2014

SCALABLE APPROXIMATION OF KERNEL FUZZY C-MEANS

Zijian Zhang
Michigan Technological University

Follow this and additional works at: https://digitalcommons.mtu.edu/etds

Part of the Computer Engineering Commons

SCALABLE APPROXIMATION OF KERNEL FUZZY C-MEANS

By

Zijian Zhang

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Computer Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2014

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Computer Engineering.

Department of Electrical and Computer Engineering

Report Advisor:	Dr. *Timothy C. Havens*
Committee Member:	*Dr. Laura E. Brown*
Committee Member:	*Dr. Saeid V. Nooshabadi*
Department Chair:	*Dr. Daniel R. Fuhrmann*

Contents

List of Figures

List of Tables

Abstract

Virtually every sector of business and industry that uses computing, including financial analysis, search engines, and electronic commerce, incorporate Big Data analysis into their business model. Sophisticated clustering algorithms are popular for deducing the nature of data by assigning labels to unlabeled data. We address two main challenges in Big Data. First, by definition, the *volume* of Big Data is too large to be loaded into a computer's memory (this volume changes based on the computer used or available, but there is always a data set that is too large for any computer). Second, in real-time applications, the *velocity* of new incoming data prevents historical data from being stored and future data from being accessed. Therefore, we propose our *Streaming Kernel Fuzzy c-Means* (stKFCM) algorithm, which reduces both computational complexity and space complexity significantly. The proposed stKFCM only requires $O(n^2)$ memory where n is the (predetermined) size of a data subset (or data chunk) at each time step, which makes this algorithm truly scalable (as n can be chosen based on the available memory). Furthermore, only $2n^2$ elements of the full $N \times N$ (where $N >> n$) kernel matrix need to be calculated at each time-step, thus reducing both the computation time in producing the kernel elements and also the complexity of the FCM algorithm. Empirical results show that stKFCM, even with relatively very small n, can provide clustering performance as accurately as kernel fuzzy c-means run on the entire data set while achieving a significant speedup.

Chapter 1

Introduction

The ubiquity of personal computing technology has produced an abundance of staggeringly large data sets which may exceed the memory capacity of a computer (whether that computer is a cell phone or a high-performance cluster). One way that these large data are produced is streaming data, i.e., those that are presented to a system sequentially such that future data cannot be accessed. These challenges stimulate a great need for sophisticated algorithms by which one can elucidate the similarity and dissimilarity among and between groups in these gigantic data sets.

Clustering is an exploratory tool in which data are separated into groups, such that the objects in each group are more similar to each other than to those in different groups. Since there are no labels given for the data set, clustering is an unsupervised learning problem.

The applications of clustering algorithms are innumerable. The best example should be Google News. Articles on similar topic will be grouped into the same keyword everyday. Amazon adopts clustering algorithms into their recommendation system. Many supervised learning algorithms require clustering as a pre-step. Clustering is a common technique widely used in pattern recognition, data mining, and data compression for deducing the nature of a data set by assigning labels to unlabeled data. Clustering itself is a general task to be solved and plenty of algorithms have been proposed for solving this problem such as k-means, single-linkage clustering, and Gaussian-mixture-model clustering.

1.1 Big Data

The term Big Data usually refers to harnessing enoumous amount of data which are beyond the capability of traditional computing tools and algorithms. Along with the explosion of smartphones and wearable devices, every aspect of our lives, including behaviors, locations, temperature, or even humidity, could be gleaned and become accessable for analysts. Schönberger and Cukier argued a few revolutions for the age of Big Data in their book [1]. One of the ascendancy they mentioned was that instead of sampling, human beings usher in, for the first time, an era in which we are being capable of processing the entire dataset. Using more complete and comprehensive data will allow us to dicover hidden patterns and correlations which have never been revealed before.

The boosts brought by Big Data are remarkable. Soon exhaustively sequencing the DNA and RNA of every individual cell in a tumor will become reality. Engineers at Google are able to forecast the outbreak of winter flu weeks before CDC by looking at their more than 3 million search queries every day [2]. IBM employed Big Data on predicting the traffic congestions in Lyon, France, which allows the traffic department to react in advance. In 2012, the World Economic Forum declared data as a new class of economic asset just like gold [3]. The tide of Big Data has unpended areas as varied as sports, industry, and our daily lives.

Formidable processing power as well as the plummeting of storage costs form the basis for Big Data's flourish. Yet the amount of information is growing incredibly fast. Today the data we generate exceeds petabytes and is headed toward exabytes rapidly. As of 2013, Facebook has more than 4.75 billion content items shared and 350 million photos are uploaded every day [4]. The desire for space and computational power are exacerbated by the swelling flood of data. Myriad approaches have been proposed to embrace the upcoming challenges.

1.2 K-Means

The *k*-means algorithm is one of the most popular clustering algorithms due to its simplicity. For a set of *N* feature vectors (or objects), the program will choose *k* cohesive cluster centers randomly at the beginning. Each data point will be assigned to its nearest cluster center, then the cluster centers will be recomputed. These steps are repeated until the algorithm converges (and there are many ways by which convergence can be defined). The k-Means algorithm is guaranteed to converge. The objective function of k-Means is defined as

$$J(\mathbf{u};\mathbf{v}) = \sum_{i=1}^{n} ||\mathbf{x}_i - \mathbf{v}_{\mathbf{u}_i}||^2, \tag{1.1}$$

When we iteratively minimize J with respect to $\mathbf{u}$, $\mathbf{v}$ is fixed, and vice versa. J is monotonically decrease. So that the k-Means algorithm is coordinate descent on J.

The output of k-Means will be a partition matrix $U \in \{0,1\}^{N\times k}$, a matrix of Nk values. Each element u_{ik} is the membership of vector $\mathbf{x}_i$ in cluster k; and the partition matrix element $u_{ik} = 1$ if $\mathbf{x}_i$ belongs to cluster k and is 0 otherwise. Figure 1.1 shows an illustration of k-Means algorithm in two dimensional space. In (a), it gives us a visual view of the raw data. We first initialize $(k = 2)$ clusters randomly in (b). Each object will be assigned to its nearest cluster center as showed in (c). In (d) we move the cluster centers to be the mean of all objects that are assigned to them, then rerun the algorithm in (e). After the final relocation of cluster centers, the cluster centers are located as shown in (f).

Algorithm 1: k-Means

```
Input: number of clusters – $k$; $\mathscr{X} = \{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots\}$
Initialize cluster centers $(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k)$;
while $max_{1<j<k}||\mathbf{v}_{j,new} - \mathbf{v}_{j,old}||^2 > \varepsilon$ do
    1   $\mathbf{u}_i = \arg\min_j ||\mathbf{x}_i - \mathbf{v}_j||^2$
    2   $\mathbf{v}_j = \frac{\sum_{i=1}^n \mathbf{u}_i \mathbf{x}_i}{\sum_{i=1}^n \mathbf{u}_i}$
```

1.3 Fuzzy c-Means

The k-Means algorithm aims to group the given data into hard partitions, which means each data vector belongs to exactly one group. But groups in reality may not have well defined boundaries. For example, retailers will find that customers may have favors on different items to various degrees. Hard partitions won't give us any useful insights in cases like this. Looking at Figure 1.2, k-Means could be well deployed in case (a) because of the clear boundary between two colors. But case (b) won't fit any crisp clustering algorithms since the color changes gradually. Such challenge has given birth to Fuzzy Clustering which allows each object belongs to multiple groups to different degrees.

Fuzzy c-means (FCM) is analogous to the *k*-means algorithm with fuzzy partitions, which gives more flexibility in that each object can have membership in more than one cluster. The constraint on fuzzy partitions is that all the memberships of an object must sum to 1, thus ensuring that every object has unit total membership in a partition: $(\sum_k u_{ik} = 1)$. The

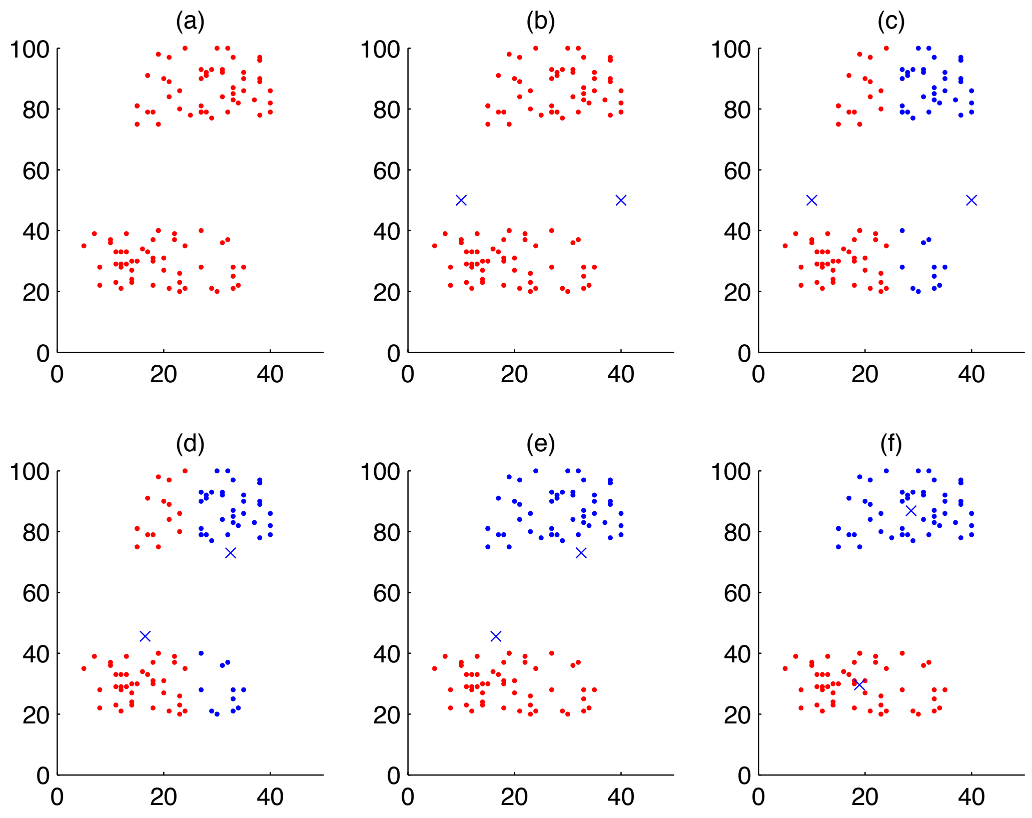

Figure 1.1: Example of *k*-Means Algorithm on a Data Set Composed of Two Clouds

objective function of FCM is shown as below,

$$J_m(\mathbf{u};\mathbf{v}) = \sum_{i=1}^{n}\sum_{j=1}^{c} u_{ij}^{m}||\mathbf{x}_i - \mathbf{v}_j||^2, \tag{1.2}$$

The result is still a partition matrix $U \in [0,1]^{N\times c}$, where the partition elements are now on the interval $[0,1]$. The parameter m>1 is the fuzzifcation constant. Figure 1.3 shows the membership functions of both crisp and fuzzy clustering on the example from Figure 1.2.

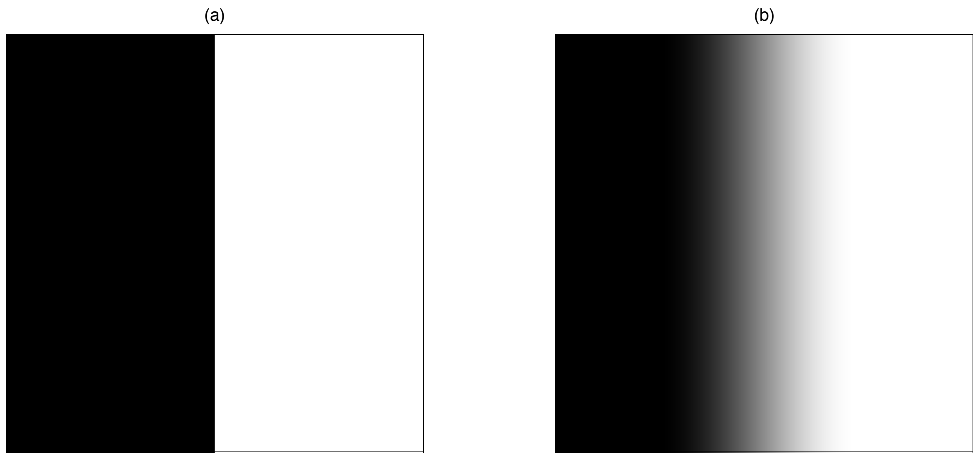

Figure 1.2: The Limitation of Crisp Clustering

Assume we have 650 samples. Using 0 and 1 to represent black and white respectively. Due to the fact that many of the datums do not belong exclusively to a well defined cluster, the membership function follows a smoother line indicates that those datums could belong to more than one clusters with different values of degrees. An absolute partition can hardly be found in reality, fuzzy clustering is more appropriate in practice. Applications of FCM are extensive. Search engines must be able to interpret fuzzy queries instead of particular keywords. Social swarms are never completely isolated to each other. Researchers are also interested in finding correlations between different clusters. Beside the result, FCM can illustrate the relationship as well.

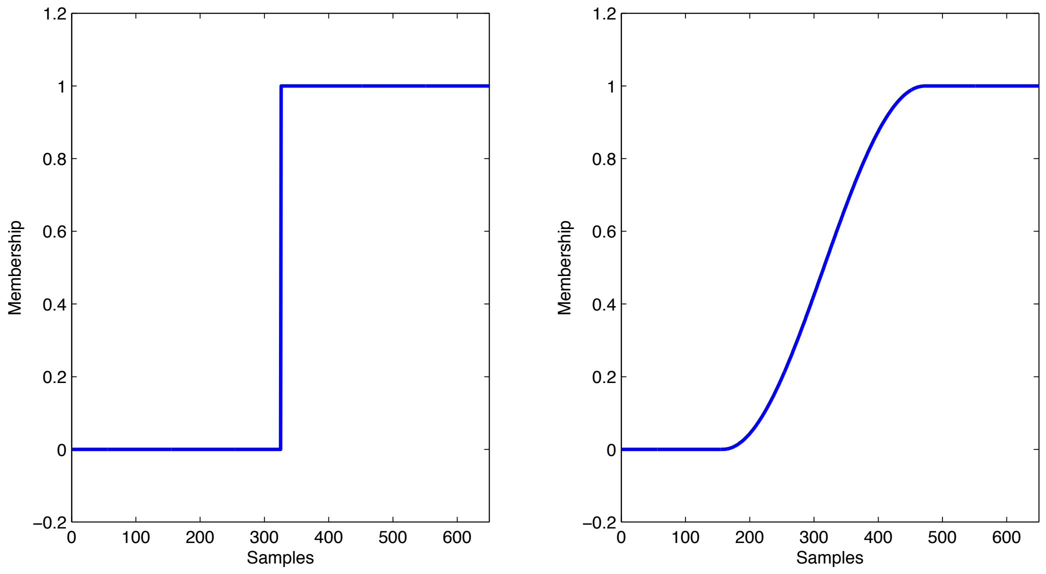

Figure 1.3: Membership Function of Crisp Clustering and Fuzzy Clustering

Algorithm 2: Fuzzy c-Means

```
Input: number of clusters – $c$; $\mathscr{X} = \{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots\}$
Initialize cluster centers $(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k)$;
while $max_{1<k<c}||\mathbf{v}_{k,new} - \mathbf{v}_{k,old}||^2 > \varepsilon$ do
    1   $\mathbf{u}_{ij} = \left[\sum_{k=1}^{c} \left(\frac{||\mathbf{x}_i - \mathbf{v}_j||}{||\mathbf{x}_i - \mathbf{v}_k||}\right)^{\frac{2}{m-1}}\right]^{-1}$
    2   $\mathbf{v}_j = \frac{\sum_{i=1}^{n} \mathbf{u}_{ij}^{m} \mathbf{x}_i}{\sum_{i=1}^{n} \mathbf{u}_{ij}{}^{m}}$
```

1.4 Kernel Methods

The FCM (as well as the *k*-means) model is based on the assumption that the feature vectors are grouped in similarly-sized hyperspheres. Kernel methods can overcome this limitation

Table 1.1
Important Acronyms and Notation

Acronym	**Definition**
FCM	fuzzy c-means
KFCM	kernel FCM
stKFCM	streaming KFCM
KPC	kernel patch clustering [5]
RKHS	reproducing kernel Hilbert space
Notation	**Definition**
c or k	number of clusters
N	number of objects
n	number of objects in data chunk
$\mathbf{x}$	feature vector $\in \mathscr{R}^d$
X	set of $\mathbf{x}$
$\mathscr{X}$	set of X, $\{X_0, \ldots, X_t, \ldots\}$
U	partition matrix
$\mathbf{u}_i$	ith column of U
$\mathbf{v}$	cluster center
$\mathbf{w}$	weight vector
ϕ_i	kernel representation of $\mathbf{x}_i$, i.e., $\phi(\mathbf{x}_i)$
Φ	set of ϕ
$\kappa(\mathbf{x}_i, \mathbf{x}_j)$	kernel function, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi_i \cdot \phi_j$
K	kernel matrix, $K = [\kappa(\mathbf{x}_i, \mathbf{x}_j)], \forall i, j$
$\mathscr{H}_K$	reproducing kernel Hilbert space imposed by K
$d_\kappa(\mathbf{x}_i, \mathbf{x}_j)$	kernel distance, $\|\|\phi_i - \phi_j\|\|^2$
$[i]$	set of integers, $\{1, 2, \ldots, i\}$

by projecting the vectors into a higher dimensional feature space where the patterns can be discovered as linear relations [6]. Figure 1.4 and Figure 1.5 shows how polynomial and RBF kernels apply on non-linearly separatable data respectively. By projecting to a higher dimensional feature space, a linear pattern is dicovered.

Consider some non-linear mapping function $\phi : \mathbf{x} \to \phi(\mathbf{x}) \in R^{D_\kappa}$ where D_κ is the dimensionality of the higher-dimensional feature space created by the function ϕ. For (most) kernel algorithms, including kernel FCM, explicitly transforming x is not necessary. Instead, a

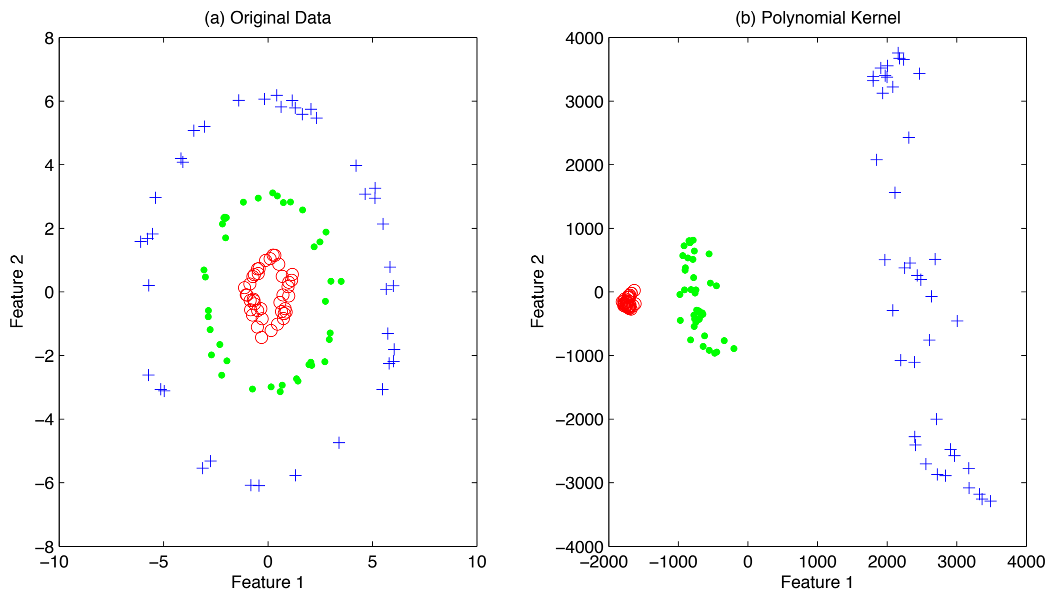

Figure 1.4: Projection by Polynomial Kernel

kernel matrix K is used, which consists of the pairwise dot products of the feature vectors in a transformed high dimensional space $\mathscr{H}_K$; this space is called the *Reproducing Kernel Hilbert Space* (RKHS). Using the kernel matrix is often computationally cheaper than explicitly representing the coordinates in RKHS. More details about Kernel FCM will be introduced in Chapter 3.

1.5 Challenges

The challenges in processing Big Data are twofold: the processing time can be long; and the memory that is required to retain the data exceeds the capability for the giving specification. Given a set of N objects, literal *kernel* FCM (KFCM) requires to store an $N \times N$ kernel

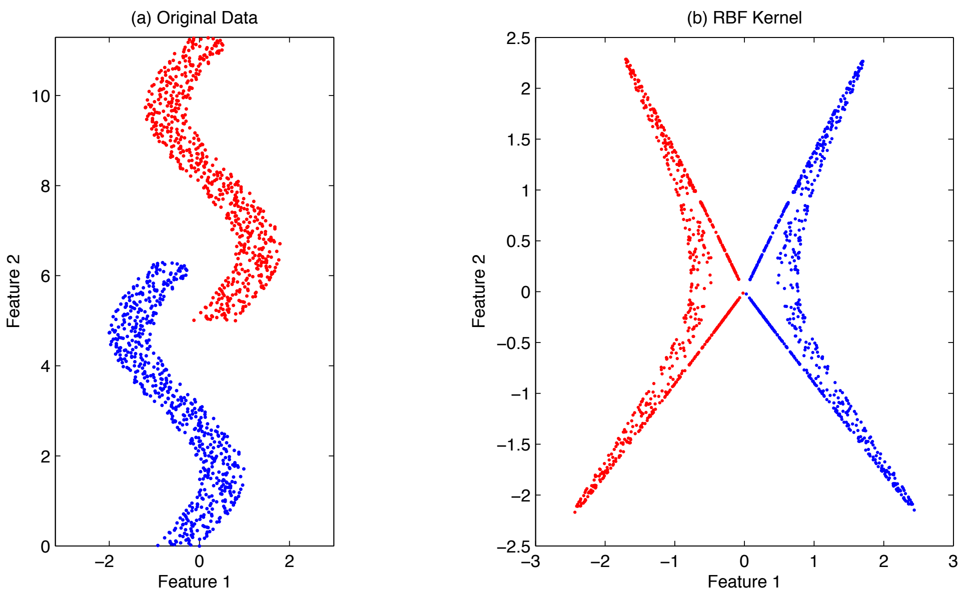

Figure 1.5: Projection by RBF Kernel

matrix $K = [\kappa(\mathbf{x}_i, \mathbf{x}_j)] = [\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)]$, $i, j \in [N]$, which poses challenges for processing very large N. Although an abundance of research has been conducted on fuzzy clustering algorithms for Big Data [5, 7–13], only a select few of these algorithms are appropriate for kernel methods.

In this report, we devise an approximation of the KFCM algorithm for streaming data, i.e., big data which could be viewed as data streams. We assess the performance of our algorithm by comparing the partition matrix to that of the literal KFCM. Empirical results demonstrate that our algorithm could provide similar results to the literal KFCM while only requiring access to small data chunks. The memory requirement is reduced from $O(N^2)$ to $O(n^2)$, where N and n are the size of the full data set and each subset data chunk, respectively.

The algorithms and results from this report were published in [14].

Chapter 2

Related Work

To date, a substantial number of algorithms have been developed for clustering big data. Roughly, these algorithms can be categorized into three classes: sampling, distributed clustering, and data transformation algorithms.

2.1 Sampling and non-iterative extension

Sampling the data set is the most basic and obvious way to address big data. In sampling methods, algorithms are run on a reduced representative sample, and then the sample partition is non-iteratively extended to approximate the clustering solution for the remaining data in the full data set. If the data are sufficiently sampled, there is only a small difference between the result of the approximated partition and the result of clustering the entire data set. Most sampling algorithms are also called extensible algorithms. The notion of extensibility for FCM was introduced in [15]. Algorithms that produce an approximate result of the full data set by first solving the problem using a sample set and then non-iteratively extending the result on the full data set are referred to as extensible algorithms.

Sampling approaches can be further divided into two categories: random sampling and progressive sampling. Progressive sampling schemes for FCM were well studied in [16]. The authors showed that progressive sampling is widely used in many clustering algorithms. Sampling schedule and termination criteria are the most central components of any of these approaches. The most well-known progressive sampling method is *generalized extensible fast* FCM [17] which is the extension of [15]. In [15], the algorithm starts with statistics-based progressive sampling and terminates with a representative sample that is appropriate to capture the overall nature of the data set. Reference [17] extends this algorithm so that it could be applied to more general cases. Instead of clustering numerical object data, [18] and [19] extend the algorithm to attack the problem of clustering numerical relational data. In kernel clustering, cluster centers are linear combinations of all the data points to be clustered; hence, the sampling approaches mentioned previously are inappropriate. The authors of [20] and [12] tackled the problem of kernel clustering by proposing a novel sampling of the kernel matrix which results in significant memory savings and computational complexity reduction while maintaining a bounded-error solution to the kernel k-means and KFCM problem; although, the solutions in [12, 20] are not truly scalable to big data as they require the loading of an $N \times n$ rectangular of the kernel matrix; hence, as N grows, eventually there is a point at which the only loadable n becomes less than 1, thus invalidating the scalability of the algorithm.

2.2 Distributed clustering

Distributed clustering algorithms can be classified into two types: incremental loading of data subsets that can be fit into current memory capacity and divide-and-conquer approaches.

2.2.1 Incremental clustering

Algorithms in this category sequentially load small chunks or samples of the data, clustering each chunk in a single pass, and then combining the results from each chunk. Representative algorithms are proposed in [21] and [22]. In these approaches, the clusters are updated periodically using information from both the incoming data and obsolete data.

Single pass FCM (spFCM) was proposed in [7]. This algorithm performs *weighted* FCM (wFCM) on sequential chunks of data, passing clustering centers from each chunk onto the next. At each time step, the algorithm clusters a union set consisting of data in the current step and the cluster centers passed from previous step. The authors of [8] extend [7] by incorporating more results from multiple preceding time steps and in [9], they propose an algorithm that passes *c* weight values which are sums of membership values of the points in the current subset onto next iteration. Another incremental algorithm called *bit-reduced* FCM (brFCM) was proposed in [10]. This algorithm first bins the data and then clusters the bin centers. The performance of brFCM highly depends on the binning strategy; brFCM has been showed to provide very efficient and accurate result on image data. Havens et al. extended spFCM and other incremental FCM algorithms to kernel clustering in [11], although the results of these kernel extensions were disappointing overall.

Bradley introduced a data compression technique in [23]. This algorithm uses a buffer to contain the current subset of data. The data are then compressed twice. In the first compression, objects that are unlikely ever to move to other clusters are discarded. Those objects are found by calculating the Mahalanobis distance between the object and the cluster center. Those that fall within a chosen radius are then discarded. In the second compression period, more cluster centers are introduced into the data set in order to find more stable points. After these two compressions, the space will be filled with new data. The algorithm will keep running until all the data have been processed. Farnstrom introduced a special case of this algorithm [24] in which all the points in the buffer are discarded each time.

Gupta uses evolutionary techniques to search for the global optimal solution to the sum of the squares (SSQ) problem which is required to find cluster centers [25]. Each chunk is viewed as a generation. The fittest cluster centers survive to the next generation; bad centers are killed off with new ones selected. In kernel methods, no actual cluster centers exist. Passing cluster centers to the next time step is unpractical. The author of [5] presented Kernel Patching Clustering (KPC) algorithm. This algorithm selects approximate pseudo-centers at each time step, merging them repeatedly until the entire data set has been processed. Since the space complexity depends on only the size of chunks, algorithms of this type are (usually) truly scalable.

2.2.2 Divide and conquer

Algorithms in this category cluster each chunk in sequence as well. But rather than passing the clustering solution from one chunk onto the next, these algorithms aggregate the solutions from each chunk in one final run. Due to this final run, most of the algorithms in this type are not truly scalable. *Online* FCM (oFCM) was proposed in [13]. oFCM aggregates the solutions from each data chunk by performing wFCM on all the resultant cluster centers. Again, it was shown in [11], that the kernel extension of oFCM performed poorly. Reference [26] views the problem of merging different results from disjoint data sets as the problem of reaching a global consensus, while [27] assigns a weight to the cluster centers in each chunk, and then performs LSEARCH on all the weighted centers retained. LSEARCH is a local search algorithm that starts with an initial solution and then refines it by making local improvements. Other algorithms like those proposed in [28] and [29] use special initialization techniques to improve the accuracy of the *k*-means algorithm.

2.3 Data transformation methods

Algorithms of this sort transform data into other structures with the intention of making the clustering process more efficient. Perhaps one of the earliest well-known clustering algorithms for data streams is BIRCH [30], which transforms the entire data set into a tree-like structure called a *clustering feature* (CF) tree. The leaves of the CF tree are then clustered. PAM [31] transforms the data set into a graph structure, and then searches for a minimum on the graph. CLARANS [32] is a variation of CLARA [31] and draws a sample of the data set and applies PAM on the sample. The key difference between CLARA and CLARANS is that CLARA draws the sample at the beginning of the search while CLARANS draws it at each step of the search. The benefit of the CLARANS approach over CLARA is that the search is not confined to a localized area.

While many of the algorithms mentioned in Chapter 2 produce high-quality partitions for big data sets, unless noted, they are not appropriate for kernel clustering. The only (that we know of) truly scalable approach to kernel fuzzy clustering is the spKFCM algorithm proposed in [11, 33] and, as shown in [11], the spKFCM approach produces less-than-desirable results for some data sets.

Chapter 3

Kernel FCM

The literal FCM algorithm can not deal with data in hyperspheres with different dimensions. Kernel FCM will be adopted to address problems like this. Kernel methods always comprise two steps: projecting the data into higher dimensional feature space and implementation of learning algorithms to detect patterns. The explicit computation of each object in feature space is infeasible. Since kernel FCM is based only on the Euclidean distance in feature space, we could represent kFCM by using pairwise inner products of feature vectors in the transformed high dimensional space. The function that performs this transition directly from the inputs is known as the kernel function.

3.1 Kernel Functions

There are numerous kernel functions used in the literature such as linear forms, Polynomial functions, and *Radial Basis Function* (RBF). Some popular kernel functions are shown in Table 3.1. It is worth to mention that many kernel functions are the variation of RBF kernel. The choise of kernel function highly depends on the task and domain knowledge. The right kernel function should capture the similarity among data and also require significantly less computation than explicit mapping to the feature space. For example, kernel FCM with linear function (i.e., the Euclidean dot product) is the same as the literal FCM. It can extract patterns only in hyperplanes, in contrast with the RBF function, which allows us to solve problems in a transformed high-dimensional space. A good kernel function should have a positive definite Gram Matrix, with the aim that the optimization problem will be convex. But there are many kernel functions which are not positive definite and still work very well in practice.

Table 3.1
Popular Kernel Functions

Category	Definition
Linear Kernel	$k(x,y) = x^T y + c$
Polynomial Kernel	$k(x,y) = \left(\alpha x^T y + c\right)^d$
Gaussian Kernel(RBF)	$k(x,y) = exp\left(-\frac{\lvert\lvert x-y\rvert\rvert^2}{2\sigma^2}\right)$
Exponential Kernel	$k(x,y) = exp\left(-\frac{\lvert\lvert x-y\rvert\rvert}{2\sigma^2}\right)$
Laplacian Kernel	$k(x,y) = exp\left(-\frac{\lvert\lvert x-y\rvert\rvert}{\sigma}\right)$
ANOVA Kernel	$k(x,y) = \sum_{k=1}^{n} exp\left(-\sigma\left(x^k - y^k\right)^2\right)$
Sigmoid Kernel	$k(x,y) = tanh(\alpha x^T y + c)$
Rational Quadratic Kernel	$k(x,y) = 1 - \frac{\lvert\lvert x-y\rvert\rvert^2}{\lvert\lvert x-y\rvert\rvert^2 + c}$

3.2 Kernel FCM

Kernel FCM (KFCM) can be generally defined as the constrained minimization of

$$J_m(U;\kappa) = \sum_{j=1}^{c}\sum_{i=1}^{n} u_{ij}^m ||\phi_i - \mathbf{v}_j||^2, \tag{3.1a}$$

$$= \sum_{j=1}^{c}\left(\sum_{i=1}^{n}\sum_{k=1}^{n}\left(u_{ij}^m u_{kj}^m d_\kappa(\mathbf{x}_i,\mathbf{x}_k)\right)/2\sum_{l=1}^{n} u_{lj}^m\right), \tag{3.1b}$$

where U is a fuzzy partition, $m > 1$ 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 *i*th and *k*th feature vectors. The function $\kappa(\mathbf{x}_i,\mathbf{x}_k) = \phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_k)$ is the kernel function.

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

$$u_{ij} = \left[\sum_{k=1}^{c}\left(\frac{d_\kappa(\mathbf{x}_i,\mathbf{v}_j)}{d_\kappa(\mathbf{x}_i,\mathbf{v}_k)}\right)^{\frac{1}{m-1}}\right]^{-1}, \forall i,j, \tag{3.2}$$

where the kernel distance between input datum $\mathbf{x}_i$ and cluster center $\mathbf{v}_j$ (in the RKHS) is

$$d_\kappa(\mathbf{x}_i,\mathbf{v}_j) = ||\phi(\mathbf{x}_i)-\mathbf{v}_j||^2. \tag{3.3}$$

The cluster centers $\mathbf{v}$ are linear combinations of the feature vectors,

$$\mathbf{v}_j = \frac{\sum_{l=1}^{n} u_{lj}^m \phi(\mathbf{x}_l)}{\sum_{l=1}^{n} u_{lj}^m}. \tag{3.4}$$

Equation (3.3) 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 (3.4) into (3.3) we get

$$\begin{aligned} d_\kappa(\mathbf{x}_i,\mathbf{v}_j) =& \frac{\sum_{l=1}^{n}\sum_{s=1}^{n} u_{lj}^m u_{sj}^m \phi(\mathbf{x}_l)\cdot\phi(\mathbf{x}_s)}{\sum_{l=1}^{n} u_{lj}^{2m}} \\ &+ \phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_i) - 2\frac{\sum_{l=1}^{n} u_{lj}^m \phi(\mathbf{x}_l)\cdot\phi(\mathbf{x}_i)}{\sum_{l=1}^{n} u_{lj}^m} \\ =& \tilde{\mathbf{u}}_j^T K \tilde{\mathbf{u}}_j + \mathbf{e}_i^T K \mathbf{e}_i - 2\tilde{\mathbf{u}}_j^T K \mathbf{e}_i \\ =& \tilde{\mathbf{u}}_j^T K \tilde{\mathbf{u}}_j + K_{ii} - 2(\tilde{\mathbf{u}}_j^T K)_i, \end{aligned} \tag{3.5}$$

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 [34] and, furthermore, is identical to *relational* FCM [35] if the kernel $\kappa(\mathbf{x}_i, \mathbf{x}_k) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ is used [36].

Equation (3.5) shows the obvious problem which arises when using kernel clustering with big data: the distance equation's complexity is quadratic with the number of objects, i.e. $O(N^2)$ (assuming the kernel matrix is precomputed). Furthermore, the memory requirement to store K is also quadratic with the number of objects.

3.3 Weighted KFCM

Assume that each data point in X has a different weight, w_i, which represents its influence on the clustering solution. These weights can be applied to the KFCM objective at (3.1b) by

$$J_m(U;\kappa) = \sum_{j=1}^{c} \sum_{i=1}^{n} w_i u_{ij}^m ||\phi_i - \mathbf{v}_j||^2, \tag{3.6}$$

where it is now obvious how $\mathbf{w} \in \mathscr{R}^n$, $w_i \geq 0$, affects the solution of KFCM. Hence, the only difference between KFCM and wKFCM is that the distance at (3.5) in the iterated updates of (3.2) are computed with the weights included, i.e.,

$$\begin{aligned} d_{\kappa}^{\mathbf{w}}(\mathbf{x}_i, \mathbf{v}_j) = & \frac{1}{\|\mathbf{w} \circ \mathbf{u}_j^m\|} (\mathbf{w} \circ \mathbf{u}_j^m)^T K (\mathbf{w} \circ \mathbf{u}_j^m) + K_{ii} \\ & - \frac{2}{\|\mathbf{w} \circ \mathbf{u}_j^m\|} \left((\mathbf{w} \circ \mathbf{u}_j^m)^T K \right)_i , \end{aligned} \tag{3.7}$$

where **w** is the vector of weights and $\circ$ indicates the Hadamard product. The idea behind wKFCM will be instrumental in our design of the proposed stKFCM algorithm.

Chapter 4

Streaming KFCM Algorithm

Consider a streaming data set $\mathscr{X} = \{X_1, X_2, \ldots, X_t, \ldots\}$, and its projection onto a set of RKHSs $\mathscr{H}_{K^t}$, such that $\Phi = \{\phi_1, \phi_2, \ldots, \phi_t, \ldots\}$ are the kernel representations of $\mathscr{X}$, where $\Phi_t = \left\{\phi_1^t, \phi_2^t, \ldots, \phi_n^t\right\}$, and $\phi_i^t = \phi\left(\mathbf{x}_i^t\right) \in \mathscr{H}_{K^t}$. The goal of the proposed stKFCM algorithm is to approximate the clustering solution of KFCM on $\mathscr{X}$, while only having access to a limited number of chunks of $\mathscr{X}$ up to some time t. The naive solution is to store all history of $\mathscr{X}$ and run KFCM on all the samples. However, the memory requirement for storing the kernel matrix K is $O(N^2)$, where N is the number of samples in the history; hence, KFCM is not appropriate for big data, which all streaming data sets become at some point. The proposed stKFCM algorithm only requires access to X_t and X_{t-1} at time step t and only requires $O(n^2)$ storage requirement, where n is the size of the data chunk X_t. Figure 4.1 illustrates the sub-matrices required by the stKFCM algorithm.

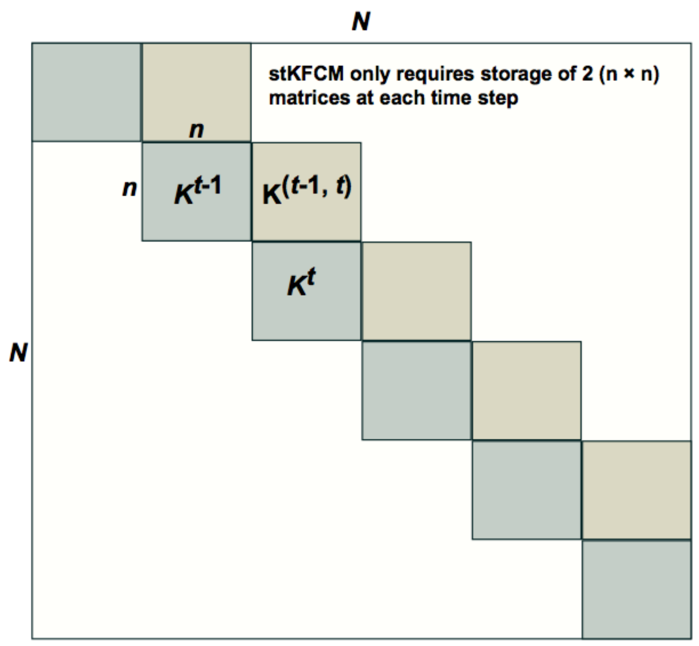
N
stKFCM only requires storage of 2 (n × n) matrices at each time step
n
n
K^{t-1}
$K^{(t-1, t)}$
K^{t}
N

Figure 4.1: Streaming KFCM

The stKFCM algorithm only requires storage of two $(n \times n)$ portions of the full $(N \times N)$ kernel matrix at each time-step. The sub-matrix $K^{t,t-1}$ is used to project the cluster centers from time $(t-1)$ into the RKHS imposed by the kernel matrix K^t.

In similar spirit to stKFCM, Havens proposed the *streaming kernel k-means* (stKKM) algorithm in [37]. It provides accurate results for using kernel k-means with streaming or incremental data. The main idea of this algorithm is to take the cluster centers $V^{t-1} = \{\mathbf{v}_1^{t-1}, \ldots, \mathbf{v}_c^{t-1}\}$, where $\mathbf{v}_i^{t-1} \in H_{K^{t-1}}$, and project them into H_{K^t} (the RKHS produced by K^t) as meta-vectors. We will explain meta-vectors in the following section, which proposes stKFCM. Using these meta-vectors, information is passed from previous time steps into the current time step. Then, at each time step, the data chunk X_t is clustered together with the (appropriately weighted) meta-vectors. We now propose the use of meta-vectors to approximate KFCM with the stKFCM algorithm.

4.1 Meta-vectors

Assume we are clustering not only the current data chunk X_t, but also a set of meta-vectors $A = \{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\}$, $\mathbf{a}_i \in H_{K^t}$. The meta-vectors $\mathbf{a}_i$ are linear combinations of all $\phi_i^t \in \Phi_t$; i.e., $\mathbf{a}_j = \sum_{i=1}^{n} \alpha_{ij}\phi_i$, $\alpha_{ij} \in \mathscr{R}$.

Proposition 1. Let the partitions U^t and U^α denote fuzzy partition values of X_t and A, respectively. Let $\mathbf{w}^{X_t}$ and $\mathbf{w}^\alpha$ be the weights of X_t and A. Since the cluster centers are linear combinations of the feature vectors, the cluster center $\mathbf{v}_c^t$ in the kernel feature space can be written as

$$\mathbf{v}_k^t = \sum_{i=1}^{n} \tilde{q}_{ik}^t, \tag{4.1}$$

where

$$\mathbf{q}_k^t = \begin{pmatrix} w_1^{X_t}\left(u_{1k}^t\right)^m + \sum_{j=1}^{c} \alpha_{1j}^t w_j^\alpha (u_{jk}^\alpha)^m \\ w_2^{X_t}\left(u_{2k}^t\right)^m + \sum_{j=1}^{c} \alpha_{2j}^t w_j^\alpha (u_{jk}^\alpha)^m \\ \cdots \\ w_n^{X_t}\left(u_{nk}^t\right)^m + \sum_{j=1}^{c} \alpha_{nj}^t w_j^\alpha (u_{jk}^\alpha)^m \end{pmatrix}; \tag{4.2a}$$

$$\tilde{\mathbf{q}}_k^t = \frac{\mathbf{q}_k^t}{|\mathbf{w}^{X_t} \circ (\mathbf{u}_k^t)^m| + |\mathbf{w}^{\alpha} \circ (\mathbf{u}_k^{\alpha})^m|}. \tag{4.2b}$$

Proof. The meta-partition $\tilde{\mathbf{q}}^t$ at (4.2) is formed by representing the cluster center $\mathbf{v}_k^t$ as the weighted linear sum of the data chunk Φ_t (i.e., X_t) and the meta-vectors A, and using the fact that A is a linear sum of Φ_t itself. See [37] for a more detailed proof of a similar proposition. $\square$

Now we show how to project the cluster centers produced at time $t-1$ (i.e., in $\mathscr{H}_{K^{t-1}}$) to the current time t. Note that this proposition is similar to that proposed in [37] for the use with stkKM; however, it is important to note that the formulation of $\tilde{\mathbf{q}}_k$ for KFCM clustering is different from that of kernel k-means.

Proposition 2. A projection of $\mathbf{v}_k^{t-1} \in H_{K^{t-1}}$ into H_{K^t} can be computed by the optimization over meta-vectors $\mathbf{a}_k$,

$$\underset{\mathbf{a}_k}{\arg\min} \, ||\mathbf{v}_k^{t-1} - \mathbf{a}_k||^2 = \sum_{i=1}^{n} \alpha_{ik}^t \phi_i^t. \tag{4.3}$$

Proof. The optimization has the closed form solution of

$$\alpha_k^t = \left(K^t\right)^{-1} K^{(t,t-1)} \tilde{\mathbf{q}}_k^{t-1}, \tag{4.4}$$

$$K^{(t,t-1)} = \kappa(\mathbf{x}_i^t, \mathbf{x}_j^{t-1}), i, j = 1, ..., n. \tag{4.5}$$

$\square$

Remark 1. The closed form solution for the weights α_k^t at (4.4) is the solution to the optimization under the squared Euclidean norm. The inverse operation in (4.4) is often best replaced, in practice, with a pseudo-inverse, which we denote by $(\cdot)^\dagger$. One could also use a simple gradient descent to minimize the quadratic. Furthermore, this is only one way to project the cluster center $\mathbf{v}_k^{t-1}$ into the current RKHS $\mathscr{H}_{K^t}$. We imagine that an L1-norm optimization could also find utility when a sparser solution for α_k^t is desired.

The distances between the cluster center $\mathbf{v}_k^T$ and each of $\mathbf{a}_i$, ϕ_i^t, and an arbitrary feature vector $\phi(\mathbf{x})$ are computed as

$$||\mathbf{a}_i - \mathbf{v}_k^t||^2 = (\alpha_i^t)^T K^t \alpha_i^t + (\tilde{\mathbf{q}}_k^t)^T K^t \tilde{\mathbf{q}}_k^t - 2(\alpha_i^t)^T K^t \tilde{\mathbf{q}}_k^t; \tag{4.6a}$$

$$||\phi_i^t - \mathbf{v}_k^t||^2 = K_{ii}^t + (\tilde{\mathbf{q}}_k^t)^T K^t \tilde{\mathbf{q}}_k^t - 2(K^t \tilde{\mathbf{q}}_k^t)_i; \tag{4.6b}$$

$$||\phi(\mathbf{x}) - \mathbf{v}_k^t||^2 = \kappa(\mathbf{x}, \mathbf{x}) + (\tilde{\mathbf{q}}_k^t)^T K^t \tilde{\mathbf{q}}_k^t - 2\kappa(\mathbf{x}, X_t)\tilde{\mathbf{q}}_k^t. \tag{4.6c}$$

These distances at (4.6) allow us to propose the stKFCM algorithm at Algorithm 3. The algorithm has the following basic steps:

1. The KFCM solution is computed for the first data chunk;
2. The weights of each cluster center are computed as the sum of the partition elements associated with each center;
3. The cluster centers are projected into the next time step;
4. The meta-partition $\tilde{\mathbf{q}}^t$ is computed;
5. The partition of the meta-vectors A is updated;
6. The partition of the feature vectors X^t is updated;
7. Optionally, the partition of the full data set X can be computed in one single-pass at the end.

The stKFCM algorithm is essentially a single-pass algorithm that computes the KFCM cluster solution of each X^t together with the weighted meta-vectors A, which are the projected cluster centers from step $t-1$. Hence, the (compressed) information from all previous time-chunks is passed down through the meta-vectors A.

Remark 2. The important projection step at Line 3 of the stKFCM algorithm is equivalent to taking the vectors $\mathbf{v}^{t-1}$ (appended by n zeroes) represented in the RKHS of the kernel matrix

$$K_{t-1,t} = \begin{bmatrix} K^{t-1} & K^{(t-1,t)} \\ K^{(t,t-1)} & K^{t} \end{bmatrix},$$

and using the Nystrom approximation to represent them in the low-rank approximation of the kernel matrix computed as (and reordered such that the time t columns are first)

$$\tilde{K}_{t,t-1} = \left[K^t | K^{(t,t-1)}\right]^T \left(K^t\right)^{-1} \left[K^t | K^{(t,t-1)}\right].$$

Furthermore, it is known that the error $\|K_{t,t-1} - \tilde{K}_{t,t-1}\|_2 \leq \lambda_{n+1} + O(N/\sqrt{n})$, where λ_{n+1} is the $(n+1)$th eigenvalue of $K_{t,t-1}$ [38]. Jin et al. [39] also showed that this error is further bounded if there is a large eigengap, which is often the case for data sets that can be partitioned into high-quality clusters. What this shows for the stKFCM algorithm is that the projection error at each step is bounded; hence, the total error is bounded by the size and number of data chunks used to complete the stKFCM process.

Algorithm 3: Streaming Kernel Fuzzy c-Means (stKFCM)

```
  Input: number of clusters – $c$; fuzzifier – m; $\mathscr{X} = \{X_0, X_1, X_2, \ldots\}$; kernel – $\kappa$
  Compute $K^0 = \kappa(X_0, X_0)$
1 $U^0 = \text{KFCM}(c, m, K^0)$
  $\tilde{\mathbf{q}}_c^0 = \mathbf{u}_c^0 / |\mathbf{u}_c^0|$, $c = 1, \ldots, k$
  for $t = 1, 2, \ldots$ do
      $K^t = \kappa(X_t, X_t)$, $K^{(t,t-1)} = \kappa(X_t, X_{t-1})$
      for $k = 1$ to $c$ do
2         $w_k = |\mathbf{q}_k^{t-1}|$
3         $\alpha_k^t = (K^t)^\dagger K^{(t,t-1)} \tilde{\mathbf{q}}_k^{(t-1)}$
      $u_{ik}^\alpha = 1$, if $i = k$, else $u_{ik}^\alpha = 0$, $U^t = [0]^{n \times c}$
      while any $u_{ij}^\alpha$ or $u_{ij}^t$ changes do
4         Compute $\tilde{\mathbf{q}}_k^t$ with (4.2)
          for $i, j = 1, \ldots, c$ do
5             $u_{ij}^\alpha = \left[ \sum_{k=1}^{c} \left( \frac{d_\kappa(\mathbf{a}_i, \mathbf{v}_j^t)}{d_\kappa(\mathbf{a}_i, \mathbf{v}_k^t)} \right)^{\frac{1}{m-1}} \right]^{-1}$
              where $d_\kappa(\mathbf{a}_i, \mathbf{v}_j^t)$ is computed with (4.6a)
          for $i = 1, \ldots, n,\ j = 1e, \ldots, c$ do
6             $u_{ij}^t = \left[ \sum_{k=1}^{c} \left( \frac{d_\kappa(\mathbf{x}_i^t, \mathbf{v}_j^t)}{d_\kappa(\mathbf{x}_i^t, \mathbf{v}_k^t)} \right)^{\frac{1}{m-1}} \right]^{-1}$
              where $d_\kappa(\mathbf{x}_i^t, \mathbf{v}_j^t)$ is computed with (4.6b)
      Compute $\tilde{\mathbf{q}}_k^t$ with (4.2)
7 Optional extension: The partition of the full data set $X$ is computed by the following steps.
  for $i = 1, \ldots, N,\ j = 1, \ldots, c$ do
      $u_{ij} = \left[ \sum_{k=1}^{c} \left( \frac{d_\kappa(\mathbf{x}_i, \mathbf{v}_j^t)}{d_\kappa(\mathbf{x}_i, \mathbf{v}_k^t)} \right)^{\frac{1}{m-1}} \right]^{-1}$
      where $d_\kappa(\mathbf{x}_i, \mathbf{v}_j^t)$ is computed by (4.6c).
```

Chapter 5

Experiments

We evaluated the performance of our algorithm on data sets for which ground truth exist. In these experiments, we compared the hardened partition from the proposed stKFCM to the the recently proposed Kernel Patch Clustering (KPC) [5], as well as the KFCM partition run on the whole data set. We present results for different chunk sizes, from $0.0001N$ to $0.5N$. The value of the fuzzifier m was fixed at 1.7. The experiments on the 2D15 and 2D50 were run on a Intel Core 2 Duo core processor with 4 GB of memory. Results of MNIST and Forest data set were generated by a quad-core CPU with 32 GB of memory. The results are expressed as the mean and standard deviation over 100 independent experiments, with random initializations and random data sample ordering.

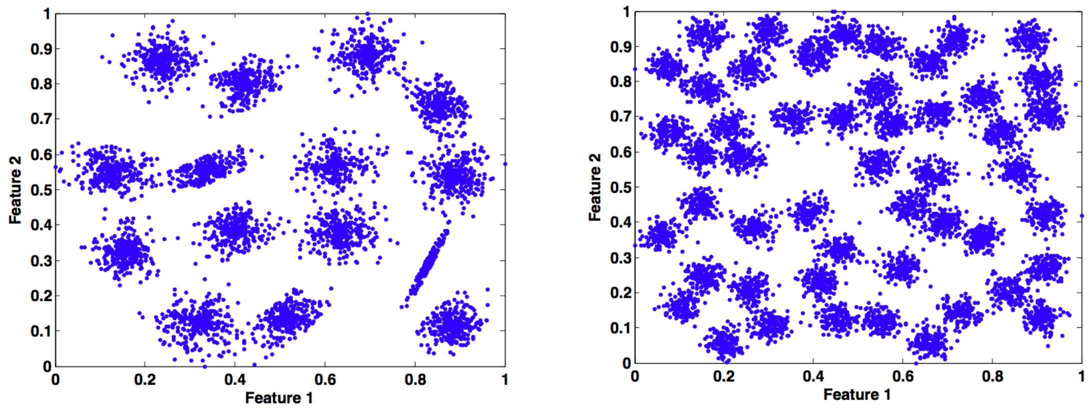

Figure 5.1: Synthetic data sets

5.1 Data sets

5.1.1 2D15

This synthetic data set is composed of 5,000 2-dimensional vectors. As shown in Fig. 5.1(a), it is obvious that 15 clusters are preferred in this data set. We used an RBF kernel with width of 1 on this data set.

5.1.2 2D50

This data set consists of 7,500 2-dimensional vectors with 50 clusters preferred. An RBF kernel with a width of 1 was used.

5.1.3 MNIST

These data were collected from 70,000 28×28 images of handwriting digits from 0 to 9 by the *National Institute of Standards and Technology* (NIST). We normalized the value of each pixel to the unit interval and organized the pixels column-wise into a single 784-dimensional vector. Therefore, this data set is composed by 70,000 784-dimensional vectors with 10 clusters preferred. An inhomogeneous polynomial kernel with degree of 5 was used in our experiment.

5.1.4 Forest

This data set is composed of 581,012 cartographic variables that were collected by the *United States Geological Survey and United State Forest Service* (USFS) data. There are 10 quantitative variables and 44 binary variables. These features were collected from a total of 581,012 30×30 meter cells of the forest, which were then determined to be one of 7 forest cover types by the USFS. We normalized the features to the unit interval by subtracting the minimum and then dividing by the subsequent maximum. We used the RBF kernel with a width of 1.

5.2 Evaluation criteria

5.2.1 Adjusted Rand Index

Rand index is one of the most popular comparison indices of measuring agreement between two crisp partitions of a data set. It is the ratio of pairs of agreement to the number of pairs. The *Adjusted Rand Index* (ARI) we are using here is a bias-adjusted formulation developed by Hubert and Arabie [40]. The result is a number between 0 and 1 where 1 indicates perfect match. In order to compute ARI, we first harden the fuzzy partition and then compare it with the ground-truth partition.

5.2.2 Purity

Purity, also called clustering accuracy, is an external validation measure to evaluate the quality of the clustering solution. The purity of each cluster is given by the ratio between the amount of right assignments in this cluster and the size of the cluster. The purity of the clustering solution is then expressed as a weighted sum of the individual purities. Thus, the purity is a real number between 0 and 1. The larger the purity, the better the performance.

5.2.3 Run time

Our algorithm could be used either as an approximation for unloadable data or acceleration for loadable data. Thus, time consumption is a crucial criteria. We compared times to compute the partition matrix with different chunk size as well as to KFCM run on the entire data set. All the times are recorded in seconds.

Table 5.1 contains the results of our experiments. On the 2D15 and 2D50 data sets, both KPC and stKFCM are successful at finding the preferred partitions. However, the stKFCM shows better results than KPC on the 2D50 data set, equaling the performance of KFCM down to the 2% chunk size. Furthermore, stKFCM is much faster than KPC at small data chunk sizes for 2D15 and 2D50. This is because the KFCM iterations at each chunk converge faster with the stKFCM algorithm. Note that KPC is faster than stKFCM for larger chunks; this is because of the inverse calculation at Line 3 of stKFCM. Both KPC and stKFCM produce very good speedup over KFCM at small data chunk sizes, while still producing partitions nearly equivalent to the literal KFCM.

All three fuzzy clustering algorithms produce partitions that do not match well to the ground-truth for the MNIST and Forest data sets (which is also the case of *k*-means and other similar crisp partitioning algorithms). This does not alarm us as there is a big difference between classification and clustering results; i.e., classification uses user-supplied

Table 5.1
Clustering Results*

	KFCM				**KPC** [5]			**stKFCM**		
Data set	**Purity**	**ARI**	**Time (secs)**	n	**Purity**	**ARI**	**Time (secs)**	**Purity**	**ARI**	**Time (secs)**
				50%	0.94 (0.03)	0.91 (0.05)	4.0	0.94 (0.04)	0.91 (0.06)	30
2D15				25%	0.94 (0.04)	0.91 (0.05)	2.0	0.93 (0.04)	0.90 (0.05)	9.3
$N = 5,000$	0.95	0.91	5.5	10%	0.94 (0.04)	0.91 (0.05)	1.05	0.93 (0.04)	0.89 (0.05)	1.6
$c = 15$	(0.04)	(0.05)		5%	0.93 (0.04)	0.90 (0.06)	0.77	0.92 (0.04)	0.88 (0.05)	0.73
$d = 2$				2%	0.91 (0.04)	0.89 (0.05)	1.1	0.92 (0.04)	0.89 (0.05)	0.42
				1%	0.88 (0.05)	0.85 (0.07)	1.2	0.88 (0.18)	0.85 (0.19)	0.09
				50%	0.85 (0.03)	0.84 (0.04)	31	0.88 (0.03)	0.83 (0.04)	126
2D50				25%	0.88 (0.03)	0.82 (0.04)	15	0.88 (0.03)	0.82 (0.04)	36
$N = 7,500$	0.92	0.84	56	10%	0.87 (0.03)	0.82 (0.03)	8.8	0.87 (0.03)	0.82 (0.04)	6.8
$c = 50$	(0.02)	(0.04)		5%	0.85 (0.03)	0.79 (0.03)	6.2	0.88 (0.03)	0.84 (0.04)	2.9
$d = 2$				2%	0.82 (0.03)	0.75 (0.04)	5.8	0.88 (0.02)	0.84 (0.03)	1.7
				1%	0.79 (0.03)	0.72 (0.04)	6.8	0.79 (0.27)	0.77 (0.24)	6.2
				10%	0.20 (0.01)	0.038 (0.0007)	689	0.19 (0.03)	0.037 (0.0013)	4488
				5%	0.20 (0.01)	0.039 (0.0040)	112	0.27 (0.02)	0.035 (0.015)	1460
MNIST				2%	0.20 (0.01)	0.038 (0.0057)	46	0.26 (0.02)	0.037 (0.015)	415
$N = 70,000$	0.20	0.027	**	1%	0.20 (0.02)	0.036 (0.0103)	24	0.23 (0.01)	0.044 (0.0121)	193
$c = 10$	(0.01)	(0.00)		0.5%	0.20 (0.02)	0.033 (0.011)	14	0.20 (0.01)	0.0401 (0.0107)	69
$d = 784$				0.2%	0.16 (0.01)	0.023 (0.013)	14	0.23 (0.01)	0.047 (0.0094)	14
				0.1%	0.18 (0.02)	0.016 (0.0103)	19	0.24 (0.01)	0.045 (0.0007)	15
				0.05%	0.18 (0.02)	0.011 (0.0067)	33	0.23 (0.02)	0.043 (0.0098)	94
				0.02%	0.16 (0.01)	0.0051 (0.0037)	61	0.22 (0.02)	0.033 (0.0105)	60
Forest				0.2%	0.52 (0.01)	0.0014 (0.022)	121	0.51 (0.02)	0.019 (0.019)	1122
$N = 581,012$	0.52	0.03	**	0.1%	0.52 (0.01)	0.0010 (0.025)	81	0.51 (0.01)	0.017 (0.019)	333
$c = 7$	(0.03)	(0.03)		0.05%	0.51 (0.01)	0.0078 (0.024)	60	0.50 (0.01)	0.015 (0.027)	118
$d = 54$				0.02%	0.52 (0.03)	0.0089 (0.023)	57	0.49 (0.00)	0.0011 (0.0006)	43
				0.01%	0.51 (0.02)	0.0018 (0.0087)	65	0.49 (0.00)	0.00 (0.00)	29

*Mean and standard deviation over 100 independent trials. **Timing information inappropriate for MNIST and Forest data as these experiments were performed on a high-performance computing cluster.

labels (a.k.a. ground truth), while clustering aims to find natural groupings. Hence, these results simply tell us that the natural groupings in these two data sets (as produced by c-means partitioning) do not match well to the ground truth labels. The aim of our proposed algorithm is to approximate the partitions of the KFCM for large data sets. And both KPC and stKFCM succeed at that for the MNIST data. The stKFCM algorithm exceeds the performance of KFCM for all chunk sizes, except for 0.02%, while KPC meets or exceeds the KFCM performance for chunk sizes $> 0.5\%$; clearly, the stKFCM outperforms the KPC algorithm for the MNIST data. We have seen this behavior, i.e., the sampled algorithm exceeding the performance of the literal algorithm, in other studies [12, 20] and attribute it to the influence of outliers or noise on the literal algorithm. This hypothesis has not been proved.

On the Forest data, both KPC and stKFCM struggle to match the performance of KFCM (run on the entire data set) in terms of the ARI criteria; however, in terms of Purity both KPC and stKFCM perform fairly, with KPC having a slight edge here. We believe that this is caused by the fact that the classes in the Forest data have very unbalanced numbers of samples; two of the seven classes, Spruce-Fir and Lodgepole Pine, comprise greater than 85% of the data set. Hence, the KPC and stKFCM algorithms, which sequentially operate on small samples of the data set, can be presented with samples that are comprised mostly of these two classes.

Overall, these results are very pleasing as they show that stKFCM, even with very small data chunk sizes, achieves clustering performance near to that of KFCM run on the entire data set. Furthermore, the projection method used in stKFCM shows better performance than the medoid method used by KPC, even showing better run-time for some data sets.

Chapter 6

Conclusion

Big data analysis has become very pertinent. It creates many wonderful opportunities but is accompanied by numerous big challenges. As digital devices pervade everyday life and generate petabyte scale datasets, both memory space and computational resources are important considerations in real life applications. Algorithms that could reconcile these requirements are highly desired.

Kernel Fuzzy c-Means is nearly infeasible with Big Data since both its computational complexity and space requirement are quadratic. Hence, computers to-date are incapable of using KFCM with Big Data. Challenges in exploring Big Data arise not only from their sheer quantity but also from inaccessable future data in real-time applications. In this paper, we proposed the stKFCM algorithm that significantly reduces both the memory requirement and computational complexity for performing KFCM clustering. By splitting data into sequencial pieces, we effectively reduce the space complexity from $O(N^2)$ to $O(n^2)$, where n is the size of a small data chunk at each time step. Smaller size of data also accelerate the convergence of the algorithm (as long as n is not too small). Empirical results show that stKFCM achieves accurate results while only requiring access to a very small portion of the kernel matrix.

In the future, we will examine how we can better represent and pass on the information from previous data chunks, including using hybrids of random sampling and projection methods. We will also look at other methods of projection, with a focus on overall clustering performance. We plan to further improve the speed and efficiency of our algorithm by investigating how it can be deployed on multicore processors, GPUs, and cloud-based architectures, taking advantage of massively parallel computing.

Bibliography

[1] M. Schönberger and K. Cukier, *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. Eamon Dolan/Houghton Mifflin Harcourt, 2013.

[2] J. Ginsberg, M. H. Mohebbi, R. S. Patel, L. Brammer, M. S. Smolinski, and L. Brilliant, "Detecting influenza epidemics using search engine query data," *Nature*, 2009.

[3] T. W. E. Forum, "Personal data: The emergence of a new asset class," 2012.

[4] Facebook, Ericsson, and Qualcomm, "A focus on efficiency: A whitepaper from facebook, ericsson and qualcomm," 2013.

[5] S. Fausser and F. Schwenker, "Clustering large datasets with kernel methods," in *Proc. Int. Conf. Pattern Recognition*, Nov. 2012, pp. 501–504.

[6] J. S. Tayler and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, UK, 2004.

[7] P. Hore, L. O. Hall, and D. B. Goldgof, “Single pass fuzzy c means,” in *Fuzzy Systems Conferences*, 2007.

[8] P. Hore, D. B. Goldgof, and L. O. Hall, “Creating streaming iterative soft clustering algorithms,” in *NAFIPS ’07 Annual Meeting of the North American*, 2007.

[9] P. Hore, “A fuzzy c means variant for clustering evolving data streams,” in *IEEE International Conference on Systems, Man and Cybernetics*, 2007.

[10] S. Eschrich, J. Ke, L. O. Hall, and D. B. Goldgof, “Fast accurate fuzzy clustering through data reduction,” *IEEE Trans. Fuzzy Systems*, vol. 11, pp. 262–269, April 2003.

[11] T. Havens, J. C. Bezdek, C. Leckie, L. O. Hall, and M. Palaniswami, “Fuzzy c-means algorithms for very large data,” *IEEE Trans. Fuzzy Systems*, vol. 20, no. 6, pp. 1130–1146, 2012.

[12] 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.

[13] P.Hore, L. O. Hall, D. B. Goldgof, and W. Cheng, “Online fuzzy c means,” in *NAFIPS*, 2008.

[14] Z. Zhang and T. Havens, “Scalable approximation of kernel fuzzy c-means,” in *IEEE International Conference on Big Data*, 2013, pp. 161–168.

[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] F. Provost, D. Jensen, and T. Oates, “Efficient progressive sampling,” *Fifth KDDM, ACM Press*, no. 23-32, 1999.

[17] R. J. Hathaway and J. C. Bezdek, “Extending fuzzy and probabilistic clustering to very large data sets,” *Computation Statistics Data Analysis*, 2006.

[18] J. C. Bezdek, R. J. Hathaway, J. Huband, C. Leckie, and R. Kotagiri, “Approximate clustering in very large relational data,” *International Jounal of Intelligent Systems*, vol. 21, pp. 817–841, 2006.

[19] L. Wang, J. C. Bezdek, C. Leckie, and R. Kotagiri, “Selective sampling for approximate clustering of very large data sets,” *International Jounal of Intelligent Systems*, vol. 23, pp. 313–331, 2008.

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

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

[22] 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. 101–144, 1995.

[23] P. S. Bradley, U. Fayyad, and C. Reina, "Scaling clustering algorithms to large databases," in *the Fourth International Conference on Knowledge Discovery and Data Mining*, 2000, pp. 51–57.

[24] F. Farnstrom, J. Lewis, and C. Elkan, "Scalability for clustering algorithms revisited," *ACM SIGKDD Explorations*, vol. 2, pp. 51–57, 2000.

[25] C. Gupta and R. Grossman, "Genic: A single pass generalized incremental algorithm for clustering," in *SIAM Int. Conf on Data Mining*, 2004.

[26] P. Hore, L. O. Hall, and D. B. Goldgof, "A cluster ensemble framework for large data sets," in *Systems, Man and Cybernetics*, 2006.

[27] 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, Jun 2003.

[28] N. Ailon, R. Jaiswal, and C. Monteleoni, "Streaming k-means approximation," in *NIPS*, 2009.

[29] B. Bahmani, B. Mosesley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *VLDB Endowment*, vol. 5, no. 7, pp. 622–633, 2012.

[30] T. Zhang, R. Ramakirshnan, and M. Livny, “Birch: An efficient data clustering method for very large databases,” in *ACM SIGMOD Int Conf. Manag. Data*, 1996, pp. 103–144.

[31] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley and Sons, 1990.

[32] R. Ng and J. Han, “Clarans: A methods for clustering objects for spatial data mining,” *IEEE Trans. Knowl. Data Eng*, vol. 14, no. 5, pp. 1003–1016, Oct 2002.

[33] T. C. Havens, J. C. Bezdek, and M. Palaniswami, *Computational Intelligence: Revised and Selected Papers from IJCCI 2010*. Berlin: Springer, 2012, vol. 399, ch. Incremental Kernel Fuzzy c-Means, pp. 3–18.

[34] Z. Wu, W. Xie, and J. Yu, “Fuzzy c-means clustering algorithm based on kernel method,” in *Proc. Int. Conf. Computational Intelligence and Multimedia Applications*, Sept. 2003, pp. 49–54.

[35] R. Hathaway, J. Davenport, and J. C. Bezdek, “Relational duals of the c-means clustering algorithms,” *Pattern Recognition*, vol. 22, no. 2, pp. 205–212, 1989.

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

[37] T. Havens, “Approximation of kernel k means for streaming data,” in *21st International Conference on Pattern Recognition*, 2012.

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

[39] R. Jin, T. Yang, Y. F. Li, and Z. H. Zhou, “Improved bounds for the nystrom method with application to kernel classification,” *IEEE Trans. Info. Theory*, vol. 10.1109/TIT.2013.2271378, 2013.

[40] L. Hubert and P. Arabie, “Comparing partition,” *J. Class*, vol. 2, pp. 193–218, 1985.