Speedup of Fuzzy and Possibilistic Kernel c-Means for Large-Scale Clustering

Timothy C. Havens, Radha Chitta, Anil K. Jain, and Rong Jin
Department of Computer Science and Engineering
Michigan State University, East Lansing, MI 48824
E-mail: havenst@gmail.com, {chittara, jain, rongjin}@msu.edu

***Abstract*—The ubiquity of personal computing technology has produced an abundance of staggeringly large data sets—the Library of Congress has stored over 160 terabytes of web data and it is estimated that Facebook alone logs over 25 terabytes of data per day. There is a great need for systems by which one can elucidate the similarity and dissimilarity among and between groups in these data sets. Clustering is one way to find these groups. In this paper, we propose an approximation method for the fuzzy and possibilistic kernel c-means clustering algorithms. Our approximation constrains the cluster centers to be linear combinations of a size m randomly selected subset of the n input objects, where $m << n$. The proposed algorithm only requires an $m \times n$ rectangular portion of the full $n \times n$ kernel matrix and the n diagonal values, resulting in significant memory savings. Furthermore, the computational complexity of the c-means algorithm is substantially reduced. We demonstrate that up to 3 orders of magnitude of speedup are possible while achieving almost the same performance as the original kernel c-means algorithm.**

I. Introduction

Clustering algorithms are an integral part of both computational intelligence and pattern recognition. Often researchers are mired in data sets that are large and unlabeled. There are many methods, under the heading exploratory data analysis, by which researchers can elucidate these data. Clustering is one such exploratory tool for deducing the nature of the data by providing labels to individual objects that describe how the data separate into groups. 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 [1, 2]. Also, clustering has 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) [3, 4].

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 a measure of similarity—is a basic part of intelligence and is ubiquitous to the scientific endeavor.

Consider a set of n objects, denoted $O = \{o_1, \ldots, o_n\}$, e.g., sets of vintage bass guitars, freshwater fish, or genes in a microarray experiment. Each object is typically represented by an associated set of vectors $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathbb{R}^d$, where each $\mathbf{x}$ is a numerical feature vector that describes the objects' attributes — examples include height, weight, length, or expression value.

Clustering in unlabeled data X is defined as the assignment of *labels* to groups of similar (unlabeled) objects O. In other words, objects are *partitioned* into groups such that each group is composed of objects with similar attributes. 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—each object is a cluster. A wide array of algorithms exists for clustering unlabeled object data O. Descriptions of many of these algorithms, both relational and not, can be found in the following general references on clustering: [5–12].

A *partition* of the objects is defined as the set of cn values, where each value $\{u_{ik}\}$ represents the degree to which an object o_k is in the ith cluster. The c-partition is often represented as a $c \times n$ matrix $U = [u_{ik}]$, where each row represents a cluster and each column represents an object. There are three main types of partitions, crisp, fuzzy (or probabilistic), and possibilistic [7, 13]. *Crisp* partitions of the unlabeled objects are non-empty mutually-disjoint subsets of O such that the union of the subsets covers O. The set of all non-degenerate (no zero rows) crisp c-partition matrices for the object set O is described in (1), where u_{ik} is the *membership* of object o_k in cluster i; the partition element $u_{ik} = 1$ if o_k is labeled i 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 a probability $p(i|o_k)$ that o_k is in the i-th class. We assume, in this paper, 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 described in (2). Each column of the fuzzy partition U must sum to 1, thus ensuring that every object is involved in a partition ($\sum_i u_{ik} = 1$).

Possibilistic partitions relax this condition, allowing partition columns that do not necessarily sum to 1. Possibilistic clustering has been shown to be especially effective in partitioning data that has outliers and intersecting clusters [13]. The

$$M_{hcn} = \left\{ U \in \mathbb{R}_{0,+}^{c\times n} | u_{ij} \in 0, 1, \forall j, i; 0 < \sum_{i=1}^{n} u_{ij} < n, \forall j; \sum_{j=1}^{c} u_{ij} = 1, \forall i \right\} \quad (1)$$

$$M_{fcn} = \left\{ U \in \mathbb{R}_{0,+}^{c\times n} | u_{ij} \in [0, 1], \forall j, i; 0 < \sum_{i=1}^{n} u_{ij} < n, \forall j; \sum_{j=1}^{c} u_{ij} = 1, \forall i \right\} \quad (2)$$

$$M_{pcn} = \left\{ U \in \mathbb{R}_{0,+}^{c\times n} | u_{ij} \in [0, 1], \forall j, i; 0 < \sum_{i=1}^{n} u_{ij} < n, \forall j; \max_{1\le j\le c} u_{ij} > 0, \forall i \right\} \quad (3)$$

$$\min_{U} \left\{ J_m(U; H, K) = \sum_{j=1}^{c} \left(\sum_{i=1}^{n} \sum_{k=1}^{n} \left(u_{ij}^m u_{kj}^m d_\kappa(\mathbf{x}_i, \mathbf{x}_k) \right) / 2 \sum_{l=1}^{n} u_{lj}^m \right) + \sum_{j=1}^{c} \nu_j \sum_{i=1}^{n} (1 - u_{ij})^m \right\} \quad (4)$$

set of all possibilistic c-partitions is described in (3), where u_{ik} is the possibility that o_k is in cluster i (this was also described in [13] as the *typicality* of o_k to cluster i). Notice that the possibilistic partition ensures that there is at least one object that has a non-zero possibility of being in each cluster (the empty cluster cannot exist). An interesting property of these three types of partitions is that all crisp partitions are fuzzy partitions and all fuzzy partitions are possibilistic partitions. Equations (1), (2), and (3), show that $M_{hcn} \subset M_{fcn} \subset M_{pcn}$.

All cluster analyses address the same questions, independent of the type of partition. The three main questions are: i) *Cluster tendency*—are there clusters, and how many clusters are there? ii) *Partitioning*—which objects belong to which cluster and to what degree? And iii) *cluster validity*—are the partitions "good"? There are many algorithms that attempt to answer these questions. This paper focuses on a specific form of question ii), namely soft c-means partitioning of large datasets in kernel-induced spaces.

A. *Kernel c-Means*

Consider some non-linear mapping function $\phi : \mathbf{x} \to \phi(\mathbf{x}) \in \mathbb{R}^{D_K}$, 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 $\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + 1)^p$ and radial-basis-function (RBF) $\kappa(\mathbf{x}, \mathbf{y}) = \exp(||\mathbf{x} - \mathbf{y}||^2/\sigma)$ being two of the most well known. Given a set of n objects 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 associated with n objects in the transformed high-dimensional space—called the Reproducing Kernel Hilbert Space (RKHS).

Given a kernel matrix K, the kernel c-means can be generally defined as the constrained optimization in (4), where $U \in M_{pcn}$, $H = \{\nu_1, \ldots, \nu_c\} \in \mathbb{R}_{0,+}$ is the cluster radii set (which only applies to possibilistic clustering), $m \ge 1$ is the fuzzification parameter, and $d_\kappa(\mathbf{x}_i, \mathbf{x}_k)$ is the kernel-based distance between the ith and kth objects. All three of the major c-means algorithms—hard, fuzzy, and possibilistic—can be instantiated by adjusting the parameters appropriately in the objective function in (4). Table I shows how the parameters of (4) are set for each algorithm. For simplicity, we denote the cluster center to object distance $d_\kappa(\mathbf{v}_j, \mathbf{x}_i)$ as $d_\kappa(j, i)$.

TABLE I: Hard, Fuzzy, and Possibilistic Variations of Solving the c-Means Optimization in Eq. (4) [14]

Algorithm	m	H	U
Hard (Crisp)	$m = 1$	$\nu_j = 0, \forall j$	Use Eq. (5)
Fuzzy	$m > 1$	$\nu_j = 0, \forall j$	Use Eq. (6)
Possibilistic	$m > 1$	$\nu_j > 0, \forall j$	Use Eq. (7)

Hard c-means membership update [15]:

$$u_{ij} = \left\{ \begin{array}{ll} 1, & d_\kappa(j, i) = \min_{1\le k\le c} d_\kappa(k, i) \\ 0, & \text{else} \end{array} \right. , \forall j, i \quad (5)$$

Fuzzy c-means membership update [15]:

$$u_{ij} = \left(\sum_{k=1}^{c} \left(\frac{d_\kappa(j, i)}{d_\kappa(k, i)} \right)^{\frac{2}{m-1}} \right)^{-1}, \forall i, j \quad (6)$$

Possibilistic c-means membership update [15]:

$$u_{ij} = \left(1 + \left(\frac{d_\kappa(j, i)}{\nu_j} \right)^{\frac{2}{m-1}} \right)^{-1}, \forall i, j \quad (7)$$

The partition update equations in Table I are all based on a kernel distance between an object and a cluster center, which is computed as

$$d_\kappa(j, i) = ||\phi(\mathbf{v}_j) - \phi(\mathbf{x}_i)||^2, \quad (8)$$

where

$$\phi(\mathbf{v}_j) = \frac{\sum_{l=1}^{n} u_{lj}^m \phi(\mathbf{x}_l)}{\sum_{l=1}^{n} u_{lj}^m}. \quad (9)$$

We can simplify (8) by using the identity $K_{ij} = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$ and denoting $\tilde{\mathbf{u}}_j = \mathbf{u}_j^m / ||\mathbf{u}_j^m||_1$ ($\mathbf{u}_j$ is the jth column of U), which gives

$$\begin{aligned} d_\kappa(j,i) &= \frac{\sum_{l=1}^n \sum_{s=1}^n u_{lj}^m u_{sj}^m \phi(\mathbf{x}_l)\cdot\phi(\mathbf{x}_s)}{\sum_{l=1}^n u_{lj}^{2m}} \\ &\quad + \phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_i) - 2\frac{\sum_{l=1}^n u_{lj}^m \phi(\mathbf{x}_l)\cdot\phi(\mathbf{x}_i)}{\sum_{l=1}^n u_{lj}^m} \\ &= \tilde{\mathbf{u}}_j^T K \tilde{\mathbf{u}}_j + \mathbf{e}_i^T K \mathbf{e}_i - 2\tilde{\mathbf{u}}_j^T K \mathbf{e}_i \\ &= \tilde{\mathbf{u}}_j^T K \tilde{\mathbf{u}}_j + K_{ii} - 2(\tilde{\mathbf{u}}_j^T K)_i, \end{aligned} \tag{10}$$

where $\mathbf{e}_i$ is a length n unit vector with the ith element equal to 1.

In *possibilistic c-means* (PCM), the parameters ν determine the influence of each cluster on the value of the objective function in (4), where the second term in (4) penalizes the trivial solution of $u_{ij} = 0$, $\forall i, j$.

Algorithms 1, 2, and 3 outline the kernel-based variants of *hard c-means* (HCM), *fuzzy c-means* (FCM), and PCM, respectively. This variant of the PCM algorithm first initializes the memberships by using FCM, then the ν parameter is estimated by

$$\nu_j = \Theta\frac{\sum_{i=1}^n u_{ij}^m d_\kappa(j,i)}{\sum_{i=1}^n u_{ij}^m},\ \forall j, \tag{11}$$

where $\Theta > 0$ and is usually set to $\Theta = 1$ [8, 13].

Algorithm 1: Hard c-Means (HCM) Kernel Clustering Algorithm

```
Input: $K$ - $n \times n$ kernel matrix
Data: $m = 1$, $U \in \{0,1\}^{c\times n}$ cluster membership matrix
Initialize $U$
while $\max\{|U - U'|\} > 0$ do
    $U' = U$
    Compute $d_\kappa(j,i)$, $\forall i, j$, using Eq.(10)
    Update $U$ using Eq. (5)
```

Algorithm 2: Fuzzy c-Means (FCM) Kernel Clustering Algorithm

```
Input: $K$ - $n \times n$ kernel matrix; $m$ - fuzzifier
Data: $U \in [0,1]^{c\times n}$ cluster membership matrix
Initialize $U$
while $\max\{|U - U'|\} > \epsilon$ do
    $U' = U$
    Compute $d_\kappa(j,i)$, $\forall i, j$, using Eq.(10)
    Update $U$ using Eq. (6)
```

In all these c-means variants, Eq.(10) is computationally expensive for large number of objects n as it requires $O(n^2)$ operations. Furthermore, the entire kernel matrix K is required and producing and storing this matrix for the entire data set is often very computationally expensive; e.g., $n = 100,000$ requires 40 gigabytes of memory, which is not a small amount by today's standards. We approximate $d_\kappa(j,i)$ by constraining the space in which the cluster centers exist.

Algorithm 3: Possibilistic c-Means (PCM) Kernel Clustering Algorithm [13, 16]

```
Input: $K$ - $n \times n$ kernel matrix; $m$ - fuzzifier
Data: $U \in [0,1]^{c\times n}$ cluster membership matrix
Run FCM to initialize $U$
Estimate $\nu_j$ by Eq. (11)
while $\max\{|U - U'|\} > \epsilon$ do
    $U' = U$
    Compute $d_\kappa(j,i)$, $\forall i, j$, using Eq. (10)
    Update $U$ using Eq.(7)
```

II. Large Scale c-Means Approximation Algorithms

The cluster centers in kernel c-means algorithms are represented by a linear sum of the high-dimensional transformed features $\phi(\mathbf{x}_i)$,

$$\phi(\mathbf{v}_j) = \sum_{i=1}^n \tilde{u}_{ij}\phi(\mathbf{x}_i), \tag{12}$$

where this equation is equivalent to (9) with $\tilde{u}_{ij}$ substituted. In data where the number of objects n is large, we hypothesize that the cluster centers $\phi(\mathbf{v}_j)$ can be accurately represented by a subset of the feature vectors $\phi(\mathbf{x})$. Let $\xi = (\xi_1, \ldots, \xi_n)$ be binary variables indicating the instances that are selected for constructing the cluster centers, with $\xi_i = 1$ if $\mathbf{x}_i$ is selected and 0 otherwise. Let s denote the number of selected objects, where $s = \sum_i \xi_i$. The following derivation follows, very closely, that of our previous work [17] on approximating HCM for large-scale data.

In our approximation, the cluster centers are approximated by a linear sum of the selected feature vectors

$$\hat{\phi}(\mathbf{v}) = \sum_{i=1}^n a_i \xi_i \phi(\mathbf{x}_i), \tag{13}$$

where a_i is the weight of the ith feature vector. We are only interested in the values of a_i when $\xi_i = 1$; hence, we denote the weights for the jth cluster center as $\alpha_j = (\alpha_{1j} = a_{(1)}, \ldots, \alpha_{sj} = a_{(s)})$, where $a_{(1)}, \ldots, a_{(s)}$ are the set of associated weights where $\xi = 1$. Although this notation seems cumbersome, it dramatically simplifies the description of our derivation. With this notation, we can rewrite (13) for the jth cluster center as

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

where it is clear now that we are representing the cluster center $\phi(\mathbf{v}_j)$ as a linear sum of s selected objects.

Now we want to solve for the weights α_j such that the distance between the true cluster center and the approximated cluster center, $||\phi(\mathbf{v}_j) - \hat{\phi}(\mathbf{v}_j)||^2$, is minimized. This distance can be written as

$$||\phi(\mathbf{v}_j) - \hat{\phi}(\mathbf{v}_j)||^2 = \left|\left|\sum_{i=1}^{n} \tilde{u}_{ij}\phi(\mathbf{x}_i) - \sum_{i=1}^{n} a_i\xi_i\phi(\mathbf{x}_i)\right|\right|^2$$
$$= \tilde{\mathbf{u}}_j^T K \tilde{\mathbf{u}}_j + (\mathbf{a}\circ\xi)_j^T K(\mathbf{a}\circ\xi)_j - 2\tilde{\mathbf{u}}_j^T K(\mathbf{a}\circ\xi)_j, \quad (15)$$

where $\circ$ indicates the Hadamard product or element-by-element multiplication (.* in MATLAB) and $(\mathbf{a}\circ\xi)_j$ is an n-length vector that represents the weights of the approximated cluster center $\hat{\phi}(\mathbf{v}_j)$. We can substitute α_j into (15) by defining two sub-matrices of the kernel matrix K,

$$K_{\xi\xi} = [K_{ij}]^{s\times s}, \ \xi_i = 1, \xi_j = 1, \quad (16)$$
$$K_{\xi} = [K_i]^{n\times s}, \ \xi_i = 1, \quad (17)$$

where K_i is the ith column of K. Notice that $K_{\xi\xi}$ is the $s\times s$ square matrix corresponding to the pairwise elements of K for the ξ-selected objects. K_ξ is the $n\times s$ rectangular matrix corresponding to the columns of K for the ξ-selected objects; also, $K_{\xi\xi}$ is a sub-matrix of K_ξ. Hence, we can now rewrite (15) as

$$||\phi(\mathbf{v}_j) - \hat{\phi}(\mathbf{v}_j)||^2 = \tilde{\mathbf{u}}_j^T K\tilde{\mathbf{u}}_j + \alpha_j^T K_{\xi\xi}\alpha_j - 2\tilde{\mathbf{u}}_j^T K_\xi \alpha_j. \quad (18)$$

We determine α_j by minimizing (18) with respect to α, which is solved by setting the derivative of (18) with respect to α_j to zero,

$$\frac{d||\phi(\mathbf{v}_j) - \hat{\phi}(\mathbf{v}_j)||^2}{d\alpha_j} = 2K_{\xi\xi}\alpha_j - 2\tilde{\mathbf{u}}_j^T K_\xi = 0,$$

leading to the solution

$$\alpha_j = -\left(K_{\xi\xi}^{-1}K_\xi^T\tilde{\mathbf{u}}_j\right)^T. \quad (19)$$

In practice, $K_{\xi\xi}$ could be rank-deficient (have a number of relatively small eigenvalues); hence, it is advisable to use a pseudo-inverse when computing $K_{\xi\xi}^{-1}$, which we denote as $K_{\xi\xi}^{\dagger}$. Also, 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 our approximation algorithms is the redefinition of Eq. (10) using the solution of α_j, i.e.,

$$\hat{d}_\kappa(j,i) = \alpha_j^t K_{\xi\xi}\alpha_j + K_{ii} - 2(K_\xi\alpha_j)_i. \quad (20)$$

Notice that we no longer require the full kernel matrix to compute the distance $\hat{d}_\kappa(j,i)$; the only required kernel matrix elements are the diagonal elements (which for the RBF are all equal to one) and the s columns of K corresponding to the selected objects.

We are now ready to define our large-scale approximations of the kernel FCM and PCM algorithms. Algorithms 4 and 5 outline the approximations of the kernel FCM and PCM algorithms, respectively, which we call akFCM and akPCM.

The computational complexity of the distance calculation in (20) is $O(sn)$, which for $s << n$ is a significant improvement on the $O(n^2)$ complexity of the full-kernel distance calculation in (10). Furthermore, only $sn + n - s$ elements of K need to be calculated—the sn elements of K_ξ and

Algorithm 4: Approximate Fuzzy c-Means (akFCM) Kernel Clustering Algorithm

```
Input: $K_\xi$ - $n \times s$ kernel sub-matrix; diag($K$) - Kernel
      matrix diagonal; $m$ - fuzzifier; $\xi$ - selection vector
Data: $U \in [0,1]^{c\times n}$ cluster membership matrix
Initialize $U$
$\hat{K} = -K_{\xi\xi}^{\dagger}K_\xi^T$
while $\max\{|U - U'|\} > \epsilon$ do
    $U' = U$
    $\alpha_j = \left(\hat{K}\tilde{\mathbf{u}}_j\right)^T, \ \forall j$
    $\hat{d}_\kappa(j,i) = \alpha_j^t K_{\xi\xi}\alpha_j + K_{ii} - 2(K_\xi\alpha_j)_i, \ \forall i,j$
    Update $U$ using Eq. (6)
```

Algorithm 5: Approximate Possibilistic c-Means (akPCM) Kernel Clustering Algorithm

```
Input: $K_\xi$ - $n \times s$ kernel sub-matrix; diag($K$) - Kernel
      matrix diagonal; $m$ - fuzzifier; $\xi$ - selection vector
Data: $U \in [0,1]^{c\times n}$ cluster membership matrix
$\hat{K} = -K_{\xi\xi}^{\dagger}K_\xi^T$
Run akFCM to initialize $U$
Estimate $\nu_j$ by Eq. (11)
while $\max\{|U - U'|\} > \epsilon$ do
    $U' = U$
    $\alpha_j = \left(\hat{K}\tilde{\mathbf{u}}_j\right)^T, \ \forall j$
    $\hat{d}_\kappa(j,i) = \alpha_j^t K_{\xi\xi}\alpha_j + K_{ii} - 2(K_\xi\alpha_j)_i, \ \forall i,j$
    Update $U$ using Eq. (7)
```

the $n - s$ additional diagonal elements—which both reduces the computation time to produce the kernel elements and the storage requirements. However, for very large n both the computational and memory requirements can still be significant—e.g. a $10,000 \times 1,000,000$ array of floats (4 bytes) requires 40 gigabytes of working memory. While this memory requirement is large (at the time this paper was written), it is manageable compared to the 4 terabyte memory requirement for the full $1,000,000 \times 1,000,000$ kernel matrix.

The computational complexity of the pseudo-inverse of $K_{\xi\xi}$ is $O(m^3)$, which could lead to problems for large m. However, in practice, one could employ gradient descent to solve for α.

Next, we move on to some experiments.

III. Experiments

We performed two sets of experiments. The first compared the performance of the proposed algorithms with the full kernel solution. The second set of experiments applies the proposed algorithms to data sets for which computing the full kernel solution was impossible because of memory limitations.

The experiments described in this paper 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 [18] computing environment.

A. Evaluation Criteria

We judge the performance of our proposed algorithms with three criteria. Each criteria is computed for 20 independent runs. For each run, we initialize the full kernel c-means and the sampled kernel c-means algorithms with the same initialization. Thus, both the approximation algorithm and the full kernel algorithm begin with the same initial partition, with a different initialization for each run. We initialize U by choosing c objects as the initial cluster centers. The value of $\epsilon = 1 \times 10^{-3}$ and the fuzzifier $m = 2$.

1) Speedup Factor: This criteria represents an actual run-time comparison. Speedup is defined as t_{full}/t_{samp}, where these values are times in seconds for computing the membership matrix U. Note that we de not include the time savings from having to only compute a portion of the kernel matrix in our calculation. Thus, the speedup we present is a conservative measure. In practice, one would see further improvement by only calculating the necessary portions of the kernel matrix—the $m \times n$ rectangular matrix K_ξ and the diagonal of K.

2) Cluster Purity: This measure is defined as

$$P = \frac{1}{n}\sum_{j=1}^{c} \max_{\omega \in \Omega} |\omega \cap C_j|, \tag{21}$$

where Ω is the set of all class labels and C_j is the set of labels of the objects in the jth cluster. This measure cannot be directly applied to soft partitions; hence, we first *harden* the partitions before applying (21). Hardening is the process of setting the maximum cluster membership in each column of U to 1 and all else to 0.

The figures in this paper show *relative purity*, which is calculated as $P_{samp} - P_{full}$ for each run.

3) Squared Error Distortion: This criteria is the c-means objective and is defined as

$$E = \sum_{j=1}^{c}\sum_{i=1}^{n} u_{ji}^m ||\phi(\mathbf{v}_j) - \phi(\mathbf{x}_i)||^2 = \sum_{j=1}^{c}\sum_{i=1}^{n} u_{ji}^m d_\kappa(j,i). \tag{22}$$

We present the % error in E, which is calculated as

$$\%\text{ Error} = \frac{E_{samp} - E_{full}}{E_{full}} * 100.$$

As with relative purity, % Error is only calculated between the approximation result and the full kernel result that start with the same initialization. By doing this, we are negating the effect that the initialization has on the result.

B. Comparison with Full Kernel c-Means

We compared the proposed approximation to the full kernel method on two different data sets.

1) **A3**: These data are composed of 7,500 2-dimensional vectors, with a visually-preferred grouping into 50 clusters.[1] Figure 1 shows a plot of these data. We used the RBF kernel with $\sigma = 0.5$.

[1]The A3 data was designed by Ilja Sidoroff and can be downloaded at http://cs.joensuu.fi/~isido/clustering/.

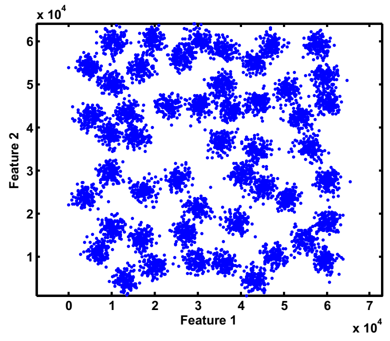


Fig. 1: Plot of A3 data set.

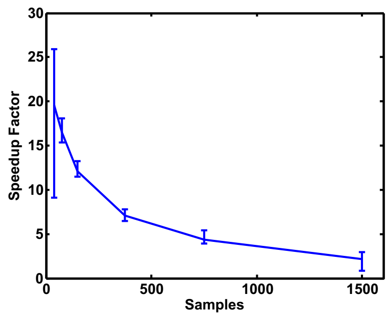
2) **MNIST**: This data set is a subset of the collection of handwritten digits available from the *National Institute of Standards and Technology* (NIST)[2]. There are 70,000 28×28 pixel images of the digits 0 to 9. Each pixel has an integer value between 0 and 255. We normalize the the pixel values to the interval $[0, 1]$ by dividing by 255 and concatenate each image into a 784-dimensional column vector. The kernel used for this example was a degree-5 polynomial kernel $\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + 1)^5$, which was shown to be effective in [19].

Figure 2 illustrates the evaluation criteria for the sFCM clustering algorithm on the A3 data set. The error-bars indicate the maximum and minimum values of each criteria over 20 independent runs. View (a) shows that an order of magnitude speedup is attained with sample sizes $m \leq 250$, or at a 3% sampling rate. Views (b) and (c) show that even at very low sample rates, the quality of the approximated clustering solution is equal to that of the full kernel solution and in one instance, at sample size $m = 1500$, the cluster purity was 2% greater than that of the full kernel solution.

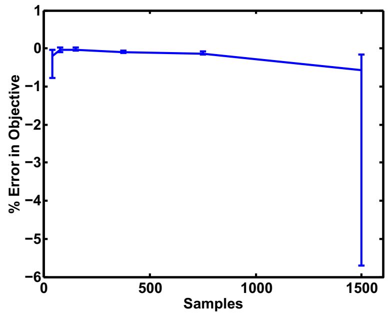
The results of the akPCM algorithm on the A3 data set are illustrated in Fig. 3. View (a) shows that akPCM provides similar speedup results as akFCM and views (b) and (c) demonstrate that the quality of the solution is better than that of the full kernel algorithm. We conjecture that this is because the influence of objects in regions where classes overlap is perhaps lessened by the sampling process.

Figures 4 and 5 show the results of the experiments performed on the MNIST data set. The results of the akFCM algorithm show significant speedup at low sampling rates (the two lowest sampling rates are 0.1% and 0.2%, respectively) with negligible difference in the quality of the clustering, relative to the full kernel c-means. Figure 3(a) shows that 1-2 orders of magnitude in speedup is achieved; however, in the akPCM experiments, a degradation in the cluster purity of about 3% was observed. Again, the lower sampling rates perform better than the higher sampling rates. We aim to investigate this phenomenon in subsequent studies and conjecture that this behavior is either due to the psuedo-inverse calculation in the approximation algorithm or because of the influence of outlier

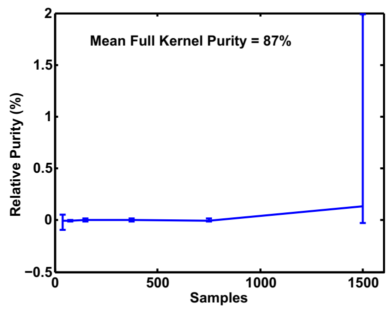
[2]The MNIST data can be downloaded at http://yann.lecun.com/exdb/mnist/.



(a) Speedup — t_{full}/t_{samp}

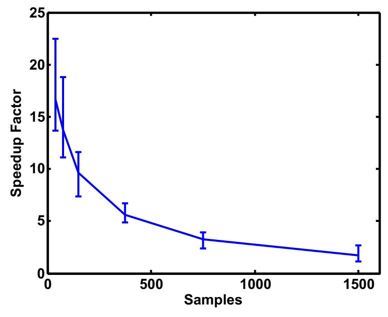


(b) % Error in squared distortion — $(E_{samp} - E_{full})/E_{full}$

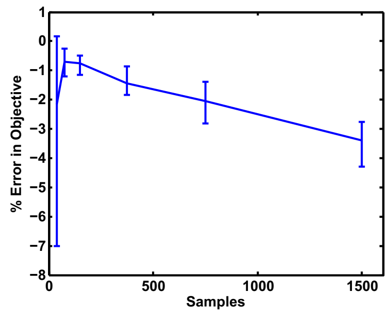


(c) Relative cluster purity — $P_{samp} - P_{full}$

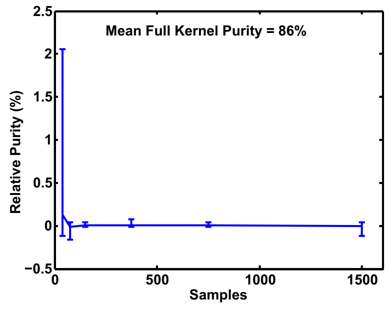
Fig. 2: Performance of akFCM on *A3* data set using RBF kernel with $\sigma = 0.5$, $n = 7500$, $c = 50$. Lines denotes mean values over 21 runs; error-bars represent maximum and minimum values. Membership matrix was hardened to compute cluster purity.



(a) Speedup — t_{full}/t_{samp}



(b) % Error in squared distortion — $(J_{samp} - J_{full})/J_{full}$



(c) Relative cluster purity — $P_{samp} - P_{full}$

Fig. 3: Performance of akPCM on *A3* data set using RBF kernel with $\sigma = 0.5$, $n = 7500$, $c = 50$. Lines denote mean values over 21 runs; error-bars represent maximum and minimum values. Membership matrix was hardened to compute cluster purity.

and overlapping classes.

C. Performance on Very Large Data Sets

Now we present the results of our proposed approximation algorithms on two data sets which are considerably larger than than the those in the previous experiments.

To store the full kernel matrix for either of the data sets in this section in single-precision format would require 1+ terabytes (1,000 gigabytes) of memory. At the time of this study, 1 terabyte of memory was an exceptionally large amount. Without considering distributed memory architectures, this is out of reach for most researchers and consumers.

1) **Forest Cover Type** [20][3]: These data are composed of cartographic variables obtained from *United States Geological Survey* (USGS) and *United State Forest Service* (USFS) data. 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 meter cells, which were then determined to be one of 7 forest cover types by the USFS. We normalize the features to the interval $[0, 1]$.

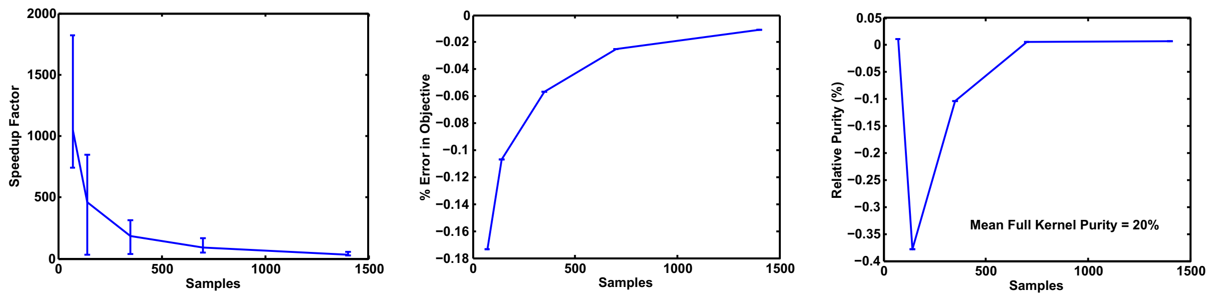
[3]The *Forest Cover Type* and *Quadreped Mammals* data sets can be downloaded at http://uisacad2.uis.edu/dstar/data/clusteringdata.html.

2) **Quadruped Mammals** [20]: These data are generated by the program animals.c, which can be downloaded from the UCI Machine Learning depository. The program generates instances of 72-dimensional feature vectors which belong to one of 4 classes of animals: dogs, cats, horses, and giraffes. We use 500,000 examples from this data set, with an approximately equal number of each class.

These problems become manageable by applying our approximation algorithms. We collected results for these two data sets at sample rates, $m = 0.01\%n$, $0.05\%n$, and $0.1\%n$. We produced 5 random draws of the selection variable ξ and for each random draw computed the akFCM and akPCM solutions for 5 different initializations; thus, a total of 25 experiments were done at each sample rate. Two different kernels were used, a degree-2 polynomial and an RBF with $\sigma = 2$.

Table II contains the run-time and cluster purity values for the two very large data sets. These results show that the akFCM algorithm can be used to cluster the Mammals data set in 8 seconds, with a cluster purity of 85%, and the Cover Type data set in 58 seconds, with a cluster purity of 52%.

For comparison, it was reported in [21] that the CLUTO [22] hard clustering algorithm achieved a purity score of 0.49 in on the Cover Type data and 0.82 purity on the Mammals data set. Our approximation algorithm produces results slightly

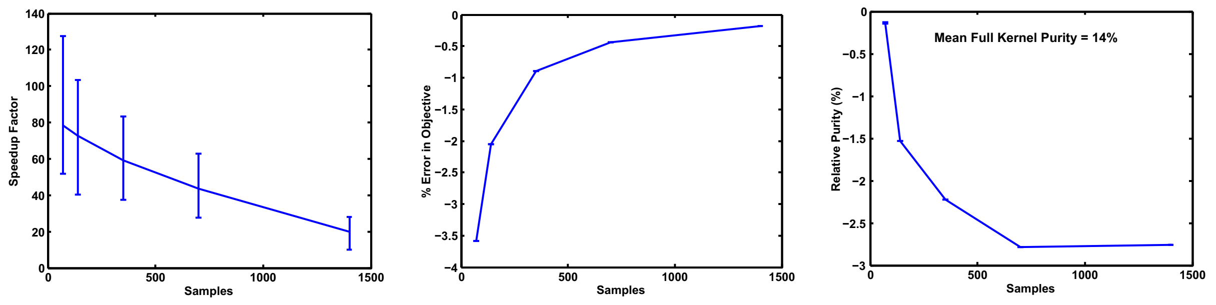


(a) Speedup — t_{full}/t_{samp}

(b) % Error in squared distortion — $(J_{samp} - J_{full})/J_{full}$

(c) Relative cluster purity — $P_{samp} - P_{full}$

Fig. 4: Performance of akFCM on *MNIST* data set using degree-5 polynomial kernel, $n = 70000$, $c = 10$. Lines denote mean values over independent 21 runs; error-bars represent maximum and minimum values. Membership matrix was hardened to compute cluster purity.



(a) Speedup — t_{full}/t_{samp}

(b) % Error in squared distortion — $(J_{samp} - J_{full})/J_{full}$

(c) Relative cluster purity — $P_{samp} - P_{full}$

Fig. 5: Performance of akPCM on *MNIST* data set using degree-5 polynomial kernel, $n = 70000$, $c = 10$. Lines denote mean values over independent 21 runs; error-bars represent maximum and minimum values. Membership matrix was hardened to compute cluster purity.

TABLE II: Clustering Results of akFCM and akPCM Algorithms on Very-Large Data

Data Set	Quadruped Mammals						Forest Cover Type					
Kernel	Degree-2 Polynomial			RBF ($\sigma = 1$)			Degree-2 Polynomial			RBF ($\sigma = 1$)		
Sample size m	50	250	500	50	250	500	59	291	582	59	291	582
akFCM	$c = 4$						$c = 7$					
Run-time (secs)	8	34	91	8	60	88	75	229	416	58	154	384
Purity	0.85	0.86	0.86	0.85	0.86	0.86	0.53	0.53	0.53	0.52	0.52	0.52
akPCM	$c = 4$						$c = 7$					
Run-time (secs)	30	129	199	35	151	224	125	352	600	87	215	570
Purity	0.68	0.69	0.71	0.68	0.69	0.69	0.50	0.50	0.50	0.49	0.49	0.49

better to those of CLUTO on these data. The GARDEN$_{HD}$ hard clustering algorithm [21] achieved a purity score of 0.65 on the Cover Type data; although, this algorithm is based on recursive division of the vector data and determined that the Cover Type data has 84 clusters. Cluster purity, by definition, tends to increase as the number of clusters increases; e.g., when the number of clusters $c = n$, purity $= 1$. Thus, comparing our results directly against GARDEN$_{HD}$ is not advised. The GARDEN$_{HD}$ algorithm achieved a purity of 0.76 with 1250 clusters and a purity of 1 with 6314 clusters on the Mammals data set. In comparison, akFCM achieved a purity of 0.86 with 4 clusters on this data and many of the individual runs in our experiment did produce a perfect cluster purity of 1. We believe that these results show that our algorithm is producing results that are as good as or better than existing algorithms that work on large scale data.

IV. Conclusion

We proposed an approximation of the fuzzy and possibilistic kernel c-means that has a reduced computational complexity and memory requirement. Our experimental results show that run-time can be decreased by 1-2 orders of magnitude, while achieving equal clustering performance to the classical kernel c-means. Furthermore, the memory requirements of our approximation is $O(mn)$, compared to $O(n^2)$ for the full kernel solution, where $m << n$.

In the future we are going to do a theoretical analysis on the loss imposed on the fuzzy and possibilistic c-means objective functions by the error in our approximation of the kernel-based distance between the cluster-center and objects. We have done analysis on a related approximation of the hard c-means, which can be found in [17]. In short, we discovered that with some modest assumptions the error imposed by our approximation on the hard c-means objective decreases at the rate of $O(1/m)$, which is intuitively pleasing. We expect similar results with the soft partitioning algorithms.

Also, we plan to investigate other sampling methods. There have been many sampling methods, both random and deterministic, proposed and analyzed for the Nyström method of approximating a kernel matrix [23, 24]. We aim to further investigate and build upon these methods to optimize the trade-off between the computational complexity and memory requirements and the imposed error.

Acknowledgements

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

References

[1] H. Frigui, *Advances in Fuzzy Clustering and Feature Discrimination with Applications*. John Wiley and Sons, 2007, ch. Simultaneous Clustering and Feature Discrimination with Applications, pp. 285–312.

[2] W. Bo and R. Nevatia, "Cluster boosted tree classifier for multi-view, multi-pose object detection," in *Proc. ICCV*, October 2007.

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

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

[5] R. Duda, P. Hart, and D. Stork, *Pattern Classification*, 2nd ed. Wiley-Interscience, October 2000.

[6] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed. San Diego, CA: Academic Press, 2009.

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

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

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

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

[11] A. Jain, M. Murty, and P. Flynn, "Data clustering: A review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, September 1999.

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

[13] R. Krishnapuram and J. Keller, "A possibilistic approach to clustering," *IEEE Trans. on Fuzzy Sys.*, vol. 1, no. 2, May 1993.

[14] I. Sledge, T. Havens, J. Bezdek, and J. Keller, "Relational duals of cluster validity functions for the c-means family," *IEEE Trans. Fuzzy Systems*, vol. 18, no. 6, pp. 1160–1170, Dec. 2010.

[15] R. Hathaway and J. Bezdek, "Optimization of clustering criteria by reformulation," *IEEE Trans. Fuzzy Systems*, vol. 3, pp. 241–245, 1995.

[16] R. Krishnapuram and J. Keller, "The possibilistic c-means: insights and recommendations," *IEEE Trans. Fuzzy Systems*, vol. 4, pp. 385–393, 1996.

[17] R. Chitta, T. Havens, A. Jain, and R. Jin, "Large scale clustering using approximate kernel k-means," in *To be submitted, Proc. ACM SIGKDD*, 2011.

[18] *Using MATLAB*, The Mathworks, Natick, MA, November 2000.

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

[20] A. Asuncion and D. Newman, "UCI machine learning repository," http://www.ics.uci.edu/~mlearn/MLRepository.html, 2007.

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

[22] G. Karypis, "CLUTO: A clustering toolkit," University of Minnesota, Computer Science, Tech. Rep. 02-017, 2003.

[23] P. Drineas and M. Mahoney, "On the Nyström method for approximating a gram matrix for improved kernel-based learning," *J. Machine Learning Research*, vol. 6, pp. 2153–2175, 2005.

[24] S. Kumar, M. Mohri, and A. Talwalkar, "Sampling techniques for the Nyström method," in *Proc. Int. Conf. Artificial Intelligence and Statistics*, 2009.