Incremental Kernel Fuzzy c-Means

Article *in* Computational Intelligence · January 2012

DOI: 10.1007/978-3-642-27534-0_1

3 authors:

Timothy Craig Havens
Michigan Technological University
194 PUBLICATIONS **3,834** CITATIONS
SEE PROFILE

James C. Bezdek
University of Missouri
428 PUBLICATIONS **64,011** CITATIONS
SEE PROFILE

Marimuthu Palaniswami
University of Melbourne
652 PUBLICATIONS **32,017** CITATIONS
SEE PROFILE

Incremental Kernel Fuzzy c-Means

Timothy C. Havens[1], James C. Bezdek[2], and Marimuthu Palaniswami[2]

[1] Michigan State University, East Lansing, MI 48824, E-mail: havenst@gmail.com
[2] University of Melbourne, Parkville, Victoria 3010, Australia,
E-mail: jcbezdek@gmail.com, palani@unimelb.edu.au

Abstract. The size of everyday data sets is outpacing the capability of computational hardware to analyze these data sets. Social networking and mobile computing alone are producing data sets that are growing by terabytes *every day*. Because these data often cannot be loaded into a computer's working memory, most literal algorithms (algorithms that require access to the full data set) cannot be used. One type of pattern recognition and data mining method that is used to analyze databases is clustering; thus, clustering algorithms that can be used on large data sets are important and useful. We focus on a specific type of clustering: kernelized fuzzy c-means (KFCM). The literal KFCM algorithm has a memory requirement of $O(n^2)$, where n is the number objects in the data set. Thus, even data sets that have nearly 1,000,000 objects require terabytes of working memory—infeasible for most computers. One way to attack this problem is by using incremental algorithms; these algorithms sequentially process chunks or samples of the data, combining the results from each chunk. Here we propose three new incremental KFCM algorithms: rseKFCM, spKFCM, and oKFCM. We assess the performance of these algorithms by, first, comparing their clustering results to that of the literal KFCM and, second, by showing that these algorithms can produce reasonable partitions of large data sets. In summary, the rseKFCM is the most efficient of the three, exhibiting significant speedup at low sampling rates. The oKFCM algorithm seems to produce the most accurate approximation of KFCM, but at a cost of low efficiency. Our recommendation is to use rseKFCM at the highest sample rate allowable for your computational and problem needs.

1 Introduction

The ubiquity of personal computing technology, especially mobile computing, has produced an abundance of staggeringly large data sets—Facebook alone logs over 25 terabytes (TB) of data per day. Hence, there is a great need for algorithms that can address these gigantic data sets. In 1996, Huber [24] classified data set size as in Table 1. Bezdek and Hathaway [17] added the *Very Large* (VL) category to this table in 2006. Interestingly, data with $10^{>12}$ objects is still unloadable on most current (circa 2011) computers. For example, a data set composed of 10^{12} objects, each with 10 features, stored in short integer (4 byte) format would require 40 TB of storage (most high-performance computers have $<$ 1 TB of working memory). Hence, we believe that Table 1 will continue to be pertinent for many years.

Clustering, also called unsupervised learning, numerical taxonomy, typology, and partitioning [41], is an integral part of computational intelligence and machine learning. Often researchers are mired in data sets that are large and unlabeled. There are

Table 1: Huber's Description of Data Set Sizes [17, 24]

Bytes	10^2	10^4	10^6	10^8	10^{10}	10^{12}	$10^{>12}$	∞
"size"	tiny	small	medium	large	huge	monster	**VL**	infinite

many methods by which researchers can elucidate these data, including projection and statistical methods. Clustering provides another tool for deducing the nature of the data by providing labels that describe how the data separates into groups. These labels can be used to examine the similarity and dissimilarity among and between the grouped objects. Clustering has also been shown to improve the performance of other algorithms or systems by separating the problem-domain into manageable sub-groups—a different algorithm or system is tuned to each cluster [6, 14]. Clustering has also been used to infer the properties of unlabeled objects by clustering these objects together with a set of labeled objects (of which the properties are well understood) [29, 40].

The problem domains and applications of clustering are innumerable. Virtually every field, including biology, engineering, medicine, finance, mathematics, and the arts, have used clustering. Its function—grouping objects according to context—is a basic part of intelligence and is ubiquitous to the scientific endeavor. There are many algorithms that extract groups from unlabeled object sets: k-means [33–35] and c-means [3], and hierarchical clustering [28] being, arguably, the most popular. We will examine a specific, but general, form of clustering: incremental *kernel fuzzy c-means* (KFCM). Specifically, we will develop three new kernel fuzzy clustering algorithms, *random sample and extend* KFCM (rseKFCM), *single-pass* KFCM (spKFCM), and *online* KFCM (oKFCM). The spKFCM and oKFCM algorithms are based on an extension of the *weighted* FCM and weighted kernel k-means models proposed in [13, 22, 23] and [10], respectively.

1.1 The Clustering Problem

Consider a set of objects $O = \{o_1, \ldots, o_n\}$. These objects can represent virtually anything—vintage bass guitars, pure-bred cats, cancer genes expressed in a microarray experiment, cake recipes, or web-pages. The object set O is *unlabeled data*; that is, each object has no associated class label. However, it is assumed that there are subsets of similar objects in O. These subsets are called *clusters*.

Numerical object data is represented as $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^p$, where each dimension of the vector $\mathbf{x}_i$ is a feature value of the associated object o_i. These features can be a veritable cornucopia of numerical descriptions, i.e., RGB values, gene expression, year of manufacture, number of stripes, et cetera.

A wide array of algorithms exists for clustering unlabeled object data O. Descriptions of many of these algorithms can be found in the following general references on clustering: [3, 5, 12, 15, 26, 27, 41, 44]. *Clustering* in unlabeled data X is defined as the assignment of *labels* to groups of similar (unlabeled) objects O. In other words, objects are *sorted* or *partitioned* into groups such that each group is composed of objects with similar traits. There are two important factors that all clustering algorithms must consider: 1) the number (and, perhaps, type) of clusters to seek and, 2) a mathematical way to determine the similarity between various objects (or groups of objects). Let c denote the integer number of clusters. The number of clusters can take the values $c = 1, 2, \ldots, n$, where $c = 1$ results in the universal cluster (every object is in one cluster) and $c = n$ results in single-object clusters.

A *partition* of the objects is defined as the set of cn values, where each value $\{u_{ij}\}$ represents the degree to which an object o_i is in (or represented by) the jth cluster. The c-partition is often arrayed as a $n \times c$ matrix $U = [u_{ij}]$, where each column represents a cluster and each row represents an object. There are three types of partitions (to date), crisp, fuzzy (or probabilistic), and possibilistic [3, 31] (we do not address possibilistic clustering here).

Crisp partitions of the unlabeled objects are non-empty mutually-disjoint subsets of O such that the union of the subsets cover O. The set of all non-degenerate (no zero columns) crisp c-partition matrices for the object set O is:

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

where u_{ij} is the *membership* of object o_i in cluster j; the partition element $u_{ij} = 1$ if o_i is labeled j and is 0 otherwise.

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, the partition values are interpreted as the posterior probability $p(j|o_i)$ that o_i is in the j-th class. We assume that fuzzy and probabilistic partitions are essentially equivalent from the point of view of clustering algorithm development. The set of all fuzzy c-partitions is:

$$M_{fcn} = \{U \in \mathbb{R}^{cn} | 0 \le u_{ij} \le 1 \,\forall i,j; \sum_{j=1}^{c} u_{ij} = 1 \,\forall i; \sum_{i=1}^{n} u_{ij} > 0 \,\forall j\}. \tag{2}$$

Each row of the fuzzy partition U must sum to 1, thus ensuring that every object is completely partitioned ($\sum_i u_{ij} = 1$).

Notice that all crisp partitions are fuzzy partitions, $M_{hcn} \subset M_{fcn}$. Hence, the methods applied here can be easily generalized to kernel HCM.

1.2 FCM

The FCM model is defined as the constrained minimization of

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

where $m \ge 1$ is a fixed fuzzifier and $|| \cdot ||_A$ is any inner product A-induced norm on $\mathbb{R}^d$, i.e., $||\mathbf{x}||_A = \mathbf{x}^T A \mathbf{x}$. Optimal c-partitions U are most popularly sought by using *alternating optimization* (AO) [3, 4], but other methods have also been proposed. The *literal* FCM/AO (LFCM/AO) algorithm is outlined in Algorithm 1. There are many ways to initialize LFCM/AO; any method that covers the object space and does not

```
Algorithm 1: LFCM/AO to minimize $J_m(U,V)$ [3]
Input: $X$, $c$, $m$
Output: $U$, $V$
Initialize $V$
while $\max\{||V_{new} - V_{old}||^2\} > \epsilon$ 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)
```

produce identical initial cluster centers would work. We initialize by randomly selecting c feature vectors from the data to serve as initial centers.

The alternating steps of LFCM in Eqs. (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, $\max\{||V - V_{old}||^2\} < \epsilon$, where ϵ is a pre-determined constant (we use $\epsilon = 10^{-3}$ in our experiments).

It was shown in [2, 18, 42] that minimizing (3) produces the same result as minimizing the reformulation,

$$J_m(U) = \sum_{j=1}^{c} \left(\sum_{i=1}^{n} \sum_{k=1}^{n} \left(u_{ij}^m u_{kj}^m d_A^2(\mathbf{x}_i, \mathbf{x}_k) \right) / 2 \sum_{l=1}^{n} u_{lj}^m \right), \qquad (6)$$

where $d_A^2(\mathbf{x}_i, \mathbf{x}_k) = ||\mathbf{x}_i - \mathbf{x}_k||_A^2$. This reformulation led to *relational* algorithms, such as RFCM [19] and NERFCM [16], where the data take the relational form $R_A = [||\mathbf{x}_i - \mathbf{x}_j||_A^2] \in \mathbb{R}^{n \times n}$. Later, we will use (6) to define the KFCM model.

1.3 Related Work on FCM for VL Data

There has been a bevy of research done on clustering in VL data, but only a small portion of this research addresses the fuzzy clustering problem. Algorithms fall in three main categories: i) *Sample and Extend* schemes apply clustering to a (manageably-sized) sample of the full data set, and then non-iteratively extend the sample results to approximate the clustering solution for the remaining data. These algorithms have also been called *extensible* algorithms [36]. An extensible FCM algorithm includes the geFFCM [17]. ii) *Incremental* algorithms sequentially load small groups or singletons of the data, clustering each chunk in a single pass, and then combining the results from each chunk. The SPFCM [22] algorithm runs *weighted* FCM (WFCM) on sequential chunks of the data, passing the clustering solution from each chunk onto the next. SPFCM is truly scalable as its space complexity is only based on the size of the sample. A similar algorithm, OFCM [23], performs a similar process as SPFCM; however, rather than passing the clustering solution from one chunk to the next, OFCM clusters the centers from each chunk in one final run. Because of this final run, OFCM is not truly scalable and is not recommended for truly VL data. Another algorithm that is incremental in spirit is brFCM [13], which first bins the data and then clusters the bin centers. However, the efficiency and accuracy results of brFCM are very dependent on the binning strategy; brFCM has been shown to be very effective on image data, which can be binned very efficiently.

Although not technically an incremental algorithm, but more in the spirit of acceleration, the FFCM algorithm [39] applies FCM to larger and larger nested samples of the data set until there is little change in the solution. Another acceleration algorithm that is incremental in spirit is mrFCM [8], which combines the FFCM with a final literal run of FCM on the full data set. These algorithms are not scalable, however, as they both contain final runs on nearly full-size data set, with one last run on the full data set. iii) *Approximation* algorithms use numerical tricks to approximate the clustering solution using manageable size chunks of the data. Many of these algorithms utilize some sort of data transformation to achieve this goal. The algorithms described in [7, 30] fit this description.

None of these algorithms address *kernel* fuzzy clustering, which we describe next.

1.4 KFCM

Consider some non-linear mapping function $\phi : \mathbf{x} \rightarrow \phi(\mathbf{x}) \in \mathbb{R}^{D_K}$, where D_K is the dimensionality of the transformed feature vector $\mathbf{x}$. Most, if not all, kernel-based methods do not explicitly transform $\mathbf{x}$ and then operate in the higher-dimensional space of $\phi(\mathbf{x})$; instead, they use a kernel function κ that represents the inner product of the transformed feature vectors, $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_1) \rangle$. This kernel function can take may forms, with the polynomial, $\kappa(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^T \mathbf{x}_2 + 1)^p$, and *radial-basis-function* (RBF), $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \exp(\sigma ||\mathbf{x}_1 - \mathbf{x}_2||^2)$, being two of the most popular forms.

The KFCM algorithm can be generally defined as the constrained minimization of

$$J_m(U; \kappa) = \sum_{j=1}^{c} \left(\sum_{i=1}^{n} \sum_{k=1}^{n} \left(u_{ij}^m u_{kj}^m d_\kappa(\mathbf{x}_i, \mathbf{x}_k) \right) / 2 \sum_{l=1}^{n} u_{lj}^m \right), \tag{7}$$

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.

The KFCM/AO algorithm solves the optimization problem $\min\{J_m(U; \kappa\}$ by computing iterated updates of

$$u_{ij} = \left(\sum_{k=1}^{c} \left(\frac{d_\kappa(i, j)}{d_\kappa(i, k)} \right)^{\frac{1}{m-1}} \right)^{-1}, \ \forall i, j, \tag{8}$$

where, for simplicity, we denote the cluster center to object distance $d_\kappa(\mathbf{x}_i, \mathbf{v}_j)$ as $d_\kappa(i, j)$. This kernel distance is computed as

$$d_\kappa(i, j) = ||\phi(\mathbf{x}_i) - \phi(\mathbf{v}_i)||^2, \tag{9}$$

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

$$\phi(\mathbf{v}_j) = \frac{\sum_{l=1}^{n} u_{lj}^m \phi(\mathbf{x}_l)}{\sum_{l=1}^{n} u_{lj}^m}. \tag{10}$$

Equation (9) cannot by explicitly solved, but by using the identity $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$, denoting $\tilde{\mathbf{u}}_j = \mathbf{u}_j^m / ||\mathbf{u}_j^m||_1$ ($\mathbf{u}_j$ is the jth column of U), and substituting (10) into (9) we get

$$\begin{aligned} d_\kappa(j,i) &= \frac{\sum_{l=1}^{n} \sum_{s=1}^{n} u_{lj}^m u_{sj}^m \langle \phi(\mathbf{x}_l), \phi(\mathbf{x}_s) \rangle}{\sum_{l=1}^{n} u_{lj}^{2m}} \\ &\quad + \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_i) \rangle - 2 \frac{\sum_{l=1}^{n} u_{lj}^m \langle \phi(\mathbf{x}_l), \phi(\mathbf{x}_i) \rangle}{\sum_{l=1}^{n} u_{lj}^m} \\ &= \tilde{\mathbf{u}}_j^T K \tilde{\mathbf{u}}_j + \mathbf{e}_i^T K \mathbf{e}_i - 2 \tilde{\mathbf{u}}_j^T K \mathbf{e}_i \\ &= \tilde{\mathbf{u}}_j^T K \tilde{\mathbf{u}}_j + K_{ii} - 2(\tilde{\mathbf{u}}_j^T K)_i, \end{aligned} \tag{11}$$

where $\mathbf{e}_i$ is the n-length unit vector with the ith element equal to 1. Algorithm 2 outlines the KFCM/AO procedure.

Algorithm 2: KFCM/AO to minimize $J_m(U; \kappa)$

```
Input: $K$, $c$, $m$
Output: $U$
Initialize $U$
while $\max\{||U_{new} - U_{old}||^2\} > \epsilon$ do
    $d_\kappa(j,i) = \tilde{\mathbf{u}}_j^T K \tilde{\mathbf{u}}_j + K_{ii} - 2(\tilde{\mathbf{u}}_j^T K)_i \quad \forall i,j$
    $u_{ij} = \left( \sum_{k=1}^{c} \left( \frac{d_\kappa(i,j)}{d_\kappa(i,k)} \right)^{\frac{1}{m-1}} \right)^{-1} \quad \forall i,j$
```

This formulation of KFCM is equivalent to that proposed in [43] and, furthermore, is identical to *relational* FCM (RFCM) [19] if the kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle_A = \mathbf{x}_i^T A \mathbf{x}_j$ is used [20]. If one replaces the relational matrix R in RFCM with $R = -\gamma K$, for any $\gamma > 0$, then RFCM will produce the same partition as KFCM run on K (assuming the same initialization). Likewise, KFCM will produce the same partition as RFCM if $K = -\gamma R$, for any $\gamma > 0$.

1.5 Weighted KFCM

In the KFCM model, each object is considered equally important in the clustering solution. The *weighted* KFCM (wKFCM) model introduces weights that define the relative importance of each object in the clustering solution, similar to the wFCM in [13, 22, 23] and weighted kernel k-means in [10]. The wKFCM model is the constrained minimization of

$$J_{m\mathbf{w}}(U;\kappa) = \sum_{j=1}^{c}\left(\sum_{i=1}^{n}\sum_{k=1}^{n}\left(w_i w_k u_{ij}^m u_{kj}^m d_\kappa(\mathbf{x}_i,\mathbf{x}_k)\right)/2\sum_{l=1}^{n} w_l u_{lj}^m\right), \tag{12}$$

where $\mathbf{w} \in \mathbb{R}^n$, $w_i \geq 0\ \forall i$, is a set of weights, one element for each feature vector.

The cluster center $\phi(\mathbf{v}_j)$ is a weighted sum of the feature vectors, as shown in (10). Now assume that each object $\phi(\mathbf{x}_i)$ has a different predetermined influence, given by a respective weight w_i. This leads to the definition of the center as

$$\phi(\mathbf{v}_j) = \frac{\sum_{l=1}^{n} w_l u_{lj}^m \phi(\mathbf{x}_l)}{\sum_{l=1}^{n} w_l u_{lj}^m}. \tag{13}$$

Substituting (13) into (11) gives

$$\begin{aligned} d_\kappa^{\mathbf{w}}(i,j) &= \frac{\sum_{l=1}^{n}\sum_{r=1}^{n} w_l w_r u_{lj}^m u_{rj}^m \langle\phi(\mathbf{x}_l),\phi(\mathbf{x}_r)\rangle}{\sum_{l=1}^{n} w_l^2 u_{lj}^{2m}} \\ &\quad + \langle\phi(\mathbf{x}_i),\phi(\mathbf{x}_i)\rangle - 2\frac{\sum_{l=1}^{n} w_l u_{lj}^m \langle\phi(\mathbf{x}_l),\phi(\mathbf{x}_i)\rangle}{\sum_{l=1}^{n} w_l u_{lj}^m} \\ &= \frac{1}{||\mathbf{w}\circ\mathbf{u}_j||^2}(\mathbf{w}\circ\mathbf{u}_j)^T K(\mathbf{w}\circ\mathbf{u}_j) + K_{ii} \\ &\quad - \frac{2}{||\mathbf{w}\circ\mathbf{u}_j||}\left((\mathbf{w}\circ\mathbf{u}_j)^T K\right)_i, \end{aligned} \tag{14}$$

where $\mathbf{w}$ is the vector of predetermined weights and $\circ$ indicates the Hadamard product (.* in MATLAB). This leads to the *weighted* KFCM (wKFCM) shown in Algorithm 3. Notice that wKFCM also outputs the index of the nearest object to each cluster center, called the cluster prototype. The vector of indices P is important in the VL data schemes now proposed.

Algorithm 3: wKFCM/AO to minimize $J_{m\mathbf{w}}(U,\kappa)$

```
Input: $K$, $c$, $m$, $\mathbf{w}$
Output: $U$,$P$
Initialize $U \in M_{fcn_s}$
while $\max\{||U_{new} - U_{old}||^2\} > \epsilon$ do
    $d_\kappa^{\mathbf{w}}(i,j) = \frac{1}{||\mathbf{w}\circ\mathbf{u}_j||^2}(\mathbf{w}\circ\mathbf{u}_j)^T K(\mathbf{w}\circ\mathbf{u}_j) + K_{ii} - \frac{2}{||\mathbf{w}\circ\mathbf{u}_j||}\left((\mathbf{w}\circ\mathbf{u}_j)^T K\right)_i \quad \forall i,j$
    $u_{ij} = \left[\sum_{k=1}^{c}\left(\frac{d_\kappa^{\mathbf{w}}(i,j)}{d_\kappa^{\mathbf{w}}(i,k)}\right)^{\frac{1}{m-1}}\right]^{-1} \quad \forall i,j$
$p_j = \arg\min_i\{d_\kappa(i,j)\},\ \forall j$
```

2 Incremental Algorithms

The problem with KFCM and wKFCM is that they require working memory to store the full $n \times n$ kernel matrix K. For large n, the memory requirement can be very significant; e.g., with $n = 1,000,000$, 4 TB of working memory are required. This is infeasible for even most high-performance computers. Hence, even Huber's "medium" data sets (on the order of 10^6) are impossible to cluster on moderately powerful computers. For this reason, kernel clustering of large-scale data is infeasible without some method that scales well.

2.1 rseKFCM

The most basic, and perhaps obvious, way to address kernel fuzzy clustering in VL data is to sample the dataset and then use KFCM to compute partitions of the sampled data. This is similar to the approach of geFFCM (geFFCM uses literal FCM instead of KFCM); however, geFFCM uses a progressive sampling[3] approach to draw a sample that is representative (enough) of the full data set. However, for VL data, this representative sample may be large itself. Thus, we use randomly draw without replacement a predetermined sub-sample of X. We believe that random sampling is sufficient for VL data and is, of course, computationally less expensive than a progressive approach. There are other sampling schemes addressed in [1, 11, 32]; these papers also support our claim that uniform random sampling is preferred. Another issue with directly applying the sample and extend approach of geFFCM is that it first computes cluster centers V and then uses (4) to extend the partition to the remaining data. In constrast, KFCM does not return a cluster center (per se); hence, one cannot directly use (4) to extend the partition to the remaining data. However, recall that wKFCM, in Algorithm 3, returns a set of cluster prototypes P, which are the indices of the c objects that are closest to the cluster centers (in the RKHS). Thus follows our rseKFCM algorithm, outlined in Algorithm 4.

```
Algorithm 4: rseKFCM to approximately minimize $J_m(U;\kappa)$
  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 $U, P = \text{wKFCM}(K, c, m, \mathbf{1}_{n_s})$
4 Extend partition to $X$:
    $d_\kappa(j,i) = \kappa(\mathbf{x}_i,\mathbf{x}_i) + \kappa(\mathbf{x}_{P_j},\mathbf{x}_{P_j}) - 2\kappa(\mathbf{x}_i,\mathbf{x}_{P_j}) \quad \forall i,j$
    $u_{ij} = \left[\sum_{k=1}^{c}\left(\frac{d_\kappa(i,j)}{d_\kappa(i,k)}\right)^{\frac{1}{m-1}}\right]^{-1} \quad \forall i,j$
```

[3] **(author?)** [37] provide a very readable analysis and summary of progressive sampling schemes.

First, a random sample of X is drawn at Line 1. Then the kernel matrix K is computed for this random sample at Line 2, and at Line 3 wKFCM is used to produce a set of cluster prototypes. Finally, the partition is extended to the remaining data at Line 4.

The rseKFCM algorithm is scalable as one can choose a sample size n_s to suit their computational resources. However, the clustering iterations are not performed on the entire data set (in literal or in chunks); hence, if the sample X_s is not representative of the full data set, then rseKFCM will not accurately approximate the literal KFCM solution.

This leads to the discussion of two algorithms that operate on the full data set by separating it into multiple chunks.

2.2 spKFCM

The spKFCM algorithm is based upon the spFCM algorithm proposed in [22]. Essentially, spKFCM (and spFCM) splits the data into multiple (approximately) equally size chunks, then clusters each chunk separately. The clustering result from each chunk is passed to the next chunk in order to approximate the literal KFCM solution on the full data set. In SPFCM, the cluster center locations from each chunk are passed to the next chunk as data points to be included in the data set that is clustered. However, these cluster centers are weighted in the WFCM by the sum of the respective memberships, i.e. the cth cluster is weighted by the sum of the cth row of U. Essentially, the weight causes the cluster centers to have more influence on the clustering solution that the data in the data chunk. In other words, each cluster center represents the multiple data points in each cluster.

Because there are no cluster centers in KFCM (or wKFCM), the data that is passed on to the next chunk are, instead, the cluster prototypes—the objects nearest to the cluster center in the RKHS. Hence, the kernel matrix for each data chunk is the $(n_s + c) \times (n_s + c)$ kernel function results—n_s columns (or rows) for the objects in the data chunk and c columns for the c prototypes passed on from the previous chunk.

Algorithm 5 outlines the spKFCM algorithm. At Line 1, the data X is randomly separated into s equally-sized subsets, where the indices of the objects in each subset of denoted ξ_i, $i = 1, \ldots, s$. Lines 2-4 comprise the operations on the first data chunk. At Line 2, the $n_s \times n_s$ kernel matrix of the first data chunk is computed and, at Line 3, wKFCM is used to cluster the first data chunk. The weights of the c cluster prototypes returned by wKFCM are computed at Line 4. Lines 5-9 are the main loop of spKFCM. For each data chunk, Line 5 creates a vector of weights, where the weight for the c prototypes is calculated from the previous data chunk results and the weights of the n_s objects are set to 1. At Line 6, the indices of the objects in the lth data chunk and the c cluster prototypes are combined and, at Line 7, the $(n_s + c) \times (n_s + c)$ kernel matrix is computed (for the objects indexed by ξ_l and the c cluster prototypes). wKFCM is then used to produce the clustering results at Line 8. And, at Line 9, the weights are calculated, which are then used in the next data chunk loop. Finally, at Line 10, the indices of the c cluster prototypes are returned.

spKFCM is a scalable algorithm because one can choose the size of the data chunk to be clustered and the maximum size of the data to be clustered is $n_s + c$. The storage requirements for the kernel matrices is thus $O((n_s + c)^2)$.

Algorithm 5: spKFCM to approximately minimize $J_m(U;\kappa)$

```
Input: Kernel function $\kappa$, $X$, $c$, $m$, $s$
Output: $P$
1  Randomly draw $s$ (approximately) equal-sized subsets of the integers $\{1,\dots,n\}$, denoted
   $\Xi = \{\xi_1,\dots,\xi_s\}$. $n_l$ is the cardinality of $\xi_l$.
2  $K = [\kappa(\mathbf{x}_i,\mathbf{x}_j)] \quad i,j = \xi_1$
3  $U, P$ = wKFCM$(K, c, m, \mathbf{1}_{n_1})$
4  $w'_j = \sum_{i=1}^{n_1} u_{ij} \quad \forall j$
   for $l = 2$ to $s$ do
5      $\mathbf{w} = \{\mathbf{w}', \mathbf{1}_{n_l}\}$
6      $\xi' = \{\xi'(P), \xi_l\}$
7      $K = [\kappa(\mathbf{x}_i,\mathbf{x}_j)] \quad i,j = \xi'$
8      $U, P$ = wKFCM$(K, c, m, \mathbf{w})$
9      $w'_j = \sum_{i=1}^{n_l+c} u_{ij} \quad \forall j$
10 $P = \xi'(P)$
```

2.3 oKFCM

The oKFCM algorithm is similar to spKFCM, and is based on the oFCM algorithm proposed in [23]. Algorithm 6 outlines the oKFCM procedure. Like spKFCM, oKFCM starts by separating the objects into s equally-sized data chunks. However, unlike spKFCM, it does not pass the cluster prototypes from the previous data chunk onto the next data chunk. oKFCM simply calculates s sets of c cluster prototypes, one set from each data chunk. It then computes a weight for each of the cs cluster prototypes, which is the sum of the row of the respective membership matrix (there is one membership matrix computed for each of the s data chunks). FInally, the cs cluster prototypes are partitioned using wKFCM, producing a final set of c cluster prototypes.

Because there is no interaction between the initial clustering done on each data chunk, oKFCM could be easily implemented on a distributed architecture. Each iteration of Lines 4-6 is independent and could thus be simultaneously computed on separate nodes of a parallel architecture (or cloud, if you will). However, the final clustering of the cs cluster prototypes prevents oKFCM from being a truly scalable algorithm. If s is large, then this final data set is large. In extreme cases, if $s >> n_s$ then the storage requirement for oKFCM becomes $O((cs)^2)$.

3 Experiments

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

For all algorithms, we initialize U by randomly choosing c objects as the initial cluster centers. The value of $\epsilon = 10^{-3}$ and the fuzzifier $m = 1.7$. The termination

Algorithm 6: oKFCM to approximately minimize $J_m(U;\kappa)$

```
    Input: Kernel function $\kappa$, $X$, $c$, $m$, $s$
    Output: $U$,$P$
 1  Randomly draw $s$ (approximately) equal-sized subsets of the integers $\{1,\ldots,n\}$, denoted
    $\Xi = \{\xi_1,\ldots,\xi_s\}$. $n_l$ is the cardinality of $\xi_l$.
 2  $K = [\kappa(\mathbf{x}_i,\mathbf{x}_j)] \quad i,j = \xi_1$
 3  $U_1, P_1$ = wKFCM$(K,c,m,\mathbf{1}_{n_1})$
    for $l = 2$ to $s$ do
 4      $K = [\kappa(\mathbf{x}_i,\mathbf{x}_j)] \quad i,j = \mathbf{x}'_l$
 5      $U_l, P'$ = wKFCM$(K,c,m,\mathbf{1}_{n_l})$
 6      $P_l = \xi_l(P')$
 7  $P_{all} = \{P_1,\ldots,P_s\}$
 8  $K = [\kappa(\mathbf{x}_i,\mathbf{x}_j)] \quad i,j = P_{all}$
 9  $\mathbf{w}_l = \sum_{j=1}^{n_s}(U_l)_{\cdot j} \quad \forall l$
10  $U, P'$ = wKFCM$(K,c,m,\mathbf{w})$
11  $P = P_{all}(P')$
```

criteria is $\max\{||U_{new} - U_{old}||^2\} < \epsilon$. The experiments were performed on a single core of an AMD Opteron in a Sun Fire X4600 M2 server with 256 gigabytes of memory. All code was written in the MATLAB computing environment.

3.1 Evaluation Criteria

We judge the performance of the incremental KFCM algorithms using three criteria. Each criteria is computed for 21 independent runs with random initializations and samplings. The results presented are the average values.

1. **Speedup Factor or Run-Time** This criteria represents an actual run-time comparison. When the KFCM solution is available, speedup is defined as $t_{literal}/t_{incremental}$, where these values are times in seconds for computing the the membership matrix U. When the data is too large to compute KFCM solutions, we present run-time in seconds for the incremental algorithms.
2. **Adjusted Rand Index** The Rand index [38] is a measure of agreement between two crisp partitions of a set of objects. One of the two partitions is usually a 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*, ARI(U,U'), is a bias-adjusted formulation developed by Hubert and Arabie [25]. To compute the ARI, we first harden the fuzzy partitions by setting the maximum element in each row of U to 1, and all else to 0. We use the ARI to compare the clustering solutions to ground-truth labels (when available), and also to compare the VL data algorithms to the literal FCM solutions.

Note that the rseKFCM, spKFCM, and oKFCM algorithms do not produce full data partitions; they produce cluster prototypes as output. Hence, we cannot directly compute ARI and fuzzy ARI for these algorithms. To complete the calculations, we used the Extension step to produce full data partitions from the output cluster prototypes. The Extension step was *not* included in the speedup factor or run-time calculations for these algorithms as these algorithms were designed to return cluster prototypes (as the analogous rseFCM, SPFCM, and OFCM), not full data partitions. However, we observed in our experiments that the Extension step added a nearly negligible amount of time to the overall run-time of the algorithms.

The data we used in this study are:

1. **2D50** $(n = 7,500, c = 50, d = 2)$: These data are composed of 7,500 2-dimensional vectors, with a visually-preferred grouping into 50 clusters.[4] Figure 1 shows a plot of these data. An RBF kernel with $\sigma = 1$, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\sigma||\mathbf{x}_i - \mathbf{x}_j||^2\right)$, was used.
2. **MNIST** $(n = 70,000, c = 10, d = 784)$: This data set is a subset of the collection of handwritten digits available from the *National Institute of Standards and Technology* (NIST)[5]. There are 70,000 28×28 pixel images of the digits 0 to 9. Each 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 column vector. A 5-degree inhomogeneous polynomial kernel was used, which was shown to be (somewhat) effective in [9, 21, 45].

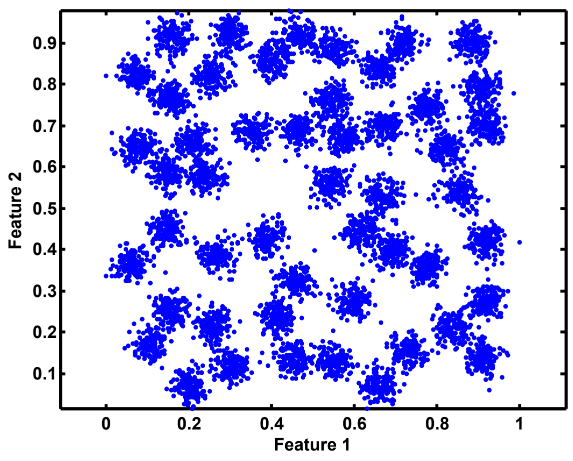


Fig. 1: 2D50 synthetic data; $n = 7500$ objects, $c = 50$ clusters

Figure 2 shows the results of the incremental algorithms on the 2D50 data set. The speedup factor, shown in view (a), demonstrates that rseKFCM is the fastest algorithm overall, with a speedup of about 450 at a 1% sample rate. However, at sample

[4] The 2D50 data were designed by Ilja Sidoroff and can be downloaded at `http://cs.joensuu.fi/~isido/clustering/`.

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

rates $> 5\%$, rseKFCM and spKFCM exhibit nearly equal speedup results. As view (b) shows, at sample rates $> 5\%$, all three algorithms perform comparably. The rseKFCM algorithm shows slightly better results than oKFCM at sample rates $> 5\%$ and the spKFCM algorithm exhibits inconsistent performance, sometimes performing better than the other algorithms, sometimes worse; although, all three algorithms show about the same performance. The oKFCM shows the best performance at very low sample rates ($< 5\%$) but the oKFCM algorithm is also the least efficient of the three.

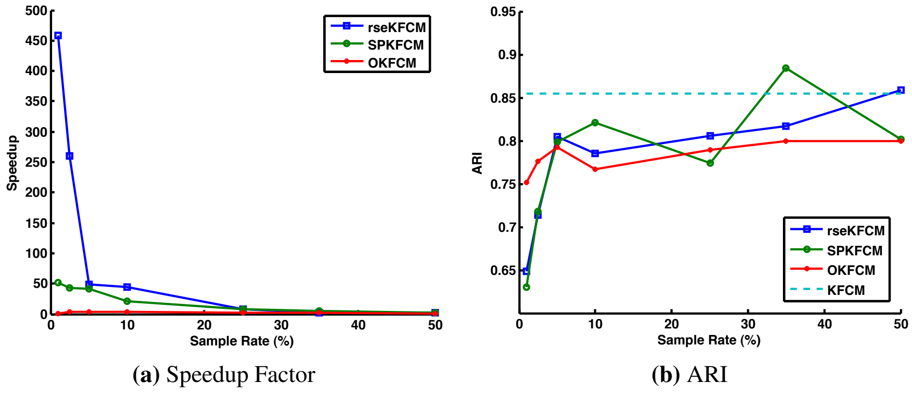
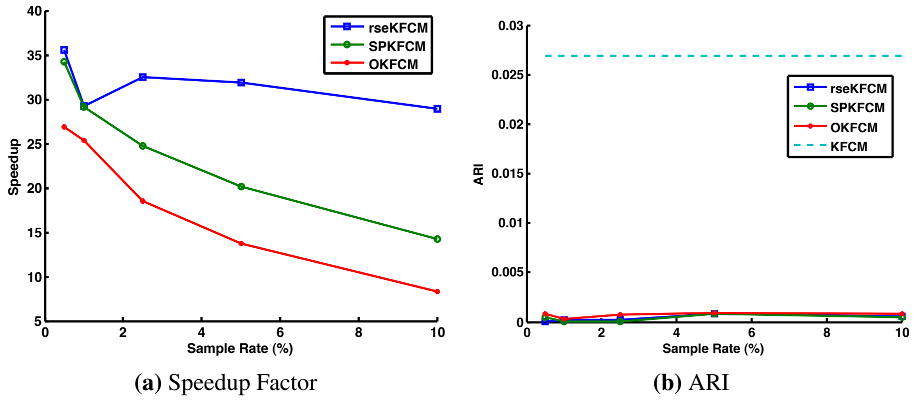


Fig. 2: Performance of incremental KFCM algorithms on 2D50 data set. ARI is calculated relative to ground-truth labels.

Figure 3 shows the performance of the incremental KFCM algorithms on the MNIST data set. These results tell a somewhat different story than the previous data set. First, rseKFCM is no longer the clear winner in terms of speedup. At low sample rates, the spKFCM and rseKFCM perform comparably. This is because rseKFCM suffered from slow convergence at the low sample rates. At sample rates $> 2\%$, rseKFCM is clearly the fastest algorithm. Second, oKFCM is comparable to the other algorithms in terms of speedup. However, the ARI results show a dismal view of the performance of these algorithms for this data set (in terms of accuracy compared to ground truth labels). All three algorithms fail to perform nearly as well as the literal KFCM algorithm (shown by the dotted line). Note that even the literal KFCM performs rather poorly on this data set (in terms of its accuracy with respect to comparison with the ground truth labels). This suggests that the incremental KFCM algorithms have trouble with data sets that are difficult to cluster. Perhaps, the cluster structure is lost when the kernel matrix is sampled.

4 Discussion and Conclusions

We present here three adaptations of an incremental FCM algorithm to kernel FCM. In a nutshell, the rseKFCM algorithm seems to be the preferred algorithm. It is the most scalable and efficient solution, and produces results that are on par with those of



(a) Speedup Factor **(b)** ARI

Fig. 3: Performance of incremental KFCM algorithms on MNIST data set. ARI is calculated relative to ground-truth labels.

spKFCM and oKFCM. The oKFCM does not suffer in performance at low sample rates, but is also the most inefficient of the three. Hence, we recommend using rseKFCM at the highest sample rate possible for your computational resources. We believe that this approach will yield the most desirable results, in terms of both in speedup and accuracy.

Although the incremental KFCM algorithms performed well on the synthetic 2D50 data set, their performance suffered, relative to literal KFCM, on the MNIST data set, which was not as “easily” clustered. To combat this issue, we are going to examine other ways by which the KFCM solution can be adapted for incremental operation. One method we are currently examining is a way by which a more meaningful cluster prototype can be produced by wKFCM. Furthermore, we plan to look at ways that the sample size can be increased without sacrificing speedup and scalability, such as in the approximate kernel k-means approach proposed in [9, 21].

Another question that arises in incremental clustering 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 several well-known cluster validity measures for incremental use by using similar strategies to the adaptations presented here.

In closing, we would like 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 is 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 how well the incremental KFCM 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.

Acknowledgements

Timothy Havens is supported by the National Science Foundation under Grant #1019343 to the Computing Research Association for the CI Fellows Project.

We wish to acknowledge the support of the Michigan State University High Performance Computing Center and the Institute for Cyber Enabled Research.

Bibliography

[1] Belabbas, M., Wolfe, P.: Spectral methods in machine learning and new strategies for very large datasets. Proc. National Academy of Sciences 106(2), 369–374 (2009)

[2] Bezdek, J.: A convergence theorem for the fuzzy isodata clustering algorithms. IEEE Trans. Pattern Analysis and Machine Intelligence 2, 1–8 (1980)

[3] Bezdek, J.: Pattern Recognition With Fuzzy Objective Function Algorithms. Plenum, New York (1981)

[4] Bezdek, J., Hathaway, R.: Convergence of alternating optmization. Nueral, Parallel, and Scientific Computations 11(4), 351–368 (Dec 2003)

[5] Bezdek, J., Keller, J., Krishnapuram, R., Pal, N.: Fuzzy Models and Algorithms for Pattern Recognition and Image Processing. Kluwer, Norwell (1999)

[6] Bo, W., Nevatia, R.: Cluster boosted tree classifier for multi-view, multi-pose object detection. In: Proc. ICCV (October 2007)

[7] Cannon, R., Dave, J., Bezdek, J.: Efficient implementation of the fuzzy c-means algorithm. IEEE Trans. Pattern Analysis and Machine Intelligence 8, 248–255 (1986)

[8] Cheng, T., Goldgof, D., Hall, L.: Fast clustering with application to fuzzy rule generation. In: Proc. IEEE Int. Conf. Fuzzy Systems. pp. 2289–2295. Tokyo, Japan (1995)

[9] Chitta, R., Jin, R., Havens, T., Jain, A.: Approximate kernel k-means: Solution to large scale kernel clustering. In: Proc. ACM SIGKDD (2011)

[10] Dhillon, I., Guan, Y., Kulis, B.: Kernel k-means, spectral clustering, and normalized cuts. In: Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery Data Mining. pp. 551–556 (August 2004)

[11] Drineas, P., Mahoney, M.: On the nystrom method for appoximating a gram matrix for improved kernel-based learning. The J. of Machine Learning Research 6, 2153–2175 (2005)

[12] Duda, R., Hart, P., Stork, D.: Pattern Classification. Wiley-Interscience, second edn. (October 2000)

[13] Eschrich, S., Ke, J., Hall, L., Goldgof, D.: Fast accurate fuzzy clustering through data reduction. IEEE Trans. Fuzzy Systems 11, 262–269 (2003)

[14] Frigui, H.: Advances in Fuzzy Clustering and Feature Discrimination with Applications, chap. Simultaneous Clustering and Feature Discrimination with Applications, pp. 285–312. John Wiley and Sons (2007)

[15] Hartigan, J.: Clustering Algorithms. Wiley, New York (1975)

[16] Hathaway, R., Bezdek, J.: NERF c-MEANS: Non-euclidean relational fuzzy clustering. Pattern Recognition 27, 429–437 (1994)

[17] Hathaway, R., Bezdek, J.: Extending fuzzy and probabilistic clustering to very large data sets. Computational Statistics and Data Analysis 51, 215–234 (2006)

[18] Hathaway, R., Bezdek, J., Tucker, W.: An improved convergence theory for the fuzzy isodata clustering algorithms. In: Bezdek, J. (ed.) Analysis of Fuzzy Information, vol. 3, pp. 123–132. CRC Press, Boca Raton (1987)

[19] Hathaway, R., Davenport, J., Bezdek, J.: Relational duals of the c-means clustering algorithms. Pattern Recognition 22(2), 205–212 (1989)

[20] Hathaway, R., Huband, J., Bezdek, J.: A kernelized non-euclidean relational fuzzy c-means algorithm. In: Proc. IEEE Int. Conf. Fuzzy Systems. pp. 414–419 (2005)

[21] Havens, T., Chitta, R., Jain, A., Jin, R.: Speedup of fuzzy and possibilistic c-means for large-scale clustering. In: Proc. IEEE Int. Conf. Fuzzy Systems. Taipei, Taiwan (2011)

[22] Hore, P., Hall, L., Goldgof, D.: Single pass fuzzy c means. In: Proc. IEEE Int. Conf. Fuzzy Systems. pp. 1–7. London, England (2007)

[23] Hore, P., Hall, L., Goldgof, D., Gu, Y., Maudsley, A.: A scalable framework for segmenting magentic resonance images. J. Signal Process. Syst. 54(1-3), 183–203 (January 2009)

[24] Huber, P.: Massive Data Sets, chap. Massive Data Sets Workshop: The Morning After, pp. 169–184. National Academy Press (1997)

[25] Hubert, L., Arabie, P.: Comparing partitions. J. Classification 2, 193–218 (1985)

[26] Jain, A., Dubes, R.: Algorithms for Clustering Data. Prentice-Hall, Englewood Cliffs, NJ (1988)

[27] Jain, A., Murty, M., Flynn, P.: Data clustering: A review. ACM Computing Surveys 31(3), 264–323 (September 1999)

[28] Johnson, S.: Hierarchical clustering schemes. Psychometrika 2, 241–254 (1967)

[29] Khan, S., Situ, G., Decker, K., Schmidt, C.: GoFigure: Automated Gene Ontology annotation. Bioinf. 19(18), 2484–2485 (2003)

[30] Kolen, J., Hutcheson, T.: Reducing the time complexity of the fuzzy c-means algorithm. IEEE Trans. Fuzzy Systems 10, 263–267 (2002)

[31] Krishnapuram, R., Keller, J.: A possibilistic approach to clustering. IEEE Trans. on Fuzzy Sys. 1(2) (May 1993)

[32] Kumar, S., Mohri, M., Talwalkar, A.: Sampling techniques for the nystrom method. In: Proc. Conf. Artificial Intelligence and Statistics. pp. 304–311 (2009)

[33] Lloyd, S.: Least square quantization in pcm. Tech. rep., Bell Telephone Laboratories (1957)

[34] Lloyd, S.: Least square quantization in pcm. IEEE Trans. Information Theory 28(2), 129–137 (1982)

[35] MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proc. 5th Berkeley Symp. Math. Stat. and Prob. pp. 281–297. University of California Press (1967)

[36] Pal, N., Bezdek, J.: Complexity reduction for “large image” processing. IEEE Trans. Systems, Man, and Cybernetics B(32), 598–611 (2002)

[37] Provost, F., Jensen, D., Oates, T.: Efficient progressive sampling. In: Proc. KDDM. pp. 23–32 (1999)

[38] Rand, W.: Objective criteria for the evaluation of clustering methods. J. Amer. Stat. Asooc. 66(336), 846–850 (1971)

[39] Shankar, B.U., Pal, N.: FFCM: an effective approach for large data sets. In: Proc. Int. Conf. Fuzzy Logic, Neural Nets, and Soft Computing. p. 332. Fukuoka, Japan (1994)

[40] The UniProt Consotium: The universal protein resource (UniProt). Nucleic Acids Res. 35, D193–D197 (2007)

[41] Theodoridis, S., Koutroumbas, K.: Pattern Recognition. Academic Press, San Diego, CA, 4 edn. (2009)

[42] Tucker, W.: Counterexamples to the convergence theorem for fuzzy isodata clustering algorithms. In: Bezdek, J. (ed.) Analysis of Fuzzy Information, vol. 3, pp. 109–122. CRC Press, Boca Raton (1987)

[43] Wu, Z., Xie, W., Yu, J.: Fuzzy c-means clustering algorithm based on kernel method. In: Proc. Int. Conf. Computational Intelligence and Multimedia Applications. pp. 49–54 (September 2003)

[44] Xu, R., Wunsch II, D.: Clustering. IEEE Press, Psicataway, NJ (2009)

[45] Zhang, R., Rudnicky, A.: A large scale clustering scheme for kernel k-means. In: Proc. Int. Conf. Pattern Recognition. pp. 289–292 (2002)