorithms.

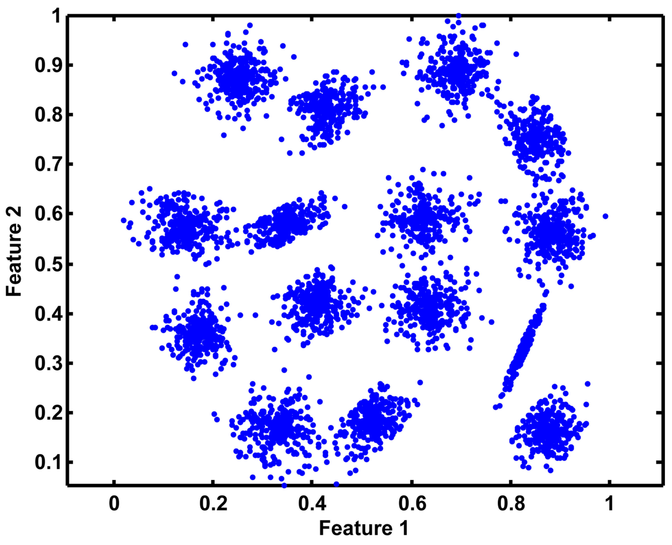


Fig. 2. 2D15 synthetic data.

B. Performance on Labeled Data

We compared the performance of the VL FCM algorithms on the following labeled datasets.

2D15[6] $(n = 5000, c = 15, d = 2)$: These data are composed of 5000 2-D vectors, with a visually preferred grouping into 15 clusters. Fig. 2 shows a plot of these data. For the kernel-based algorithms, an RBF kernel with $\sigma = 1$ was used.

MNIST $(n = 70\,000, c = 10, d = 784)$: This dataset is a subset of the collection of handwritten digits that are available from the *National Institute of Standards and Technology* (NIST).[7] There are 70 000 28×28 pixel images of the digits 0 to 9. Each

[5]There are many methods to determine the optimal fuzzifier m [46]–[48]. We have found that a fuzzifier $m = 1.7$ works well for most datasets.

[6]The 2D15 data were designed by I. Sidoroff and can be downloaded at http://cs.joensuu.fi/~isido/clustering/.

[7]The MNIST data can be downloaded at http://yann.lecun.com/exdb/mnist/.

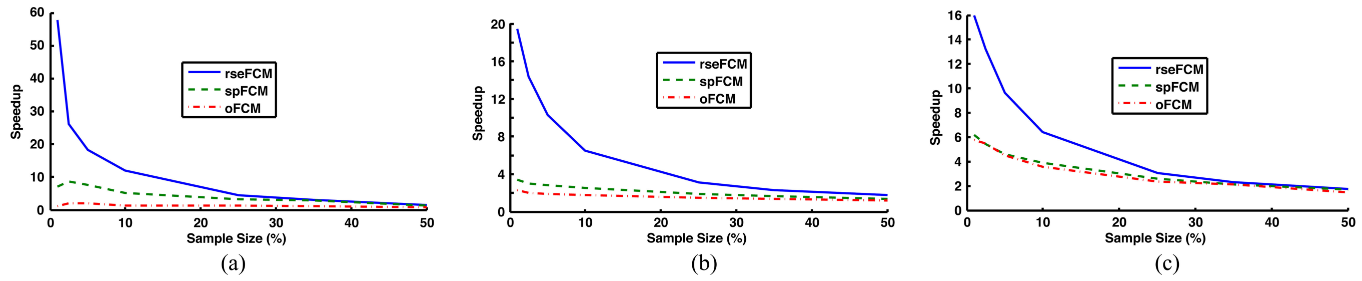


Fig. 3. Mean speedup of VL FCM vector data algorithms. (a) 2D15. (b) MNIST. (c) Forest.

TABLE III
ARI OF VL FCM VECTOR DATA ALGORITHMS

(a) 2D15

	Algorithm					
	rseFCM		spFCM		oFCM	
Sample	avg.	std.	avg.	std.	avg.	std.
size	min.	max.	min.	max.	min.	max.
1%	0.79	0.07	0.84	0.05	*0.97*	0.04
	0.64	0.97	0.76	0.99	0.89	0.99
2.5%	0.88	0.06	0.89	0.05	**0.97**	0.03
	0.72	0.99	0.77	0.99	0.89	0.99
5%	0.93	0.05	0.91	0.05	**0.98**	0.03
	0.82	0.99	0.81	0.99	0.89	0.99
10%	*0.95*	0.04	*0.95*	0.04	**0.97**	0.03
	0.89	0.99	0.89	0.99	0.88	0.99
25%	*0.96*	0.04	*0.95*	0.04	**0.98**	0.03
	0.89	0.99	0.89	0.99	0.89	0.99
35%	*0.96*	0.04	*0.96*	0.04	**0.98**	0.03
	0.89	0.99	0.89	0.99	0.89	0.99
50%	*0.96*	0.04	*0.96*	0.04	*0.97*	0.04
	0.89	0.99	0.89	0.99	0.80	0.99
LFCM			0.96	0.04		
			0.89	0.99		

(b) MNIST

	Algorithm					
	rseFCM		spFCM		oFCM	
Sample	avg.	std.	avg.	std.	avg.	std.
size	min.	max.	min.	max.	min.	max.
1%	*0.08*	0.03	**0.09**	0.07	**0.12**	0.04
	0.03	0.15	0.00	0.22	0.05	0.20
2.5%	*0.07*	0.03	**0.10**	0.04	**0.11**	0.03
	0.04	0.15	0.02	0.21	0.04	0.19
5%	*0.06*	0.02	*0.08*	0.04	**0.11**	0.03
	0.04	0.14	0.00	0.15	0.05	0.18
10%	*0.06*	0.02	*0.06*	0.03	**0.12**	0.03
	0.04	0.14	0.00	0.15	0.07	0.17
25%	*0.07*	0.03	**0.08**	0.03	**0.10**	0.02
	0.04	0.16	0.02	0.15	0.05	0.17
35%	*0.07*	0.03	**0.08**	0.03	**0.10**	0.03
	0.04	0.16	0.03	0.14	0.05	0.17
50%	*0.06*	0.02	**0.08**	0.03	**0.08**	0.02
	0.04	0.14	0.04	0.14	0.05	0.15
LFCM			0.07	0.02		
			0.04	0.13		

(c) Forest

	Algorithm					
	rseFCM		spFCM		oFCM	
Sample	avg.	std.	avg.	std.	avg.	std.
size	min.	max.	min.	max.	min.	max.
1%	*0.039*	0.013	*0.036*	0.007	0.033	0.003
	0.018	0.077	0.015	0.069	0.024	0.038
2.5%	*0.037*	0.008	*0.035*	0.003	0.032	0.004
	0.024	0.075	0.027	0.040	0.023	0.040
5%	*0.040*	0.010	*0.038*	0.012	0.033	0.004
	0.028	0.074	0.017	0.081	0.021	0.041
10%	*0.036*	0.006	*0.037*	0.006	0.033	0.004
	0.023	0.070	0.026	0.075	0.019	0.040
25%	*0.038*	0.011	*0.038*	0.011	0.033	0.005
	0.015	0.075	0.012	0.077	0.016	0.040
35%	*0.035*	0.005	*0.040*	0.012	*0.037*	0.011
	0.023	0.058	0.022	0.076	0.023	0.079
50%	*0.037*	0.008	*0.036*	0.007	0.033	0.011
	0.023	0.073	0.003	0.060	0.011	0.077
LFCM			0.037	0.009		
			0.027	0.077		

Values are calculated from 50 trials. Italic entries indicates that the mean ARI of the sampled algorithm was equal to the mean ARI of LFCM (at a 5% significance level). Bold values indicates that the mean ARI of the sampled algorithm was greater than the mean ARI of LFCM (at a 5% significance level).

pixel has an integer value between 0 and 255. We normalize the pixel values to the interval $[0, 1]$ by dividing by 255 and concatenate each image into a 784-dimensional feature vector. For the kernel-based algorithms, a degree 5 inhomogeneous polynomial kernel was used, which was shown to be effective in [19], [20], and [52].

Forest[8] $(n = 581\,012, c = 7, d = 54)$: These data are composed of cartographic variables that are obtained from *United States Geological Survey* and *United State Forest Service* (USFS) data [53]. There are 10 quantitative variables, such as elevation and horizontal distance to hydrology, 4 binary wilderness area designation variables, and 40 binary soil-type variables. These features were collected from a total of 581 012 30×30 m cells, which were then determined to be one of 7 forest cover types by the USFS. We normalized the features to the interval $[0, 1]$ by subtracting the minimum and then dividing by the subsequent maximum.

1) Vector Data Algorithms: The speedup results of the VL data algorithms on the 2D15 dataset are shown in Fig. 3(a), and the ARI results are displayed in Table III(a). The rseFCM algorithm is the fastest, with spFCM second, and oFCM third. The spFCM and oFCM algorithms also display degradation in speedup at the 1% sample size, where the number of data chunks is greater than the number of objects in each chunk (there are 50 objects in each of the 100 chunks). The ARI, in Table III(a), shows that the VL data algorithms perform on par with LFCM. Italicized entries indicate the that mean ARI of the respective sampled algorithm and LFCM are statistically equal.[9] Bold values indicate that the mean ARI of the respective sampled algorithm is greater than that of LFCM. This table clearly shows the degradation in accuracy exhibited by rseFCM and spFCM with small sample sizes. The best overall results, in terms of accuracy and speedup taken together, occur at the 10% sample size. Note that Table III shows three things. First, it shows the match of all four algorithms' partitions to the ground truth partition of the data. Second, it shows the match of the three approximation algorithms to LFCM. In this second comparison, oFCM best matches LFCM across the whole range of sample sizes because the oFCM results are statistically equal to or better than those of LFCM. Finally, the table shows that the standard deviation of the ARI, over the 50 runs, decreases as the sample size is increased—the algorithms are more consistent with larger sample sizes.

Fig. 3(b) shows the speedup of the VL data algorithms on the MNIST data. Not surprisingly, the rseFCM is again the fastest algorithm, with spFCM and oFCM showing very slight speedups. In Table III(b), we see that the oFCM has the highest ARI; however, notice that, overall, these ARI scores are much lower than what was seen with the 2D15 data. In essence, all the algorithms, including LFCM, are performing very poorly on these data (in terms of accuracy relative to ground-truth labels). These results are only slightly better than random assignment

[8]The *Forest* dataset can be downloaded at http://uisacad2.uis.edu/dstar/data/clusteringdata.html.

[9]We used a paired two-sided T-test at a 5% significance level, assuming unequal population variances.

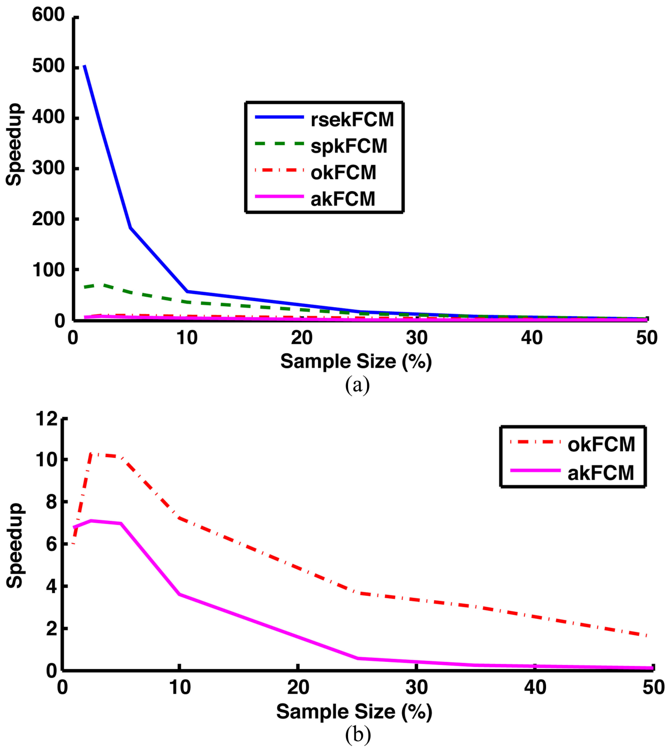


Fig. 4. Speedup of VL kernel FCM algorithms on 2D15 dataset. (a) Speedup. (b) akFCM/okFCM Speedup.

of cluster labels. Therefore, we hesitate to conclude that these results indicate that any algorithm is preferable, but of the three approximations, rseFCM seems to best match LFCM for the MNIST data across the range of sample sizes. Again, we see that the standard deviation of the results decreases as sample size increases. Interestingly, the spFCM and oFCM algorithms produced their best maximum ARI, 0.22 and 0.20, respectively, at the 1% sample size. We have seen this behavior in related studies [20] with this dataset. We believe that the effect of outliers is dampened at low sample sizes. Note that LFCM has the lowest maximum ARI of 0.13.

The results of the vector data clustering experiment on the Forest data are similar to results for the MNIST experiment. First, Fig. 3(c) shows that rseFCM is the most efficient algorithm; however, for these data, the spFCM and oFCM show a better relative speedup than with the 2D15 and MNIST data. This is because the Forest data are much larger than the other two datasets. Table III(c) shows that all the algorithms, including LFCM, show low accuracy compared with the ground-truth labels. Again, these results are not much better than random assignment but show that rseFCM and spFCM perform on par with LFCM. Interestingly, rseFCM does not show the same degradation in accuracy at small sample sizes, as was present in the previous experiments. We believe that this is because, even with small sample sizes, the number of objects in the sample is still quite large relative to the number of clusters. Hence, there is a greater probability that all the clusters are well represented in the reduced dataset. For these data, the oFCM algorithm suffers, producing a lower mean ARI than LFCM (except for the 35%

TABLE IV
ARI OF VL KERNEL FCM ALGORITHMS

(a) 2D15

	Algorithm							
	rsekFCM		spkFCM		okFCM		akFCM	
Sample size	avg. min.	std. max.	avg. min.	std. max.	avg. min.	std. max.	avg. min.	std. max.
1%	0.77	0.09	0.78	0.09	*0.92*	0.04	*0.90*	0.05
	0.64	0.90	0.57	0.90	0.89	0.99	0.81	0.99
2.5%	0.86	0.06	0.88	0.06	*0.91*	0.07	*0.92*	0.04
	0.77	0.99	0.81	0.98	0.82	0.99	0.89	0.99
5%	0.86	0.07	0.87	0.05	*0.92*	0.05	*0.92*	0.06
	0.74	0.99	0.75	0.91	0.82	0.99	0.81	0.99
10%	*0.90*	0.04	*0.90*	0.08	*0.94*	0.04	*0.90*	0.05
	0.82	0.99	0.74	0.99	0.89	0.99	0.81	0.99
25%	*0.92*	0.05	*0.90*	0.07	*0.90*	0.06	*0.91*	0.05
	0.83	0.99	0.76	0.99	0.77	0.99	0.82	0.99
35%	*0.89*	0.05	*0.91*	0.07	0.87	0.07	0.89	0.03
	0.82	0.99	0.82	0.99	0.73	0.99	0.80	0.91
50%	*0.89*	0.06	*0.90*	0.03	*0.89*	0.08	*0.91*	0.03
	0.80	0.99	0.88	0.99	0.80	0.99	0.89	0.99
kFCM				0.93	0.05			
				0.82	0.99			

(b) MNIST

	Algorithm							
	rsekFCM		spkFCM		okFCM		akFCM	
Sample size	avg. min.	std. max.	avg. min.	std. max.	avg. min.	std. max.	avg. min.	std. max.
0.5%	0.000	0.000	0.000	0.000	0.001	0.000	**0.027**	0.000
	0.000	0.000	0.000	0.000	0.001	0.001	0.027	0.027
1%	0.000	0.000	0.000	0.000	0.000	0.000	*0.027*	0.000
	0.000	0.000	0.000	0.000	0.000	0.000	0.027	0.027
2.5%	0.000	0.000	0.000	0.000	0.001	0.000	**0.027**	0.000
	0.000	0.000	0.000	0.000	0.001	0.001	0.027	0.027
5%	0.001	0.000	0.001	0.000	0.001	0.000	**0.027**	0.000
	0.000	0.001	0.001	0.001	0.001	0.001	0.027	0.027
10%	0.001	0.000	0.000	0.000	0.001	0.000		
	0.000	0.001	0.000	0.000	0.001	0.001		
kFCM				0.027	0.000			
				0.027	0.027			

(c) akFCM Small Sample Sizes

	akFCM	
Sample size	avg. min.	std. max.
0.02%	**0.033**	0.000
	0.033	0.033
0.04%	0.024	0.000
	0.024	0.024
0.07%	**0.029**	0.000
	0.029	0.029
0.1%	0.025	0.000
	0.025	0.025

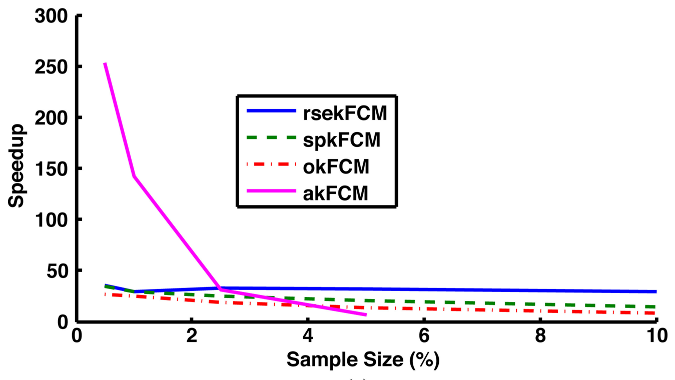
Values are calculated from 50 trials. Italic indicates that the mean ARI of the sampled algorithm was equal to the mean ARI of kFCM (at a 5% significance level). Bold indicates that the mean ARI of the sampled algorithm was greater than the mean ARI of kFCM (at a 5% significance level).

sample size). However, notice that the standard deviation of oFCM decreases with sample size—contrary to previous experiments. Hence, oFCM is producing a more consistent solution at smaller sample sizes, which is also supported by the greater minimum ARI at small sizes. Again, our conjecture is that outliers have less of an effect at small sample sizes. Even taking this into account, these results clearly show that oFCM is overall inferior for this dataset.

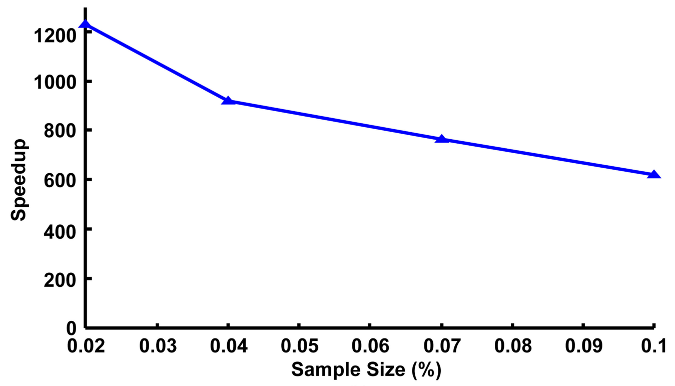
Let us turn now to the kernel-based clustering algorithms.

2) Kernel Algorithms: Fig. 4(a) shows the speedup of the VL kernel algorithms on the 2D15 data. The data were transformed with an RBF kernel. View (a) shows that the rsekFCM is the fastest algorithm, and spkFCM is the second fastest algorithm. It is difficult to see the akFCM and okFCM speedup results; hence, view (b) shows the speedup of the akFCM and okFCM algorithms separately. This plot shows that okFCM is slightly faster than akFCM. At sample sizes $>20\%$, the akFCM algorithm is actually slower than the literal kFCM because of the inverse calculation. As Table IV(a) shows, okFCM and akFCM maintain good performance, even at small sample sizes (later, we will further investigate the performance of akFCM at very small sample sizes). Furthermore, the standard deviation of the results shows that okFCM and akFCM produce very consistent solutions. The rsekFCM and spkFCM algorithms both suffer at small sample sizes, similar to what was seen in the 2D15 vector data experiment shown in Table III(a). At sample sizes $\geq 10\%$, all algorithms perform on par with literal kFCM. Comparing Table IV(a) with Table III(a) shows that the kernel algorithms (using the chosen kernel) are $\sim 5\%$ worse at matching the ground truth than the vector data counterparts. However, it has been shown that kernel clustering is very effective for many types of data; hence, this experiment shows that our algorithms are effective at producing clusters that are statistically equal to those by literal kernel FCM.

The results of the VL kernel algorithms on the MNIST data, shown in Fig. 5 and Table IV(b), tell a different story. Fig. 5(a) shows that akFCM is the most efficient at sample sizes $<2\%$, with a maximum speedup of about 180 at a 0.5% sample size. The other algorithms exhibit a nearly constant speedup of around 30 across all sample sizes tested. The most striking results are



(a)



(b)

Fig. 5. Performance of VL kernel FCM algorithms on MNIST dataset. (a) Speedup factor. (b) akFCM speedup factor.

the ARIs shown in Table IV(b). The akFCM algorithm performs as well (or slightly better) as the literal kFCM at all sample sizes; however, the other algorithms show markedly inferior performance. We believe that this is because the MNIST data do not cluster well relative to its 0–9 ground-truth labels. Hence, the cluster structure is degraded by the sampling aspect of the spkFCM, okFCM, and rsekFCM algorithms. The take-away lesson here, however, is that akFCM best captures the clusters identified by the literal kFCM algorithm.

To examine how akFCM behaves at very small sample sizes, we compiled additional results from akFCM on the MNIST dataset. Fig. 5(b) shows the speedup, and Table IV(c) shows the ARI for 0.02%, 0.04%, 0.07%, and 0.1% sample sizes. The akFCM algorithm is able to achieve a speedup factor of about 1200 at the 0.02% sample size, while still achieving good clustering performance relative to the literal kFCM.

C. Performance on Large Image Data

MR1-16, 17, 18: These data consist of the T1-channel of *magnetic resonance* (MR) images of the brain. Each dataset consists of an MRI volume with 96 (512×512) 12-bit images obtained in the axial plane using a 1.5-Tesla Siemens Sonata with a standard head coil. We have three volumes, denoted MR1-16, MR1-17, and MR1-18.[10] These images are first processed to remove air and skull pixels. The remaining pixels in each of the

[10]The three MRI datasets used in this paper correspond to the MN016, MN017, and MN018 datasets, respectively, used in [16].

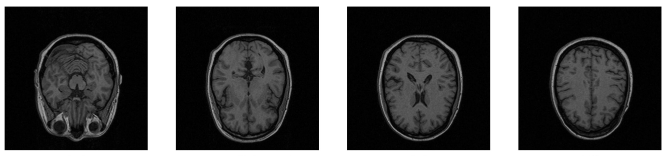


Fig. 6. 1-Channel (T1) MRI Slices, $c = 3$, $n = 512 \times 512 \times 96$ (from MR1-16). (a) Slice - 30. (b) Slice - 40. (c) Slice - 50. (d) Slice - 60.

96 images are unrolled into 1-D (pixel-value) feature vectors. The feature vectors from each of the 96 slices are combined, creating a dataset comprising about 4 million 1-D objects. Fig. 6 shows examples of four slices from the MR1-16 data (before skull and air removal).

MR3-16, 17, 18: These data are the 3-sequence (proton density (PD), T1, and T2) data of the MR1 datasets. Each dataset consists of an MRI volume with 96 3-sequence (512×512) 12-bit images obtained in the same data collection as the 1-channel data. We denote these data as MR3-16, MR3-17, and MR3-18. Like the MR1 datasets, the MR3 images are first processed to remove air and skull pixels. The remaining pixels in each of the 96 3-sequence images are unrolled into 3-D (one dimension per sequence) feature vectors: one for each pixel. These 3-D vectors from each of the 96 volume slices are combined, creating a dataset comprised of about 4 million 3-D objects.

We tested three sample sizes, 0.1%, 1%, and 10%. Speedup and ARI were calculated relative to the LFCM solution and averaged over 21 random initializations and samplings. The order in which the samples were presented to the algorithms was also randomized. Table V shows the performance of the VL clustering algorithms relative to the performance of LFCM. The bold values in each table indicate that one algorithm was statistically superior in performance, compared with the others (the mean value was greater than the other three algorithms). Italicized values indicate one algorithm was statistically inferior in performance (the mean value was less than the other three algorithms). Statistical significance was measured by a paired two-sided *T*-test at the 5% significance level.

Let us first look at the performance of the VL algorithms on the 1-D MRI image volumes at the 0.1% sample size. The left three columns of data in Table V(a) show the results. First, notice that the brFCM algorithm is clearly superior to the others. In all three image volumes, brFCM is able to achieve a speedup of >100, with the next best algorithm being rseFCM, with a speedup of about 20. Furthermore, brFCM is able to achieve a perfect ARI of 1, *with respect to the LFCM partition*, in all three image volumes. However, notice that the other algorithms also perform very well. If we had to choose a "worst" algorithm it would be spFCM; however, spFCM's worst performance is an ARI of 0.92 on volume 18, which is still a very accurate result. oFCM is the least efficient algorithm, but, like brFCM, it achieves a perfect ARI. At the 0.1% sample size, the number of cluster centers accumulated by oFCM is greater than the number of objects in each data chunk; hence, oFCM's efficiency is degraded.

Now, let us look at the 3-D MRI image experiments in the right three columns of Table V. For these volumes, we do not use

TABLE V
RESULTS OF VL FCM ALGORITHMS ON MRI DATA

(a) 0.1% sample size

Data Set	MR1						MR3					
Volume	16		17		18		16		17		18	
	SU	ARI	SU	ARI	SU	ARI	SU	ARI	SU	ARI	SU	ARI
rseFCM	22	0.97	21	0.99	21	1	**29**	0.97	**25**	0.97	**22**	0.97
spFCM	13	0.98	13	0.93	13	*0.92*	16	0.96	14	0.97	12	1
oFCM	*2*	1	*2*	1	*3*	1	*2*	*0.78*	*2*	*0.78*	*2*	*0.85*
brFCM	**108**	1	**108**	1	**107**	1						

(b) 1% sample size

Data Set	MR1						MR3					
Volume	16		17		18		16		17		18	
	SU	ARI	SU	ARI	SU	ARI	SU	ARI	SU	ARI	SU	ARI
rseFCM	18	0.99	18	1	18	1	**24**	1	**21**	0.99	**18**	1
spFCM	13	*0.98*	12	0.99	12	0.98	13	0.96	11	0.98	10	1
oFCM	*4*	1	*4*	1	*4*	1	*2*	*0.93*	*2*	*0.78*	*2*	1
brFCM	**50**	1	**52**	1	**53**	1						

(c) 10% sample size

Data Set	MR1						MR3					
Volume	16		17		18		16		17		18	
	SU	ARI	SU	ARI	SU	ARI	SU	ARI	SU	ARI	SU	ARI
rseFCM	7	1	7	1	7	1	**8**	1	**8**	1	**6**	1
spFCM	8	0.98	8	0.99	8	0.98	7	0.96	6	0.97	5	1
oFCM	4	1	4	1	4	1	3	1	2	1	2	1
brFCM	8	1	**9**	1	**9**	1						

Bold indicates that the mean value was greater than the mean value of other three algorithms (at 5% significance level). Italic indicates that the mean value was less than the mean value of other three algorithms (at 5% significance level).

TABLE VI
RESULTS OF VL VECTOR DATA ALGORITHMS ON 4D3 UNLOADABLE DATA

	Run-time (sec)						$\sum \|\|V - \mu\|\|$					
Sample size	0.001%		0.01%		0.1%		0.001%		0.01%		0.1%	
	avg.	std.	avg.	std.	avg.	std.	avg.	std.	avg.	std.	avg.	std.
rseFCM	0.004	0.010	0.30	0.09	3.1	0.9	0.54	0.11	0.31	0.04	0.28	0.02
spFCM	1.3 hrs	0.2 hrs	1.1 hrs	0.2 hrs	1.1 hrs	0.1 hrs	0.28	0	0.28	0	0.28	0
oFCM	9.1 hrs	0.9 hrs	3.2 hrs	0.4 hrs	1.8 hrs	0.2 hrs	0.28	0	0.28	0	0.28	0
LFCM*	avg. 38** std. 9.3						avg. 0.28 std. 0					

*LFCM results run on a 50 million-sized dataset with same 4D3 distribution.
**Extrapolated to 5 billion-sized dataset, LFCM would require approximately 1 h.

brFCM, as it was designed for 1-D (grayscale) images where binning is quick and efficient. For these volumes, rseFCM is the preferred algorithm, with a speedup of about 25, and an ARI near to 1. spFCM performs comparably with rseFCM, but at a slightly lesser speedup. Again, oFCM is slow when compared with the other algorithms. And, for these images, oFCM produces noticeably inferior ARI results.

The results in these three tables are pretty important, as many VL datasets will be feature vectors from VL images. We strongly recommend brFCM as the preferred algorithm for 1D image data, while rseFCM seems best for VL image data in more than one dimension. On these data, the speedup of rseFCM was significantly degraded by the sampling procedure; hence, if more efficient sampling is used, we anticipate improvement in the speed of rseFCM.

D. Performance on Unloadable Data

For our last experiment, we demonstrate the VL vector data FCM algorithms on an unloadable dataset. Because we cannot compare with LFCM for this data size, we constructed a dataset that should be accurately clustered. Hence, the performance of the VL algorithms can be measured by how well they find the apparent clusters and measuring the run-time in seconds. The 4D3 data are composed of 5 billion objects that are randomly drawn (with equal probability) from three 4-D Gaussian distributions. The parameters of the distributions are $\mu_1 = (0, 0, 0, 0)$, $\mu_2 = (5, 5, 0, 0)$, $\mu_3 = (0, 0, 5, 5)$, and $\Sigma_1 = \Sigma_2 = \Sigma_3 = I$. To cluster this dataset using LFCM would require nearly 300 GB of memory. In order to show how the VL algorithms could be used to process this dataset on a normal PC, we tested at sample sizes of 0.001%, 0.01%, and 0.1%, requiring approximately 3, 30, and 300 MB of memory, respectively. We also compared against LFCM run on a 50 million-sized dataset with the same distribution.

Table VI shows the results of this experiment. Because the dataset is so large, we skipped the extension step and had the algorithms only return cluster centers. Hence, we are unable to compute ARI directly. Instead, we use the sum of the distances between the cluster centers and the true means of the three

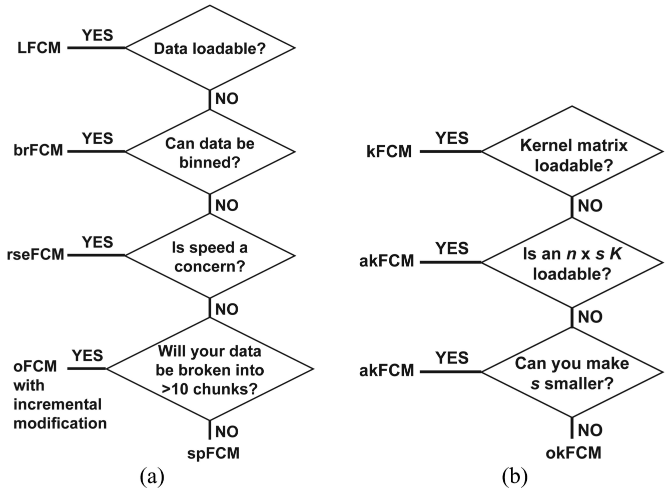


Fig. 7. Recommendation trees of VL FCM and kFCM algorithms. (a) FCM Algorithms. (b) kFCM Algorithms.

Gaussian clouds to show accuracy. As expected, rseFCM is the fastest algorithm by a significant margin. However, at small samples sizes, rseFCM suffers; the average distance between the cluster centers and the true distribution means increases. Furthermore, the rseFCM results are less consistent, as evidenced by the standard deviation. In contrast, spFCM and oFCM achieve virtually the same solution every time, regardless of the sample size. To put this result in perspective, we ran LFCM on a manageably sized dataset—50 million objects—with the same 4D3 distribution. Over 50 runs, LFCM showed the same clustering accuracy as both spFCM and oFCM, which further supports our claim that the streaming VL FCM schemes are effective at achieving the same partitions as LFCM. Finally, if we extrapolate the LFCM run-time to 5 billion objects, the algorithm would require approximately 1 h, which is on par with spFCM's run-time. We stress, though, that running LFCM on this dataset is impossible for the system we have available. Finally, oFCM displays a longer run-time than spFCM because of the last clustering step and data accumulation overhead. In the future, we will examine distributed architectures, which may lend well to hybrid instantiations of oFCM and spFCM (ideally harnessing the strengths of both).

VI. Discussion and Conclusions

As this paper shows, there are many ways to attack clustering of VL data with FCM. Fig. 7 summarizes our recommendations for using the algorithms we tested. Note that we assume that time is not the predominant problem; hence, accuracy and feasibility are the main focus points, with efficiency (or acceleration) a secondary concern. Hence, if your data can be loaded into memory, we suggest using the literal implementation of FCM or kFCM. Some of our experiments suggest that improvement in accuracy can be obtained by the VL data algorithms, but this improvement was always negligible.

If your data cannot be loaded into memory, then you must first choose whether you are going to cluster the vector data directly or use a kernel method (kernel methods could help you find nonspherical clusters, but are computationally expensive). If your vector data can be binned efficiently (and accurately), we suggest the brFCM algorithm. This algorithm had the best performance in our experiments on clustering large 1-D MRI image volumes and is scalable. The oFCM algorithm produced results equal to LFCM at most sample sizes for the vector data; however, this algorithm accumulates a set of cluster centers from each data chunk. For extremely large data, this accumulation can be a problem. Furthermore, oFCM was the least efficient of the VL vector data algorithms and was the least accurate for the 3-D MRI images. The rseFCM, spFCM, rsekFCM, and spkFCM are the only scalable solutions for multidimensional data (the binning aspect of brFCM is troublesome for multidimensional data), but oFCM and okFCM can be made scalable by using a scalable algorithm for the last clustering step or by incrementally clustering as centers accumulate. The spFCM and spkFCM algorithms are the only scalable algorithms here that use the entire dataset but performance suffers at low sample rates. Hence, we recommend an incremental application of oFCM and okFCM for extremely large datasets where brFCM and akFCM are infeasible. Recall that akFCM produced very comparable clusters to kFCM at very small sample sizes; hence, always consider whether s could be further reduced as akFCM is empirically and theoretically an accurate solution.

In the future, we will continue to develop and investigate scalable solutions for VL fuzzy clustering. Our experiments showed that the rsekFCM, spkFCM, and okFCM produced less-than-desirable results, compared with the literal kFCM solution. Hence, we are going to examine other ways by which the kFCM solution can be approximated for VL data, with an emphasis on scalability and accuracy. Furthermore, we wish to examine where kernel solutions would be best used. Is it possible to use cluster validity indices to choose the appropriate kernel or to choose when a kernelized algorithm is appropriate? We will look at this in the future.

Another question that arises in clustering of VL data is validity or, in other words, the quality of the clustering. Many cluster validity measures require full access to the objects' vector data or to the full kernel (or relational) matrix. Hence, we aim to extend some well-known cluster validity measures for use on VL data by using similar extensions as presented here.

The only algorithms proposed that are not called "incremental" are LFCM, rseFCM, akFCM, and rsekFCM. While the other algorithms do process data in a distributed fashion, they really process it "one chunk at a time." We have yet to encounter a truly incremental version of FCM (or any other VL clustering scheme) that is capable of true online streaming analysis, that is, an algorithm that operates "one vector at a time," as they arrive from a sensor. This important objective will be a major focus of our ongoing research about clustering VL data.

Finally, we would like to emphasize that clustering algorithms, by design, are meant to find the natural groupings in *unlabeled* data (or to discover unknown trends in labeled data). Thus, the effectiveness of a clustering algorithm cannot be appropriately judged by pretending it a classifier and presenting classification results on labeled data, where each cluster is considered to be a class label. Although we did compare against ground-truth labels in this paper, we used these experiments to show that some of the VL fuzzy clustering schemes were successful in producing similar partitions to those produced by literal FCM, which was our bellwether of performance. This will continue to be our standard for the work ahead.

REFERENCES

[1] H. Frigui, "Simultaneous clustering and feature discrimination with applications," in *Advances in Fuzzy Clustering and Feature Discrimination With Applications*. New York: Wiley, 2007, pp. 285–312.

[2] W. Bo and R. Nevatia, "Cluster boosted tree classifier for multi-view, multi-pose object detection," in *Proc. Int. Conf. Comput. Vision*, Oct. 2007, pp. 1–8.

[3] S. Khan, G. Situ, K. Decker, and C. Schmidt, "GoFigure: Automated Gene Ontology annotation," *Bioinformatics*, vol. 19, no. 18, pp. 2484–2485, 2003.

[4] The UniProt Consotium, "The universal protein resource (UniProt)," *Nucleic Acids Res.,* 35, pp. D193–D197, 2007.

[5] S. Gunnemann, H. Kremer, D. Lenhard, and T. Seidl, "Subspace clustering for indexing high dimensional data: A main memory index based on local reductions and individual multi-representations," in *Proc. Int. Conf. Extend. Database Technol.*, Uppsala, Sweden, 2011, pp. 237–248.

[6] A. Jain and R. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice–Hall, 1988.

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

[8] R. Xu and D. Wunsch, II, *Clustering*. Piscataway, NJ: IEEE Press, 2009.

[9] D. Johnson and D. Wichern, *Applied Multivariate Statistical Analysis*, 6th ed. Englewood Cliffs, NJ: Prentice–Hall, 2007.

[10] J. Hartigan, *Clustering Algorithms*. New York: Wiley, 1975.

[11] P. Huber, "Massive data sets workshop: The morning after," in *Massive Data Sets*. Washington, DC: Nat. Acad., 1997, pp. 169–184.

[12] R. Hathaway and J. Bezdek, "Extending fuzzy and probabilistic clustering to very large data sets," *Comput. Statist. Data Anal.*, vol. 51, pp. 215–234, 2006.

[13] J. Bezdek, *Pattern Recognition With Fuzzy Objective Function Algorithms*. New York: Plenum, 1981.

[14] R. Krishnapuram and J. Keller, "A possibilistic approach to clustering," *IEEE Trans. Fuzzy Syst.*, vol. 1, no. 2, pp. 98–110, May 1993.

[15] N. Pal and J. Bezdek, "Complexity reduction for "large image" processing," *IEEE Trans. Syst., Man, Cybern.*, vol. 32, no. 5, pp. 598–611, Oct. 2002.

[16] P. Hore, L. Hall, and D. Goldgof, "Single pass fuzzy c means," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, London, U.K., 2007, pp. 1–7.

[17] P. Hore, L. Hall, D. Goldgof, Y. Gu, and A. Maudsley, "A scalable framework for segmenting magentic resonance images," *J. Signal Process. Syst.*, vol. 54, no. 1–3, pp. 183–203, Jan. 2009.

[18] S. Eschrich, J. Ke, L. Hall, and D. Goldgof, "Fast accurate fuzzy clustering through data reduction," *IEEE Trans. Fuzzy Syst.*, vol. 11, no. 2, pp. 262–269, Apr. 2003.

[19] R. Chitta, R. Jin, T. Havens, and A. Jain, "Approximate kernel k-means: Solution to large scale kernel clustering," in *Proc. ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2011, pp. 895–903.

[20] T. Havens, R. Chitta, A. Jain, and R. Jin, "Speedup of fuzzy and possibilistic c-means for large-scale clustering," in *Proc. IEEE Int. Conf. Fuzzy Systems*, Taipei, Taiwan, 2011, pp. 463–470.

[21] S. Guha, R. Rastogi, and K. Shim, "CURE: An efficient clustering algorithm for large databases," *Inf. Syst.*, vol. 26, no. 1, pp. 35–58, 2001.

[22] S. Har-Peled and S. Mazumdar, "On coresets for k-means and k-median clustering," in *Proc. ACM Symp. Theory Comput.*, 2004, pp. 291–300.

[23] F. Can, "Incremental clustering for dynamic information processing," *ACM Trans. Inf. Syst.*, vol. 11, no. 2, pp. 143–164, 1993.

[24] F. Can, E. Fox, C. Snavely, and R. France, "Incremental clustering for very large document databases: Initial MARIAN experience," *Inf. Sci.*, vol. 84, no. 1–2, pp. 101–114, 1995.

[25] C. Aggarwal, J. Han, J. Wang, and P. Yu, "A framework for clustering evolving data streams," in *Proc. Int. Conf. Very Large Databases*, 2003, pp. 81–92.

[26] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 3, pp. 515–528, May/Jun. 2003.

[27] T. Zhang, R. Ramakirshnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 1996, pp. 103–114.

[28] R. Ng and J. Han, "CLARANS: A method for clustering objects for spatial data mining," *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 5, pp. 1003–1016, Sep./Oct. 2002.

[29] R. Orlandia, Y. Lai, and W. Lee, "Clustering high-dimensional data using an efficient and effective data space reduction," in *Proc. ACM Conf. Inf. Knowl. Manag.*, 2005, pp. 201–208.

[30] G. Karypis, "CLUTO: A clustering toolkit," Dept. Comput. Sci., Univ. Minnesota, Minneapolis, MN, Tech. Rep. 02-017, 2003.

[31] B. U. Shankar and N. Pal, "FFCM: An effective approach for large data sets," in *Proc. Int. Conf. Fuzzy Logic, Neural Nets, Soft Comput.*, Fukuoka, Japan, 1994, p. 332.

[32] T. Cheng, D. Goldgof, and L. Hall, "Fast clustering with application to fuzzy rule generation," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Tokyo, Japan, 1995, pp. 2289–2295.

[33] J. Kolen and T. Hutcheson, "Reducing the time complexity of the fuzzy c-means algorithm," *IEEE Trans. Fuzzy Syst.*, vol. 10, no. 2, pp. 263–267, Apr. 2002.

[34] R. Cannon, J. Dave, and J. Bezdek, "Efficient implementation of the fuzzy c-means algorithm," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, no. 2, pp. 248–255, Mar. 1986.

[35] L. Liao and T. Lin, "A fast constrained fuzzy kernel clustering algorithm for MRI brain image segmentation," in *Proc. Int. Conf. Wavelet Anal. Pattern Recognit.*, Beijing, China, 2007, pp. 82–87.

[36] F. Provost, D. Jensen, and T. Oates, "Efficient progressive sampling," in *Proc. Knowledge Discovery and Data Mining*, 1999, pp. 23–32.

[37] P. Drineas and M. Mahoney, "On the Nystrom method for approximating a gram matrix for improved kernel-based learning," *J. Mach. Learn. Res.*, vol. 6, pp. 2153–2175, 2005.

[38] S. Kumar, M. Mohri, and A. Talwalkar, "Sampling techniques for the Nystrom method," in *Proc. Conf. Artif. Intell. Statist.*, 2009, pp. 304–311.

[39] M. Belabbas and P. Wolfe, "Spectral methods in machine learning and new strategies for very large datasets," *Proc. Nat. Acad. Sci. U.S.A.*, vol. 106, no. 2, pp. 369–374, 2009.

[40] J. Bezdek and R. Hathaway, "Convergence of alternating optmization," *Nueral, Parallel, Sci. Comput.*, vol. 11, no. 4, pp. 351–368, Dec. 2003.

[41] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, "Streaming-data algorithms for high-quality clustering," in *Proc. IEEE Int. Conf. Data Eng.*, Mar. 2002, pp. 685–694.

[42] L. Hall and D. Goldgof, "Convergence of the single-pass and online fuzzy c-means algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 19, no. 4, pp. 792–794, Aug. 2011.

[43] Z. Wu, W. Xie, and J. Yu, "Fuzzy c-means clustering algorithm based on kernel method," in *Proc. Int. Conf. Comput. Intell. Multimedia Appl.*, Sep. 2003, pp. 49–54.

[44] R. Hathaway, J. Davenport, and J. Bezdek, "Relational duals of the c-means clustering algorithms," *Pattern Recognit.*, vol. 22, no. 2, pp. 205–212, 1989.

[45] R. Hathaway, J. Huband, and J. Bezdek, "A kernelized non-euclidean relational fuzzy c-means algorithm," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, 2005, pp. 414–419.

[46] D. Dembele and P. Kastner, "Fuzzy c-means method for clustering microarray data," *Bioinformatics*, vol. 19, pp. 973–980, 2003.

[47] M. Futschik and B. Carlisle, "Noise-robust soft clustering of gene expression time-course data," *J. Bioinform. Comput. Biol.*, vol. 3, pp. 965–988, 2005.

[48] V. Schwammle and O. Jensen, "A simple and fast method to determine the parameters for fuzzy c-means cluster analysis," *Bioinformatics*, vol. 26, no. 22, pp. 2841–2848, 2010.

[49] W. Rand, "Objective criteria for the evaluation of clustering methods," *J. Amer. Stat. Asooc.*, vol. 66, no. 336, pp. 846–850, 1971.

[50] L. Hubert and P. Arabie, "Comparing partitions," *J. Class.*, vol. 2, pp. 193–218, 1985.

[51] D. Anderson, J. Bezdek, M. Popescu, and J. Keller, "Comparing fuzzy, probabilistic, and possibilistic partitions," *IEEE Trans. Fuzzy Syst.*, vol. 18, no. 5, pp. 906–917, Oct. 2010.

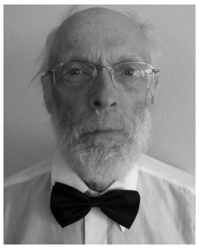
[52] R. Zhang and A. Rudnicky, "A large scale clustering scheme for kernel k-means," in *Proc. Int. Conf. Pattern Recognit.*, 2002, pp. 289–292.

[53] A. Asuncion and D. Newman. (2007). "UCI machine learning repository," [Online]. Available:http://www.ics.uci.edu/ mlearn/MLRepository.html.

Timothy C. Havens (SM'10) received the M.S. degree in electrical engineering from Michigan Tech University, Houghton, in 2000 and the Ph.D. degree in electrical and computer engineering from the University of Missouri, Columbia, in 2010.

Prior to his Ph.D. research, he was an Associate Technical Staff with the MIT Lincoln Laboratory. He is currently an NSF/CRA Computing Innovation Fellow with Michigan State University, East Lansing, and will be the William and Gloria Jackson Assistant Professor of Computer Systems with the Departments of Electrical and Computer Engineering and Computer Science at Michigan Tech University in August, 2012. He has published more than 50 journal articles, conference papers, and book chapters.

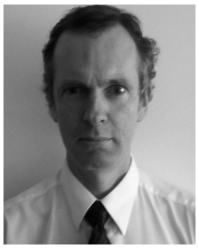
Dr. Havens received the IEEE Franklin V. Taylor Award for Best Paper at the 2011 IEEE Conference on Systems, Man, and Cybernetics and the Best Paper Award from the Midwest Nursing Research Society in 2009.



James C. Bezdek (LF'10) received the Ph.D. degree in applied mathematics from Cornell University, Ithaca, NY, in 1973.

He retired in 2007. His research interests include optimization, pattern recognition, clustering in very large data, coclustering, and visual clustering.

Dr. Bezdek was the President of the North American Fuzzy Information Processing Society, the International Fuzzy Systems Association (IFSA), and the IEEE Computational Intelligence Society (CIS). He was the founding Editor of the *International Journal of Approximate Reasoning* and the IEEE TRANSACTIONS FUZZY SYSTEMS. He received the IEEE Third Millennium, IEEE CIS Fuzzy Systems Pioneer, and IEEE (TFA) Rosenblatt and Kampe de Feriet medals. He is a Life Fellow of the IFSA.



Christopher Leckie received the B.Sc. degree in 1985, the B.E. degree in electrical and computer systems engineering in 1987, and the Ph.D. degree in computer science in 1992, all from Monash University, Melbourne, Vic., Australia.

He joined Telstra Research Laboratories in 1988, where he researched and developed artificial intelligence techniques for various telecommunication applications. In 2000, he joined the University of Melbourne, where he is currently an Associate Professor with the Department of Computer Science and Software Engineering. His research interests include scalable data mining, network intrusion detection, bioinformatics, and wireless sensor networks.

Lawrence O. Hall (F'03) received the Ph.D. degree in computer science from Florida State University, Tallahassee, in 1986.

He is a Distinguished University Professor and the Chair of the Department of Computer Science and Engineering, University of South Florida, Tampa.

Dr. Hall is a fellow of the International Association of Pattern Recognition. He received the IEEE Systems, Man, and Cybernetics (SMC) Society Outstanding Contribution Award in 2008 and the Outstanding Research Achievement Award from the University of South Florida in 2004. He has been the President of the North American Fuzzy Information Processing Society and the IEEE SMC Society. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS from 2002 to 2005. He is currently the Vice President for Publications of the IEEE Biometrics Council and an Associate Editor for several journals, including the IEEE TRANSACTIONS ON FUZZY SYSTEMS.

Marimuthu Palaniswami (F'12) received the M.E. degree from the Indian Institute of Science, Bangalore, India, the M.Eng.Sc. degree from the University of Melbourne, Australia, and the Ph.D. degree from the University of Newcastle, N.S.W., Australia.

He currently leads the ARC Research Network on Intelligent Sensors, Sensor Networks, and Information Processing program at the University of Melbourne. His research interests include support vector machines, sensors and sensor networks, machine learning, neural networks, pattern recognition, signal processing, and control. He has published over 340 refereed research papers.

Dr. Palaniswami has been a panel member for the National Science Foundation, an advisory board member for the European FP6 grant centre, a steering committee member for National Collaborative Research Infrastructure Strategy, Great Barrier Reef Ocean Observing System, and Software Engineering Method and Theory, and a board member for various information technology and supervisory control and data acquisition companies.