DIGITAL COMMONS
@ UNIVERSITY OF SOUTH FLORIDA

University of South Florida
Digital Commons @ University of South Florida

USF Tampa Graduate Theses and Dissertations

USF Graduate Theses and Dissertations

2007

Scalable frameworks and algorithms for cluster ensembles and clustering data streams

Prodip Hore
University of South Florida

Follow this and additional works at: https://digitalcommons.usf.edu/etd

 Part of the American Studies Commons

Scholar Commons Citation

Hore, Prodip, "Scalable frameworks and algorithms for cluster ensembles and clustering data streams" (2007). *USF Tampa Graduate Theses and Dissertations.*
https://digitalcommons.usf.edu/etd/2222

This Dissertation is brought to you for free and open access by the USF Graduate Theses and Dissertations at Digital Commons @ University of South Florida. It has been accepted for inclusion in USF Tampa Graduate Theses and Dissertations by an authorized administrator of Digital Commons @ University of South Florida. For more information, please contact digitalcommons@usf.edu.

Scalable Frameworks and Algorithms for Cluster Ensembles and Clustering Data Streams

by

Prodip Hore

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Lawrence O. Hall, Ph.D.
Dmitry B. Goldgof, Ph.D.
Rafael A. Perez, Ph.D.
Sudeep Sarkar, Ph.D.
Yuncheng You, Ph.D.

Date of Approval:
June 12, 2007

Keywords: partitioning, hard-c-means, fuzzy-c-means, scalability, merging, streaming

© Copyright 2007, Prodip Hore

DEDICATION

To my family

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SCALABLE FRAMEWORKS AND ALGORITHMS FOR CLUSTER ENSEMBLES AND CLUSTERING DATA STREAMS

Prodip Hore

ABSTRACT

Clustering algorithms are an important tool for data mining and data analysis purposes. Clustering algorithms fall under the category of unsupervised learning algorithms, which can group patterns without an external teacher or labels using some kind of similarity metric. Clustering algorithms are generally iterative in nature and computationally intensive. They will have disk accesses in every iteration for data sets larger than memory, making the algorithms unacceptably slow. Data could be processed in chunks, which fit into memory, to provide a scalable framework. Multiple processors may be used to process chunks in parallel. Clustering solutions from each chunk together form an ensemble and can be merged to provide a global solution. So, merging multiple clustering solutions, an ensemble, is important for providing a scalable framework. Combining multiple clustering solutions or partitions, is also important for obtaining a robust clustering solution, merging distributed clustering solutions, and providing a knowledge reuse and privacy preserving data mining framework. Here we address combining multiple clustering solutions in a scalable framework. We also propose algorithms for incrementally clustering large or very large data sets. We propose an algorithm that can cluster large data sets through a single pass. This algorithm is also extended to handle clustering infinite data streams. These types of incremental/online algorithms can be used for real time processing as they

don't revisit data and are capable of processing data streams under the constraint of limited buffer size and computational time. Thus, different frameworks/algorithms have been proposed to address scalability issues in different settings. To our knowledge we are the first to introduce scalable algorithms for merging cluster ensembles, in terms of time and space complexity, on large real world data sets. We are also the first to introduce single pass and streaming variants of the fuzzy c means algorithm. We have evaluated the performance of our proposed frameworks/algorithms both on artificial and large real world data sets. A comparison of our algorithms with other relevant algorithms is discussed. These comparisons show the scalability and effectiveness of the partitions created by these new algorithms.

CHAPTER 1

INTRODUCTION AND BACKGROUND

In this work, different frameworks/algorithms have been proposed to address scalability issues in appropriate settings. Cluster ensembles mainly deal with merging multiple clustering solutions in an efficient way, single pass algorithms address clustering large or very large data sets in a single scan, and streaming algorithms deals with clustering potentially infinite data streams, where data might evolve with time and may come in any order. We will start our discussion by focussing on providing scalable frameworks/algorithms for cluster ensembles and then discuss clustering large data sets that do not fit in a typical computer memory through an incremental clustering algorithm. Finally, we will discuss clustering potentially infinite data streams. Background on the classical clustering algorithms will follow.

1.1 Introduction

Clustering algorithms are widely used for data analysis purposes. Clustering algorithms partition unlabeled data into groups based on some similarity metric. Clustering data is often considered to be a slow process because data often must be partitioned multiple times to find an optimal partition. Due to the advancement of hardware technology, events/data can now be recorded more frequently. So, larger and larger unlabeled data sets, that will not fit into the typical computer memory, are becoming available. Generally soft clustering algorithms are slower and more resource consuming than crisp clustering algorithms. The memory requirements for

soft clustering algorithms are generally greater than the size of data. This is because they need to keep additional data structures in memory. Data could be processed in chunks, which fit into memory, to provide a scalable framework. Clustering solutions from each chunk will form an ensemble of clustering solutions, which can be merged to provide a final clustering solution. The chunks can also be clustered in parallel using multiple processors. So, merging multiple clustering solutions, an ensemble, is important for providing a scalable framework. Data could also be naturally distributed and there might be cost and privacy constraints in centrally pooling and sharing data. In these kind of scenarios, clustering solutions of individual sites have to be merged without accessing the data from which it was obtained (a knowledge reuse framework). Most clustering algorithms are sensitive to initializations, thus merging multiple clustering solutions of the same data set with different initializations may also provide robust clustering solution. So, combining multiple clustering solutions or partitions is important for obtaining a robust clustering solution, merging distributed clustering solutions, providing a knowledge reuse and privacy preserving data mining framework, and scalability. Besides providing scalable algorithms/framework for a cluster ensemble, there is the necessity to provide frameworks/algorithms for clustering unloadable data sets on disk incrementally and clustering potentially infinite data streams. These kind of algorithms can be used for real time processing as they do not revisit and process data in the order they come. Large amounts of streaming data are becoming available from everyday transactions, web click streams, telephone records, web documents, sensors data, environmental data, network monitoring, etc. Clustering streaming data is challenging because it has to processed as it comes and might evolve with time. For processing streaming data, there will be constraint on the amount of memory and processing time used.

In this work, we will propose scalable algorithms/frameworks for cluster ensembles, algorithms for clustering data in a single pass through unloadable data on disk (using limited memory allocated), and about clustering infinite streaming data. To make the discussion of each framework/algorithm complete in its context, they have been provided in different chapters. Results from applying an algorithm are also included in the appropriate chapter.

Many cluster ensemble merging algorithms have been proposed, but we propose scalable cluster ensemble merging algorithms, in terms of time and space complexity, on large real world data sets. We are also the first to introduce single pass and streaming variants of the fuzzy c means algorithms. We will provide mathematical equations which would enable other iterative fuzzy/possibilistic clustering algorithms to be turned into similar algorithms as we did for the fuzzy c means.

1.1.1 Cluster Ensembles

Combining multiple clustering solutions or partitions (a cluster ensemble) is important for obtaining a robust clustering solution, merging distributed clustering solutions, providing a knowledge reuse and privacy preserving data mining framework, and scalability. Although, many multiple partition combining algorithms have been proposed, not much work has been done on scalability issues in terms of time and memory complexity. In this work, combining multiple clustering solutions within a scalable and robust framework for large data sets is addressed. We have compared our algorithms with a relevant ensemble merging algorithm and will show that our algorithms are competitive or better, while providing very large speed up (of the order of hundreds of thousand times) on medium to large data sets. A quantitative evaluation showing the quality of the ensemble merging algorithms compared to clustering all the data at once in memory and averaging base clustering solutions in the

ensemble on seven real data sets will be provided. In addition, ensemble merging algorithms are also compared to a single pass hard-k-means algorithm to point out the strengths and limitations of both frameworks.

1.1.2 Single Pass Algorithms

Large or very large data sets are becoming increasingly difficult to fit in memory. Moreover, the memory requirement of clustering algorithms, generally soft clustering, are many times the size of the data. If the space requirement of clustering algorithms is greater than available memory, disk accesses in every iteration of the clustering algorithm will make it unacceptably slow. Recently several algorithms for clustering large data sets or streaming data sets have been proposed. Most of them address the crisp case of clustering, which is not necessarily easily generalized to the fuzzy case. In this work, we propose a simple single pass (through the data) fuzzy c means algorithm that neither uses any complicated data structure nor any complicated data compression techniques, yet produces data partitions comparable to fuzzy c means. We also show our simple single pass fuzzy c means clustering algorithm when compared to fuzzy c means produces excellent speed-ups in clustering and thus can be used even if the data can be fully loaded in memory. Experimental results using nine real data sets will be provided.

1.1.3 Streaming Algorithms

Clustering algorithms for streaming data sets are gaining importance due to the availability of large data streams from different sources. Single pass algorithms may be used for scaling clustering algorithms to large data sets, but clustering data streams may not be viewed as a single pass clustering problem because they are generally blind to an evolving distribution as a single pass algorithm over an entire stream

will be dominated by outdated history as we shall see. So, we provide an enhanced algorithm for clustering stream data. Recently a number of streaming algorithms have been proposed using crisp algorithms such as hard c means or its variants. As stated earlier, the crisp cases may not be easily generalized to fuzzy cases as these two groups of algorithms try to optimize different objective functions. In this work we propose a streaming variant of the fuzzy c means (FCM) algorithm. At any stage during processing, a good streaming algorithm should be able to summarize data seen so far and also respond to evolving distributions. We study the tradeoff involved between summarization of data seen and response to an evolving distribution by varying the amount of history used by a streaming algorithm. Empirical evaluation of the performance of our algorithm using both artificial and real data sets under a noisy setting will show its effectiveness. Another variant of the streaming algorithm has also been proposed to get cluster quality, at the end of the stream, as good as clustering with the full data set. This kind of algorithm will be useful to partition an unloadable data set via a streaming approach. One advantage of a streaming algorithm over many scalable and single pass algorithms is that it will process data as it comes.

1.2 Clustering Algorithm Background

Clustering is one of the most important unsupervised learning solutions. In unsupervised learning the data has no predefined label. Clustering could be viewed as the "process of organizing objects into groups whose members are similar in some way". Objects or examples or patterns are defined by features, which describe them. Clustering algorithms group these examples into clusters such that examples in a cluster are more similar than examples in other clusters. There are many varieties of clustering algorithms but generally most of them fall under one of the two broad cat-

egories i.e. hierarchical or partitional algorithms. Below (Figure1.1) is an incomplete taxonomy of clustering algorithms. There are also many other varieties of clustering algorithms [1] but we show the most commonly used ones.

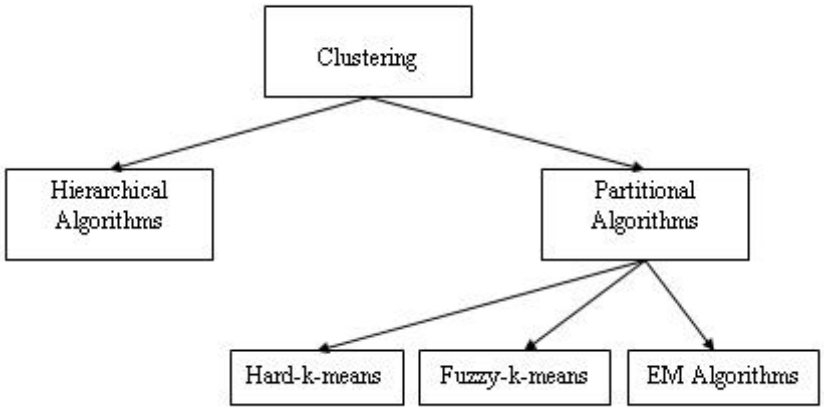


Figure 1.1 Taxonomy of Clustering Algorithms.

Hierarchical clustering algorithms do not form a final partition of the data but yield a dendrogram representing the nested grouping of patterns and similarity levels at which groupings change [1].

Illustrative example:

Consider the following 5 examples E1, E2, E3, E4, and E5 in 2-dimensional space (Figure 1.2) and let's assume that the spatial proximity among them is a measure of the similarity with each other.

Applying a hierarchical algorithm to the above data set could yield a dendrogram as shown in Figure 1.3.

We will not further discuss hierarchical algorithms because our work is based on providing scalable frameworks/algorithms for partitional algorithms. Partitional clustering algorithms produce a single partition of the data by iteratively optimizing a clustering criterion function or an objective function [1]. Hard k-means results in

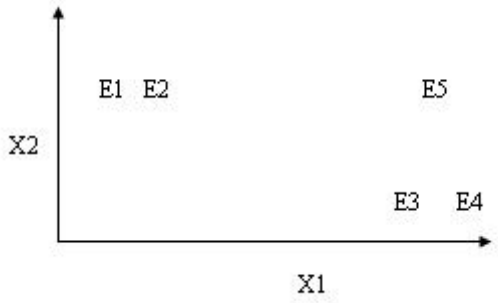


Figure 1.2 Example of data set.

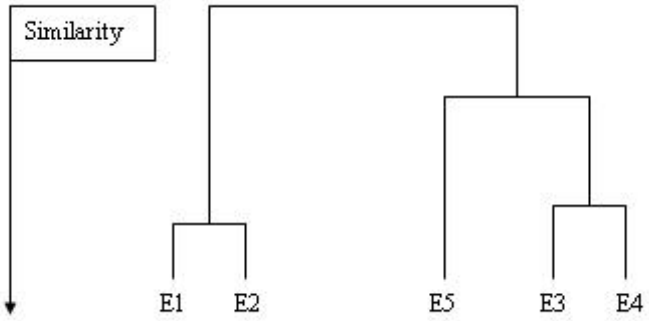


Figure 1.3 Example of Hierarchical Clustering.

a crisp partition of the data while fuzzy k-means assigns a degree of membership to each example in a data set i.e. how much membership an example has in a cluster. Membership for a particular example in a cluster indicates how well it fits in that cluster. Applying hard-k-means to the data set (Figure 1.2) could yield the following two clusters (Figure 1.4). It should be noted hard-k-means and fuzzy-k-means are the same as hard-c-means and fuzzy-c-means algorithms respectively, where k or c indicate the number of clusters.

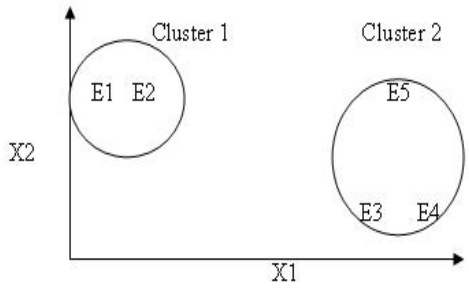


Figure 1.4 Example partition from Hard-Clustering.

Applying fuzzy-k-means to the data set (Figure 1.2) could yield the following two overlapping clusters (Figure 1.5). Here the example E5 belongs to both the clusters but its membership degree would be higher for cluster 2 than in cluster 1.

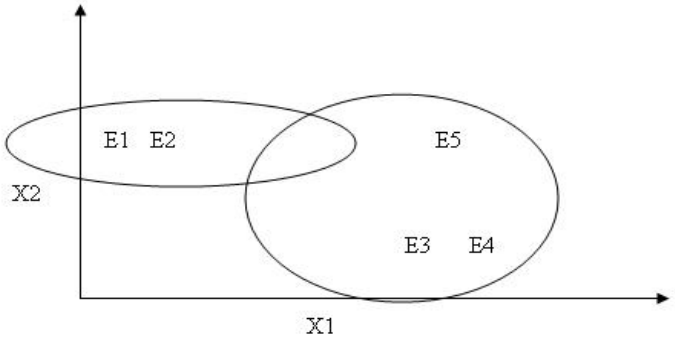


Figure 1.5 Example partition from Fuzzy Clusters.

Similarity measures among examples or patterns are of fundamental importance and must be chosen carefully [1]. We have used a well-known metric, the Euclidean distance, for measuring similarity among examples whose features are all continuous values. If every example or pattern has dimension d i.e. d features, then similarity among any examples x_i and x_j is measured as follows:

$$d_{i,j}(x_i, x_j) = \left(\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2 \right)^{\frac{1}{2}} = \|x_i - x_j\|_2$$

The widely used criterion function or objective function which hard-c-means and fuzzy-c-means try to optimize is the squared error function:
$\min_{(U,V)} \{J_m(U,V) = \sum_{i=1}^{c} \sum_{k=1}^{n} u_{ik}^m D_{ik}(x_k, v_i)\}$, where U contains the cluster memberships with $u_{ik} \in [0,1]$, $m > 1$ if fuzzy or $u_{ik} \in \{0,1\}$ if hard; $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_c) \in R^d$, $\mathbf{v}_i$ specifies the i^{th} cluster center of dimension d; m ≥ 1 is a weighting exponent that controls the degree of fuzzification of U ; and $D_{ik}(\mathbf{x}_k, \mathbf{v}_i) = D_{ik}$ is the deviation of $\mathbf{x}_k$ from the i^{th} cluster prototype (distance metric).

We now describe the Hard-c-means and Fuzzy-c-means algorithms using the Euclidean distance metric in Algorithm 1. and Algorithm 2. respectively.

Algorithm 1. Hard-c-means Algorithm.

Input: Data set of n examples of s features and the value of c (number of clusters).

Output: Partition of the input data into c groups or clusters.

1. Declare an n x c size U membership matrix.

2. Randomly generate c cluster center locations within the range of the data or randomly select c examples as initial cluster centroids. Let the centroids be v_1, v_2,..., v_c.

3. Calculate the distance measure $d_{i,j} = \|x_i - v_j\|_2$ (Euclidean distance), for all cluster centroids j=1 to c and data examples 1 to n.

4. Compute the U membership matrix as follows:

$$u_{ik} = \left\{ \begin{array}{ll} 1; & \|\mathbf{x}_k - \mathbf{v}_i\|_2 \leq \|\mathbf{x}_k - \mathbf{v}_j\|_2, j \neq i \\ 0; & otherwise \end{array} \right\} \quad \forall \text{ i, j ; (ties are broken randomly);}$$

5. Compute new cluster centroids $v_j = \frac{\sum_{i=1}^{n} (u_{i,j}) x_i}{\sum_{i=1}^{n} (u_{i,j})}$, for j=1 to c.

   Note the cluster centers in hard-c-means are just the centroids of the points in a cluster.

6. Repeat steps 3 to 5 until the change in U in two successive iterations is less than a given threshold.

Algorithm 2. Fuzzy-c-means Algorithm.

Input: Data set of n examples, value of c (number of clusters), fuzzification value m>1

Output: Partition of the input data into c groups or clusters.

1. Declare an n x c size U membership matrix.

2. Randomly generate c cluster center locations within the range of the data or randomly select c examples as initial cluster centroids. Let the centroids be v_1, v_2,..., v_c.

3. Calculate the distance measure $d_{i,j} = \|x_i - c_j\|_2$ (Euclidean distance), for all cluster centroids j=1 to k and data examples 1 to n.

4. Compute the Fuzzy membership matrix as follows: $u_{i,j} = \left[\sum_{l=1}^{k} \left(\frac{d_{i,j}}{d_{i,l}}\right)^{\frac{2}{m-1}}\right]^{-1}$ if $d_{i,j} > 0$

   $u_{i,j} = 1$ if $d_{i,j} = 0$

5. Compute new cluster centroids $v_j = \frac{\sum_{i=1}^{n} (u_{i,j})^m x_i}{\sum_{i=1}^{n} (u_{i,j})^m}$, for j=1 to c.

6. Repeat step 3 to 5 until the change in U in two successive iterations is less than a given threshold.

Although, we have not used the EM clustering algorithm in this work, it is a highly used probabilistic clustering approach. Any partitioning algorithm whose clusters can be represented by centroids can use our scaling methods to cluster large-scale data. The EM algorithm makes a hypothesis that the pattern or examples of a data set are drawn from some distribution and proceeds with the goal to identify them by iteratively updating the hypothesis. The most commonly used assumption of the distribution is Gaussian.

Chapter 2 will discuss motivation and related work. Chapter 3 discusses the scalable framework for cluster ensembles. The single pass algorithm will be discussed in Chapter 4. In Chapter 5, a clustering algorithm for streaming data will be given. In this chapter, another variant of the streaming algorithm will be proposed to provide

clustering quality as good as clustering the full stream at once. Finally, we will end with conclusions in Chapter 6.

CHAPTER 2

MOTIVATION AND RELATED WORK

In this chapter, we provide the motivation and related work for the various scalable frameworks/algorithms proposed. The central idea of all the algorithms/frameworks are to provide scalable approaches in their respective application domain.

2.1 Cluster Ensembles

Cluster Ensembles are used to combining multiple clustering solutions or partitions (an ensemble). Combining an ensemble is important for obtaining a robust clustering solution, merging distributed clustering solutions, providing a knowledge reuse and privacy preserving data mining framework, and scalability. A clustering solution in an ensemble can be created in many ways; one may create it from a partial view of a data, each is a "weak" clustering solution then. They may also be created from the full data set with different random initializations and may also be produced using different clustering algorithms.

2.1.1 Motivation

There has been research on combining ensembles of clustering solutions [2–13] where each clustering solution is either represented by a label vector or a co-association matrix or a similar representation, whose size is of the order of the number of examples from which the solution was obtained. It has been shown that combining an ensemble of clustering solutions produces a good final global partition. We will

denote these as high resolution representations because they involve a description for each example in a data set. For example, a data set of n examples will have n labels in its label vector, one for each example. A label vector is a representation of a clustering solution, which says which examples belong to a same label/group.

A co-association matrix and similar high resolution representations are generally created from label vectors. However, creating and combining multiple clustering solutions represented by a high resolution representation could pose a scalability problem in terms of memory and time complexity for large data sets [14]. In [6], it has been pointed out that existing merging algorithms suffer from a time complexity problem. In most of the approaches using high resolution representations, each clustering solution in the ensemble will have n in its time and memory complexity equations, where n is the number of examples. In [12, 13], to speed up their multiple partition combining algorithm they used a sample of the original data set and then extended the results to global data. However, for large or extreme data sets even a subsample may be quite large.

In this work, we address the scalability issues by using a kind of low resolution representation of clustering solutions. Further speed up is possible by using a subsample of the original data set. Moreover, a high resolution representation of clustering solutions in the ensemble poses a problem for merging clustering solutions for object distributed data as there will be no overlap of examples. In [2], it has been pointed out that combining clustering solutions from object distributed data is more difficult than feature distributed as there is no overlap between the arbitrary labels created from disjoint data sets. This could be an obstacle to scalability as clustering the data distributively is a method of scaling. We believe that for large data sets the representation of a cluster ensemble should be small so that it can be kept in memory and be merged efficiently, while providing a robust and good quality final clustering

solution. The merging algorithm should also be able to merge clustering solutions obtained from a set of distributed examples, so as to provide a way of scaling clustering algorithms to very large data sets.

In this work, we propose a framework for creating and merging an ensemble of multiple clustering solutions, which is scalable in terms of time and memory complexity. In our approach, multiple clustering solutions (an ensemble) are created from disjoint data sets, where each clustering solution in the ensemble is represented by its centroids. A centroid is a low resolution representation of a clustering solution as it does not require a description for each example in the data set, and the number of clusters in a data set is generally much smaller than the number of examples. Creating the ensemble of centroids from disjoint data sets also puts a restriction on the size of the ensemble i.e. if we divide a data set of 1000 examples randomly into 10 equal subsets then there can be a maximum number of 10 sets of centroids from each subset in the ensemble. In comparison to label vectors or other similar resolution representations, the centroids are smaller in size as the number of clusters is typically much smaller than number of examples in the data. Thus an ensemble formed by centroids can be kept in memory and merged efficiently. For a data set of very large dimension, keeping centroids in memory might become less efficient. However, in most cases the size of centroid vectors is not likely to be very large.

Now, the problem of finding a final clustering solution using the ensemble of centroids, a low resolution representation, can be viewed as the problem of reaching a global consensus on the position of cluster centroids for the final clustering solution. A global consensus on the position of cluster centroids is formed by partitioning them into consensus chains/groups and computing the weighted mean of centroids in each consensus chain/group to represent a global cluster centroid. A final partition in terms of label vectors may be obtained by computing the Euclidean distance of each

example from the final set of centroids, and assigning it to the nearest centroid. Thus, the set of global centroids can be used to partition the data.

As in [2] for producing the final clustering solution, we don't need access to the feature values of the examples. Hence, this approach also simultaneously provides a knowledge reuse and privacy preserving data mining framework. Recently, knowledge reuse and privacy preserving frameworks have become important because they enable merging of already clustered solutions without accessing the feature vectors of the data. This type of requirement might arise occasionally, where data might be geographically distributed and the owners of data are not willing to share records due to privacy or proprietary issues. In these scenarios, multiple partition combining algorithms can be used.

Our approach can also be used for scaling clustering algorithms because disjoint subsets of data can be clustered in parallel. Even in the case that only one machine is available; data can still be sequentially clustered in such a way that each data subset fits in memory and the ensemble of clustering solutions can be merged at the end to give a final partition.

The main focus of this paper is to show how an ensemble of centroids created from object distributed data, created here by randomly dividing the full data set, can be merged in a scalable framework. We have proposed two methods to merge ensembles of centroids. We will show a robust and good final clustering solution can be obtained when compared to a relevant multiple partition combining algorithm, while providing superiority in terms of time and space complexity. We have also compared the quality of our algorithms with a scalable single pass clustering algorithm. The quality of ensemble merging algorithms are compared to the quality of clustering all the data at once in memory and the average base clustering solutions in the ensemble. Time and memory complexity of the ensemble merging algorithms will also be discussed.

2.1.2 Related Work

Creating and combining multiple partitions of a given data set has been examined in several ways [2–15]. In [2] hyperedges are formed from each clustering solution represented as a label vector. Using these hyperedges a final partition was obtained by using graph partitioning algorithms like Cluster-based Similarity Partitioning (CSPA), Hypergraph Partitioning (HGPA), and Meta Clustering (MCLA). In [3] clustering solutions in the ensemble are represented by membership matrices of size n by k, where n is the number of examples in the data set and k is the number of clusters. A final clustering solution is then obtained by soft correspondence, where each cluster from one partition corresponds to clusters in other partitions with a different weight. In [8, 10], weak clusters of a data set are formed by using random hyperplanes and multiple views of a sample of the data. By creating many views of the data and merging them, they were able to show the final partitions were better. However, the representation of base clusterings were in the form of label vectors, and the approach might be complicated for large data sets because a large number of partitions might have to be created to get a good overall partition. In [9, 16] clusters have been merged using co-association matrices, but the approach is computationally expensive. In [5, 7, 17] combining clustering solutions from multiple bootstrap samples was discussed. In [18], a method for fast and efficient ensemble clustering was proposed using the Leader algorithm [19]. However, both the ensemble creation and merging algorithm use parameters particular to the Leader algorithm, and the method may not generalize to ensembles created from the k means family or other clustering algorithms.

As in our framework, disjoint subsets of data can be clustered in parallel and used independently or merged to allow clustering to scale for large data sets, so

we did a survey of scaling algorithms. In recent years a number of new clustering algorithms have been introduced to address the issue of scalability [20–27]. All the algorithms assume that the clustering algorithm is applied to all the data centrally pooled in a single location. Various methods for expediting fuzzy-k-means have been explored. In [28] data sub sampling was proposed and a good speed up of fuzzy-k-means was obtained. Similarly, in [29] a multistage random sampling algorithm was proposed to speed-up fuzzy-k-means. In [30], a density based data reduction method has been discussed, which condenses a data set into fewer examples. Recently two online clustering algorithms have been proposed in [31]. A parallel implementation of k-means [32] or other similar algorithms is not quite applicable to cluster ensembles because they do not combine multiple clusterings but provide a way of clustering a data set using multiple processors, where synchronized communication during each iteration is required.

In [33,34] distributed clustering has been discussed under limited knowledge sharing. In [34] a clustering solution has been represented by labels while in [33] generative or probabilistic model parameters are sent to a central site, where "virtual samples" are generated and clustered using an EM algorithm [1] to obtain the global model. In [35], local sites are first clustered using the DBSCAN algorithm and then representatives from each local site are sent to a central site, where DBSCAN is applied again to find a global model. Another density estimation based distributed clustering algorithm has been discussed in [36]. In [37], local models or prototypes are detected using EM [1] in a distributed environment and later merged by computing mutual support among the Gaussian models. In [38] a method for speeding up the EM algorithm has been discussed by aggregating subclusters obtained from a data set. Merging local models or prototypes in a distributed environment has also been discussed in our earlier work [14,39].

In this work, we address the merging of multiple partitions formed from the widely used fuzzy k means and hard k means algorithms in a scalable framework; however, it can be extended to probabilistic algorithms like EM, density based algorithms, and any other centroid based clustering algorithms. Our merging algorithms build consensus chains/groups from the ensemble of centroids. The formation of a consensus chain/group provides the framework for filtering malformed clusters in an ensemble and prevents "blind" merging of base clustering solutions, which to our knowledge has not been previously addressed; however, the performance of algorithms under noise was studied in [2]. Although the concept of filtering and Bipartite Merger have been introduced in our earlier work [14], in this work we propose and evaluate another merging algorithm, the Metis merger. In addition, we perform experiments on large or very large data sets from various domains, and quantitatively evaluate the performance of all of them compared to a relevant multiple partition combining algorithm, clustering all data at once in memory, the average base clustering solution, and a single pass algorithm.

2.2 Single Pass Fuzzy C Means

Incremental approaches are becoming an attractive choice for clustering large data sets. Single pass algorithms could be used to cluster large or very large data sets using a single pass through the data on the disk. Generally, we expect that large or very large data sets can not be loaded into memory. Clustering algorithms are generally iterative in nature. Disk accesses will be required every iteration for unloadable data sets. Disk accesses will make clustering algorithms unacceptably slow and an unattractive choice for data analysis purposes. Even if the data could be loaded in memory, generally, memory requirements for soft clustering algorithms are larger than the size of the data. This is due to the fact that soft clustering algorithms

generally need to keep additional data structures in memory. One such example is the U matrix, which keeps the membership degrees of the examples, in the fuzzy c means algorithm. Moreover, we will show our single pass algorithm to be much faster compared to the original algorithm, even if data is loadable into memory. So, it can be used for speed up purposes also. Single pass algorithms may be used to speed up clustering of the individual sites in the cluster ensemble problem.

2.2.1 Motivation

Recently various algorithms for clustering large data sets and streaming data sets have been proposed [28], [24], [26], [21], [40], [41], [42], [43]. The focus has been primarily either on sampling [28], [40], [41], [44], [27] or incrementally loading partial data, as much as can fit into memory at one time. The incremental approach [26], [21], [42], [43] generally keeps sufficient statistics or past knowledge of clusters from a previous run of a clustering algorithm in some data structures and uses them in improving the model for the future. Various algorithms [29], [22], [45], [46], [47] for speeding up clustering have also been proposed. While many algorithms have been proposed for large and very large data sets for the crisp case, not as much work has been done for the fuzzy case. As pointed out in [44], the crisp case may not be easily generalized for fuzzy clustering. This is due to the fact that in fuzzy methods an example does not belong to a cluster completely but has partial membership values in most clusters [1].

Clustering large amounts of data takes a long time. Further, new unlabeled data sets which will not fit in memory are becoming available. To cluster them, either sub sampling is required to fit the data in memory or the time will be greatly affected by disk accesses making clustering an unattractive choice for data analysis. Another source of large data sets is streaming data where you do not store all the data, but

process it and delete it. There are some very large data sets for which a little labeled data is available and the rest of the data is unlabeled i.e. for example, computer intrusion detection. Semi-supervised clustering might be applied to this type of data [48]. In general, clustering algorithms which can process very large data sets are becoming increasingly important.

2.2.2 Related Work

In [29], a multistage random sampling method was proposed to speedup fuzzy c means. There were two phases in the method. In the first phase, random sampling was used to obtain an estimate of centroids and then fuzzy c means (FCM) was run on the full data with these centroids initialized. A speed-up of 2-3 times was reported. In [45], speed up is obtained by taking a random sample of the data and clustering it. The centroids obtained then were used to initialize the entire data set. This method is similar to that in [29]; the difference is they used one random sample where in [29] they may use multiple random samples. In [28], another method based on sampling for clustering large image data was proposed, where the samples were chosen by the chi-square or divergence hypothesis test. It was shown that they achieved an average speed-up of 4.2 times on image data while providing a good final partition using 24% of the total data.

Another speed up technique for image data was proposed in [22]. In this method FCM convergence is obtained by using a data reduction method. Data reduction is done by quantization and speed-up by aggregating similar examples, which were then represented by a single weighted exemplar. The objective function of the FCM algorithm was modified to take into account the weights of the exemplars. However, the presence of similar examples might not be common in all data sets. They showed that it performs well on image data. In summary, the above algorithms attempt to

speed up fuzzy c means either by reducing the number of examples through sampling or by data reduction techniques or by providing good initialization points to reduce the number of iterations. However, the above algorithms do not seem to address the issue of clustering large or very large data sets under the constraint of limited memory. Moreover, some of them address the speed up issues for image data only where the range of features may be limited. Some work on parallel/distributed approaches has been done, where multiple processors could be used in parallel to speed up fuzzy c means [14], [49]. In [50] a parallel version of the adaptive fuzzy Leader clustering algorithm has been discussed, whereas, in [51] an efficient variant of the conventional Leader algorithm known as ARFL (Adaptive rough fuzzy leader) clustering algorithm was proposed.

There has been research on clustering large or very large data sets [24], [26], [21], [40], [41]. Birch [24] is a data clustering method for large data sets. It loads the entire data into memory by building a CF (Clustering Feature) tree, which is a compact representation of the data using the available memory. Then the leaf entries of the CF tree can be clustered to produce a partition. A hierarchical clustering algorithm was used in their paper. It provides an optional cluster refining step in which quality can be further improved by additional passes over the dataset. However, the accuracy of data summarization depends on the available memory. It was pointed out in [26] that depending on the size of the data, memory usage can increase significantly as the implementation of Birch has no notion of an allocated memory buffer. In [26], a single pass hard c means clustering algorithm was proposed under the assumption of a limited memory buffer. They used various data compression techniques to obtain a compact representation of data. In [21], another single pass scalable hard c means algorithm was proposed. This is a simpler implementation of Bradley's single pass algorithm [26], where no data compression techniques have been used. They showed

that complicated data compression techniques do not improve cluster quality much while the overhead and book-keeping of data compression techniques slow the algorithm down. Bradley's algorithm compresses data using a primary compression and secondary compression algorithm. In primary compression, data points which are unlikely to change membership were put into a discarded set. The remaining points were then over clustered with a large number of clusters compared to the actual number of clusters. This phase is known as secondary compression and its objective is to save more buffer space by representing closely packed points by their clusters. A tightness criterion was used to detect clusters which can be used to represent points. These sub-clusters were further joined by an agglomerative clustering algorithm provided after merging they are still tightly packed. So, in summary Bradley's implementation has various data compression schemes which involve overhead and book-keeping operations. Farnstorm's single pass [21] algorithm is basically a special case of Bradley's where after clustering, all data in memory is put into a discard set. A discard set is associated with every cluster, and it contains the sufficient statistics of data points in it. The discarded set is then treated as a weighted point in subsequent clustering. Other relevant single pass algorithms for the crisp case can be found in [42] and [43]. Compressing data was also studied in [52], where the number of templates needed by the probalistic neural network (PNN) was reduced by compressing training examples with exemplar. This was done by estimating the probability density functions for the PNN with Gaussian models.

Recently in [44], a sampling based method has been proposed for extending fuzzy and probabilistic clustering to large or very large data sets. The approach is based on progressive sampling and is an extension of eFFCM [28] to geFFCM, which can handle non-image data. However, the termination criteria for progressive sampling could be complicated as it depends upon the features of the data set. They used

4 acceptance strategies to control progressive sampling based on the features of the data. The first strategy (SS1) is to accept the sampled data when a particular feature signals termination. The second one (SS2) is to accept when any one of the features signals termination. The third one (SS3) is to accept when all the features sequentially signal termination and the last one accepts (SS4) when all the features simultaneously signal termination. However, the method could be complicated for a large number of features and the sample size could grow large also.

In [53], two methods of scaling EM [1] to large data sets have been proposed by reducing time spent in E step. The first method, incremental EM, is similar to [54], in which data is partitioned into blocks and then log-likelihoods are incrementally updated. In the second method, lazy EM, at scheduled iterations the algorithm performs partial E and M steps on a subset of data. In [55], EM was scaled in a similar way as they scaled k-means in [26]. In [56], data is incrementally loaded and modeled using gaussians. At the first stage the gaussians models are allowed to overfit the data and then later reduced at a second stage to output the final models. The methods used to scale EM may not generalize to FCM as they are different algorithms with different objective functions.

In [43], a streaming algorithm for hard-c-means was proposed in which data was assumed to arrive in chunks. Each chunk was clustered using a LOCALSEARCH algorithm and the memory was freed by summarizing the clustering result by weighted centroids. This is similar to the method of creating discard set in the single pass hard c means algorithm [21]. Finally, the weighted centroids were clustered to obtain the clustering for the entire stream. We also summarize clustering results in a similar way. The difference between [21, 43] and our approach is in the fact that in fuzzy clustering an example may not completely belong to a particular cluster. Our method

of summarizing clustering results involves a fuzzy membership matrix, which does not exist for the crisp cases.

Thus some work has been done for the crisp cases (hard-c-means or hierarchical clustering) and the EM algorithm, but as stated earlier the crisp case may not be easily generalized to fuzzy clustering and to our knowledge no single pass fuzzy c means algorithm has been proposed that takes into account the membership degrees, which describes the fuzziness of each example.

2.3 Algorithms for Clustering Data Streams

Clustering data streams is a challenging area. Data has to be processed as it comes and it might also evolve with time. A good streaming algorithm should be able to summarize data seen so far and also able to respond to an evolving distribution. Single pass algorithms may not be well suited for clustering streaming data as there is no notion of controlled amounts of history usage, and outdated history might dominate clustering. The streaming agorithm proposed in this work is generally a variant of the single pass FCM algorithm. In the streaming variant, the usage of history can be controlled by the user. We also proposed another variant of the streaming algorithm, the online FCM, which can be used for producing cluster quality as good as clustering an entire stream. It will be able to process data as it comes and thus can be used for real time processing. The online FCM algorithm can be viewed as a generalized version of the single pass algorithm proposed before. This generalized version can process data streams in any order.

2.3.1 Motivation

Clustering streaming data is becoming important due to the availability of large amounts of data recorded from everyday transactions. The sources of streaming data

are increasing as technology for recording events is becoming cheaper. Streaming data might flow for days, months, or even years. It might not be possible to store all the data, but necessary to process it and delete it. Streaming data may also evolve over time [57–61] and has the constraint that it cannot be revisited and has to be processed as it comes, that is, no random access is possible. Finding meaningful clusters under these constraints is challenging. A number of algorithms have been proposed recently which cluster streaming data by using a single pass approach [43, 62]. A single pass approach may be applicable for scaling clustering algorithms but may not fit well for clustering streaming data [57]. This is because streaming data might evolve over time and a single pass view of the entire stream might make algorithms insensitive to an evolving distribution [57].

A good streaming algorithm should not only summarize data seen so far, but also respond to dynamic changes. As stated in [57] a streaming algorithm should be able to produce a good quality partition even if data is evolving considerably over time. Streaming algorithms may also be used for scaling purposes by clustering extreme data sets on large RAIDS. One advantage of streaming algorithms over many single pass and other scalable algorithms [21, 28, 44, 63] is that they don't require random access to data points and process data as it arrives.

2.3.2 Related Work

Recently many algorithms have been proposed for clustering streaming data sets [43, 57–62, 64–66]. Most of them address the crisp case of clustering streaming data by using either hard c means or its variants or other crisp algorithms. In [43] a streaming algorithm was proposed using a k-Median algorithm called LOCALSEARCH. They showed that their LOCALSEARCH algorithm was better in quality but computationally expensive compared to hard-c-means. They viewed the

streaming data as arriving in chunks and then, after clustering, memory was purged by representing the clustering solution by weighted centroids. Then they applied the LOCALSEARCH algorithm to those weighted centroids obtained from chunks to obtain the weighted centroids of the entire stream seen so far. They showed that their algorithm outperformed BIRCH [24] in terms of quality measured in sum of squared distance (SSQ). Generally, this criteria is minimized in hard c means. This method of freeing the memory is similar to the method of creating a discard set in the single pass hard c means algorithm [21]. We also summarize clustering results in a similar way. The difference between [21, 43] and our approach is in the fact that in fuzzy clustering an example may not completely belong to a particular cluster. Our method of summarizing clustering results involves a fuzzy membership matrix and fuzzy centroids, which do not exist for the crisp cases. So in [43], clustering streaming data was approached using a single pass view of the data. In [57], it was pointed out that a streaming algorithm may not be viewed as single pass clustering problem because they are generally blind to evolving distributions and a single pass algorithm over an entire stream will be dominated by outdated history. They proposed a framework for analysis of clusters over different time frames. They stored summary statistics describing the streaming data periodically using micro-clusters which was the online component of their algorithm, and later analyzed these summary statistics of clusters, known as the offline component, over a user provided time horizon. They showed the superiority of their algorithm compared to [43] on data with an evolving distribution.

In [60] a density based streaming algorithm DenStream was proposed. The design philosophy of the DenStream algorithm was similar to [57] as they too had an online component for summarizing cluster information and then an offline component later to combine clusters. They used the density based DBSCAN algorithm [67] in their work. Using a density based clustering algorithm they were able to discover arbitrary

shape clusters and showed the robustness of their algorithm towards noise. However, density based algorithms are different from fuzzy clustering algorithms as they try to optimize a different objective function. In [68] a framework for efficiently archiving high volumes of streaming data was proposed, which reduces disk access for storing and retrieving data. They grouped incoming data into clusters and stored them instead of raw data.

Many other relevant single pass or scalable algorithms using hard c means, EM [1], Hierarchical Clustering and their variants exist [24, 26, 42, 54, 56, 58]. A streaming algorithm using artificial immune system (AIS) models was also proposed in [61]. As stated before the fuzzy c means algorithm optimizes a different objective function and also the single pass approach may not be suitable for clustering an evolving stream.

Non-incremental algorithms for speeding up fuzzy c means or hard c means [22, 24, 28, 29, 40, 44] are not generally applicable to clustering streaming data sets because they assume the entire data is loadable into memory. In [69] a modified FCM was proposed to simplify hardware implementation and obtain parallelism for real time video processing, but is very application specific and not applicable for data streams. In [70] a data driven fuzzy clustering method based on the Maximum Entropy Principle was proposed for a real time robot-tracking application. It is application specific and does not have the same objective function as FCM.

A good streaming algorithm should be able to respond to an evolving distribution and should also summarize the data seen so far. Obviously, the summarized results of past history should not make a streaming algorithm insensitive to an evolving distribution. By varying the amount of past history used, a tradeoff may be achieved between summarization of data seen so far and response to an evolving distribution. In this work we will study the tradeoff between summarization and responsiveness in the context of designing a streaming algorithm. If we do not use past history it

might reflect local changes sharply, but would not be able to summarize data seen so far. So, the usage of history may depend upon the context of application as some may require streaming algorithms to respond to the changing environment quickly and some may require more usage of past history.

CHAPTER 3

A SCALABLE FRAMEWORK FOR CLUSTER ENSEMBLES

In this chapter we propose a scalable framework for cluster ensembles. Clustering algorithms are used to partition unlabeled data. Clustering data is often considered to be a slow process. This is especially true of iterative clustering algorithms such as the K-means family [22] or EM [1]. As larger unlabeled data sets become available, the scalability of clustering algorithms becomes more important. It has become increasingly difficult to load large data sets into a single memory for clustering [14,39]. Data can be sequentially clustered in such a way that each data subset fits in memory and finally the clustering solution of all subsets can be merged. Sometimes, data is physically distributed and centrally pooling the data might not be feasible due to privacy issues and cost. Thus, merging clustering solutions from distributed sites is required. Moreover, iterative clustering algorithms are sensitive to initialization and produce different partitions for the same data with different initializations. Combining multiple partitions will provide a robust and stable solution [2,6]. Also, combining multiple partitions may produce novel clustering solutions which might not be possible when clustering all the data [2, 6]. In [15], it has been shown that consensus clustering converges to the true underlying distribution as the ensemble size grows. This is a justification for the use of cluster ensembles. So, combining multiple clustering solutions has emerged as an important area of research to address the issue of scalability, the distributed nature of some data, and the robustness or stability of the clustering solution. Recently in [71], improving the clustering solution by weighting examples

in a concept analogous to boosting in supervised learning has been addressed, where less efficiently clustered points are given more weight. But, it does not fall under the class of multiple partition combining algorithms.

3.1 Ensemble Merging

In this work, we set the number of clusters for each partition of the ensemble to be the same as that of the final target clustering, that is, k . The assumption is that if the base clusterings, the clustering solutions in the ensemble, are from the same/similar underlying distribution, each will likely have the same number of classes. Hence, there will exist a meaningful correspondence between clusters of the partitions. We believe data originating from the same/similar underlying distribution is a standard assumption in clustering.

In a graph theoretical formulation, if the centroids of each partition are represented by non-adjacent vertices of a graph and the Euclidean distance between a centroid of a partition and other partitions as a weighted edge, then it becomes a r-partite graph, where r is the number of partitions to combine. Our objective is to partition this r-partite graph into k (target clusters) groups such that "similar" centroids are grouped together. If the base clusterings/partitions have an equal number of clusters and meaningful correspondences exist among them, the group size will tend to be equal. The centroids in each group will then be used to compute a global centroid. The partitioning of this r-partite group is like clustering the clustering solutions, and optimizing it is generally a NP hard problem.

So, our two heuristic algorithms, Bipartite Merger and Metis Merger, attempt to partition the ensemble of centroids, the r-partite graph. The Bipartite Merger heuristic partitions the ensemble of centroids into k exactly equal size groups using the constraint of one to one centroid mapping between any two partitions. This

assumption of meaningful one to one correspondence between clusters of the partitions has also been used in [5, 7] for re-labeling purposes. Metis Merger tends to partition ensembles of centroids, using the graph partitioning package METIS [72–74], into k approximately equal size groups. Note, the group sizes may not be the same.

The assumption of an equal number of clusters in all base clustering solutions and hence one to one meaningful correspondence, especially in Bipartite Merger, may not cover all cases, for example the base clustering solutions may already exist with a different number of clusters, and in this case only a knowledge reuse framework is needed for merging. We do not specifically address this type of scenario in this work. Our Metis merger may be used to partition the r-partite graph when the number of clusters in base clusterings are not the same, but it is likely to work better under the setting where meaningful cluster correspondences exist. The METIS graph partitioning package tends to partition the r-partite graph into approximately equal size groups [2, 3, 73]. HMETIS [75], a graph partitioning package, was used in [2] to partition an ensemble represented by hyperedges, created from a label vector representation of clustering solutions, into unequal size groups by using a vertex imbalance constraint approach. To use HMETIS to partition an ensemble of centroids, multiple "weak" clustering solutions of the ensemble is required. One way of creating multiple solutions is to cluster the ensemble of centroids multiple times with a number greater than k (over clustering). Each cluster then will form a hyperedge. Thus, over clustering (with a varying number greater than k) the ensemble multiple times with different initializations will produce overlaps among the hyperedges, which can be merged later into k final macro clusters using the HMETIS package. This approach will partitions the entire ensemble of centroids and will have no restrictions on the number of clusters coming from each site. It will then address those cases in which

the number of clusters in the base clustering solutions are not the same. One may also modify our Bipartite Merger.

The focus here is on scalability issues, and we will show that for the same ensembles, a centroid based representation is better than or as good as the label vector based representation, while providing superiority in terms of time and space complexity.

In the first method, we merged the ensemble using the bipartite matching algorithm, and named it the Bipartite Merger (BM). The second method uses the graph partitioning package METIS [72–74], and is called the Metis Merger (MM). METIS was also used by Strelh and Ghosh [2] in their multiple partition combining algorithm, MCLA, to merge base clusterings represented in the form of label vectors. We have used METIS to merge base clustering solutions represented in the form of centroids. Below, we describe our two ensemble merging algorithms.

Between any two partitions, Bipartite Merger optimally matches clustering solutions, but the constraint of one to one mapping may force a matching of clusters without meaningful correspondence. This might happen if one or more base clusterings are relatively noisy or have originated from an unbalanced distribution. To address this problem and make our framework more robust, we have proposed a filtering algorithm, described later, that filters out "bad centroids". The filtering concept has also been applied to Metis Merger to remove malformed centroids, if any.

As stated earlier, the number of clusters for each partition of the ensemble was the same as that of the final target clustering, k. After clustering was applied to each disjoint subset, which in our experiments were created from randomly dividing the full data, there were set of centroids available which describe each partitioned subset. Clustering or partitioning a subset $\{S_i\}$ will produce a set of centroids, base clustering solutions, $\{C_{i,j}\}_{j=1}^{k}$, where k is the number of clusters. For r subsets there will be r sets of centroids i.e.$\{C_{1,j}\}_{j=1}^{k}$, $\{C_{2,j}\}_{j=1}^{k}$,....,$\{C_{r,j}\}_{j=1}^{k}$ forming an ensemble

of centroids. To produce the final partition, we need to reach a consensus of the position of the centroids in the target clustering.

3.1.1 Bipartite Merger

One way to reach a global consensus is to partition the ensemble of centroids into k consensus chains, where each consensus chain will contain r centroids $\{c_1^n, ..., c_r^n\}$, one from each of the subsets, where n runs from 1 to k. This assumes one to one correspondence of the clusters in the base clustering solution. The aim is to group similar centroids in each consensus chain by solving the cluster correspondence problem. The objective is to globally optimize the assignment ψ^* out of all possible families f of centroid assignments to k consensus chains:

$$\psi^* = \arg\min_{\psi \subset f} \{\psi\} \tag{3.1}$$

$$\psi = \sum_{n=1}^{k} \cos t \left(consensus\_chain(n) \right) \tag{3.2}$$

$$\cos t \left(consensus\_chain(n) \right) = \sum_{i=1}^{r} D^n \left(i \right) \tag{3.3}$$

$$D^n \left(i \right) = \frac{1}{2} \sum_{j=1}^{r} d \left(c_i^n, c_j^n \right)_{j!=i} \quad , \tag{3.4}$$

where $d \left(., . \right)$ is the distance function between centroid vectors in a consensus chain. We have used the Euclidean distance in computing the cost (3.4). So, in a graph theoretical formulation finding the globally optimum value for the objective function (3.1) reduces to the minimally weighted perfect r-partite matching problem, which is intractable for r>2. Because the optimization problem is intractable (NP-Hard), a heuristic method is used to group centroids.

We know that for 2 partitions, r=2, there is a polynomial time algorithm i.e. minimally weighted perfect bipartite matching to globally optimize the above objective function. For optimally matching centroids of 2 partitions we have used the Hungarian method of minimally weighted perfect bipartite matching [76]. So, we match two partitions at a time. After matching a pair of partitions we keep the centroids of one of the pair as the reference and a new partition is randomly chosen and matched i.e. minimally weighted bipartite matching. Now, the centroids of this new matched partition are in the same consensus chain as centroids of the reference partition. In this way we continue grouping the centroids of new partitions into consensus chains one by one until they are exhausted. After the consensus chains are created, we simply compute the weighted arithmetic mean of centroids in a consensus chain to represent a global centroid, where the weights of a centroid are determined from the size of the subset from which it was created. In some cases, especially in the knowledge reuse framework, weights/importance of a base clustering solutions may be difficult to obtain. In those cases, all the base clustering solutions may be considered to have the same weight/importance. Each consensus chain tells us which centroid in which subset is matched to which centroid in other subsets. A final partition, in the form of label vectors, may be obtained by assigning an example to the nearest global centroid. It should be noted that Bipartite merger partitions the ensemble of centroids into k perfectly balanced chains, that is, each chain has the same number of centroids (one from each base clustering solution).

In [4,11], both examples and clusters were modeled as a graph in which an example vertex can connect only to a cluster vertex, thus formulating the problem of finding a consensus partition as a bipartite matching problem. In our case, we formulated it using the centroids only, thus the time and space complexity will not involve n, the

number of examples in a data set.

Example

Consider the case that there are 4 subsets S1, S2, S3, and S4 of a data set as shown in Figure 3.1 and each subset is grouped into 3 clusters. Let S1 first match with S2, then S2 with S3, and S3 with S4. After solving the matching problem, a consensus chain contains *similar types* of centroids as shown in Figure 3.2.

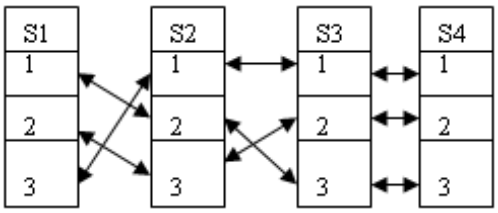


Figure 3.1 Partitions matched.

S1	S2	S3	S4
1	2	3	3
2	3	2	2
3	1	1	1

Figure 3.2 Each row a consensus chain.

3.1.2 Metis Merger

In Metis merger, the ensemble of centroids, the r-partite graph, is partitioned into k groups. METIS partitions the ensemble of centroids into approximately equal size groups; however, the groups may not be perfectly balanced, that is, the number of centroids in each group may not be the same. This is one difference from Bipartite merger, which guarantees perfectly balanced grouping. We call each group from Metis merger a *c*onsensus group. Similar to the concept of *c*onsensus chain in Bipartite Merger, it contains similar centroids grouped together. The difference is consensus

chains are equal in size while consensus groups are not guaranteed to be equal in size. Moreover, it does not use the assumption of one to one correspondence like Bipartite Merger. Also unlike Bipartite merger, which optimizes 2 partitions at a time, Metis merges partitions after processing the whole r-partite graph, that is, trying to optimize grouping cost globally.

For partitioning the ensemble of centroids using METIS, the weights of the r-partite graph have to be normalized so that the edge weights are proportional to the similarity between centroids, that is, similar centroids should be connected by high weighted edges and vice versa. This is done by finding the maximum weight, $maxWeight$, of the r-partite graph and then all edge weights are subtracted from it. This will ensure that edge weights are proportional to the similarity between centroids. Because the METIS graph package accepts only nonzero integer edge weights, floating point precision up to two places is maintained by multiplying all edge weights by 100 and then rounding off. One may multiply by a number greater than 100 to keep more precision. All edge weights are ensured to be non zero by adding 1 to all weights so that the minimum weight can be 1 and the maximum $maxWeight + 1$. After partitioning the centroids into k consensus groups, a global centroid is found by computing the weighted arithmetic mean of centroids in each group. Similar to Bipartite merger, weights of a centroid are determined from the size of the subset from which it was created. A final partition, in the form of label vectors, may be obtained by assigning an example to the nearest global centroid.

In Bipartite Merger, we are optimally matching the centroids of two randomly picked partitions at a time. But the question arises, in which order should the partitions be matched so that the overall matching cost is minimum. We have found it to be equivalent to solving the traveling salesman problem. This is because we are optimally matching two partitions at a time and if we could also optimally select their

order of selection then we would be optimizing the whole objective function (3.1), which is an intractable problem. The implication of random pair-wise matching is that in a consensus chain between any three centroids there is a transitive relation i.e. if centroid C1 is matched to C2, and C2 is matched to C3, then C1 is matched to C3. We believe in the cluster correspondence problem this assumption of transitive relation is quite valid. One could also use an approximate traveling salesman algorithm to determine a sub-optimal minimum cost order. It would obviously add some computation time to the merging algorithm. Metis Merger uses the graph partitioning package METIS, which is a randomized algorithm. So, it is also sensitive to the order the vertices are numbered in the r-partite graph. We have conducted experiments to estimate the sensitivity of our algorithms.

There may be various ways to partition this r-partite graph. But, we chose METIS and the bipartite matching algorithm because they are well studied. Metis was used to combine multiple partitions represented by label vectors or their variants.

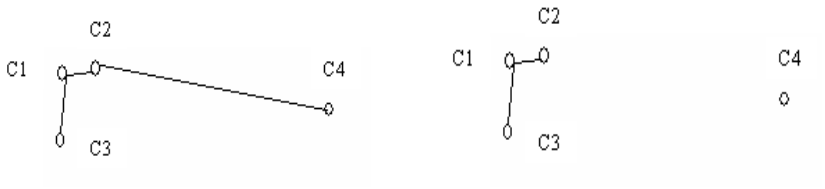
3.2 Filtering Bad Centroids

We introduced the concept of detecting and removing spurious or malformed clusters from the cluster ensemble, which may result from an unfortunate initialization or unbalanced data distribution or noise in the data set [14]. As discussed earlier, malformed centroids may also be present in a consensus chain/group due to the constraint of one to one mapping in BM and the tendency of METIS to partition an ensemble into equal size groups. This will tend to happen in the absence of meaningful cluster correspondence in an ensemble. In Bipartite Merger and MM, each consensus chain/group contains matched centroids, and the problem is equivalent to removing outliers/noise from it, if present. But with a limited number of centroids in a consensus chain and no knowledge about the distribution of noise, detecting them is

a non trivial task. It is reasonable to assume that "good" centroids would be spatially close to each other forming the largest compact cluster. Thus we have to search for this cluster and the rest would be classified as outliers/noise. So, for each consensus chain we built a minimally weighted spanning tree from a complete graph whose vertices are the centroids in a consensus chain and the Euclidean distance between centroids is the weight of the edges. Next, we sort the edges of the tree and find the statistical median. Now, if all edges of intermediate or high lengths are removed, this will reveal clusters [77] in the consensus chain. We have set the threshold to cut edges as: median+2.5*median. After cutting the edges of the spanning tree, it becomes a forest. Now, we take vertices of the largest tree/cluster from this forest for merging and the rest as outliers. This largest connected component/tree is found using a depth-first search algorithm. As stated earlier merging is then done by computing the weighted arithmetic mean of these centroids in a consensus chain, where the weights of a centroid are determined by size of the subsets it represents. Thus, each filtered consensus chain/group will result in a final centroid. In case of Metis Merger, the consensus groups, formed after partitioning the r-partite graph, are similar in concept to the consensus chain in the Bipartite Merger. So, the concept of filtering is the same. Here, for each of the k consensus groups a spanning tree is built from centroids in it, and the rest is similar to the discussion for the Bipartite Merger case.

Example: Consider the example described in the previous section by Figure 3.1 to 3.2. Let each consensus chain (consensus group in the case of Metis Merger) have 4 centroids. As mentioned earlier, for Metis Merger the consensus group is not guaranteed to be of equal size; nevertheless, in this example we assume they are of equal size, that is, 4 centroids in each group. Figure 3.3a shows the building of a spanning tree from such a consensus chain/group and Figure 3.3b shows the resulting

forest after cutting edge(s) from such a minimally weighted spanning tree. The largest connected component C1, C2, and C3 is then selected for merging.



(a) (b)

Figure 3.3 a) Minimally weighted spanning tree formed from centroids in a consensus chain/group. b) After cutting edge(s) above threshold, the largest connected component (C1, C2, and C3) will be used for merging.

The code for bipartite matching (Hungarian method), depth first search, and minimally weighted spanning were obtained from http://reptar.uta.edu/.

3.3 Experimental Setup

In [3] an extensive comparison of Soft-Correspondence Ensemble Clustering (SCEC) was done with four other state of the art algorithms on three real data sets Iris, Pen digits, and Isolet6. The algorithms compared were CSPA [2], MCLA [2], QMI [8], and MMEC [6]. CSPA and MCLA are graph partitioning algorithms while QMI uses k-means, and MMEC is based on mixture model based clustering. Although no single algorithm was a clear winner on all data sets, overall SCEC performed better in a majority of the cases followed closely by MCLA; however, they also reported that MCLA tends to give better partitions with the true number of clusters because it provides nearly balanced clusters in the ensemble. In [2], it was stated that MCLA performs best, compared to HGPA and CSPA, when meaningful cluster correspondence is likely to exist. In [12], it was also reported that MCLA was a strong competitor to the reported algorithm PLA (Probabilistic Label Aggregation) and wPLA (

weighted Probabilistic Label Aggregation) as compared to the three other algorithms CSPA [2], HGPA [2], and a mixture model based approach [6]. In our framework, we always assign the number of clusters in the base clustering solutions and target clustering to be the same, k. Hence, it is likely that cluster centroids in the ensemble will be balanced and a meaningful correspondence will exist between clusters in the base clustering solutions. Our objective function formulation also makes this assumption. So, we choose to compare our algorithms with MCLA, which is likely to perform well in this setting. As our Metis merger and Strehl/Ghosh's MCLA used the same graph partitioning package, METIS, it will also provide a good framework for evaluating the quality of ensemble formed from centroids verses ensemble formed from label vectors. We also planned to compare with SCEC, but the source code for SCEC is not available. We implemented our algorithms in C. Although a matab implementation of MCLA is available at http://www.strehl.com, we implemented a C version to empirically compare running time to our algorithms. The METIS package (in C code), which MCLA and MM uses, is available at http://glaros.dtc.umn.edu/gkhome/views/metis.

In [14], our earlier work, we showed that bipartite merger (BM) is competitive or better compared to five algorithms, SCEC [3], CSPA [2], MCLA [2], QMI [8], and MMEC [6]. The evaluation was done by performing experiments on the same three data sets used in [3], Iris, Pendigits, and Isolet6, and then comparing with the reported results on the same three data sets.

One of the main reasons for choosing MCLA to compare with our multiple partition combining algorithms is because the objective of MCLA is similar, that is, grouping similar clusters by solving the cluster correspondence problem, assuming meaningful cluster correspondence exists in the base clustering solutions. The main difference is that in MCLA the base clustering solutions are represented by label

vectors from which they generate hyperedges, and in our approach the clustering solutions are represented by centroids, that is, a centroids based ensemble.

In our approach, as stated earlier, multiple clustering solutions (an ensemble) are created from disjoint data sets. But MCLA requires representation of clustering solutions in the form of label vectors over all examples (global data) because the label vectors created from examples of disjoint subsets will not have any overlap and hence no meaningful correspondence. This could be an obstacle to scalability for MCLA and other label vector based ensemble merging algorithms as clustering the data in a distributed way allows for scaling. Nevertheless, for MCLA a label vector equivalent representation of a base clustering solution in an ensemble was created, assigning an example to the nearest centroid in that partition, over all the examples, that is, global data.

A clustering solution, in the form of label vectors, containing k clusters is represented by k hyperedges. Each hyperedge is an indicator vector, whose value is 1 if the examples are in the same cluster, 0 otherwise. So, the length of each hyperedge is n, the number of examples. Thus for a clustering solution, the hyperedge representation is k times bigger, in terms of space, than the label vector itself. In MCLA, a meta-graph is then constructed from these hyperedges, where the edge weights are proportional to the similarity between hyperedges, that is, the vertices of the graph. For computing distance between hyperedges, which are indicator vectors, they used the binary Jaccard distance. Then this meta-graph is partitioned by the graph partitioning package METIS. In summary, an ensemble formed by clustering solutions in the form of label vectors involves time complexity and space complexity in terms of n, the number of examples. An ensemble formed from centroids has time and space complexity that does not involve n. A detailed comparison of time and space complexity will be provided later.

It is well known that even if a data set is centrally available, loading all the data into memory and clustering it can be time consuming for large data sets. If we cannot load the entire data into memory, hard-k-means or fuzzy-k-means will have to make disk accesses every iteration. To address this problem, in [21] a single pass clustering algorithm (SP) was described for hard-k-means, where only part of the data was loaded into memory at a time and the whole data set was clustered in one disk access. We have also compared the ensemble merging clustering algorithms with this approach. Although ensemble merging clustering algorithms and single pass algorithms provide very different frameworks; the purpose of comparing them is to throw some light on which type of algorithms to use when we have the opportunity to apply both of them. For example, if we have data centrally pooled, we can apply a single pass algorithm or cluster the data distributively, after dividing the data, and merge the clustering solutions. So, comparing the quality of both these frameworks will be worthwhile. Obviously, if data is geographically distributed (assuming centrally pooling the data is difficult) or have already been clustered (knowledge reuse), multiple partition combining algorithms have to be used. The code for the single pass hard k means algorithm [21] is available at http://www-cse.ucsd.edu/users/elkan/skm.html.

Measuring cluster quality is a non trivial task. Unlike supervised learning, where the learning algorithm uses real class labels to minimize error over training examples, no information about the true labels are used to group data into different clusters. The sole purpose of a clustering algorithm is to try to optimize its objective function, which may result in a good set of clusters. Moreover, true labels will not typically be available. One way of evaluating cluster quality without using true labels is computing the sum of squared error criterion, that is, sum of the square of Euclidean distance of the examples in a cluster from their geometric mean. This intuitively indicates

how compact or scattered the examples are in clusters. We have used this metric to evaluate and compare clustering quality in our experiments.

The sum of squared error (SE) is defined as follows [77]: For a data set $D = \{x_1, ..., x_n\}$ of n examples, the task of clustering is to partition it into k disjoint subsets $D_i, ...D_k$. The SE of examples in the clusters is then $SE = \sum_{i=1}^{k} \sum_{x \in D_i} \|x - m_i\|^2$, where $m_i = \frac{1}{n_i} \sum_{x \in D_i} x$, where n_i the number of examples in D_i. Generally, this or its variants are minimized by many iterative clustering algorithms.

In [3], the NMI metric [2] has been used to evaluate and compare clustering quality, which requires the true labels of the data. If two clustering solutions are represented by label vectors, NMI computes the mutual information between the two label vectors. Its values range from 0 to 1. An NMI value of 1 means two partitions are exactly the same. The clustering solution of an algorithm, in the form of a label vector, can be measured in NMI by comparing with the true class labels of the data. But as stated earlier, clustering algorithms do not use the information of true labels to group data, so optimizing the objective function may not necessarily mean optimizing the solution in terms of the NMI metric, that is, minimizing error with true labels. Nevertheless, in [14], we have compared the quality of BM with the results reported in [3] using the NMI metric. We showed that our centroids based ensemble algorithm, BM, was competitive or better using the NMI metric [14].

In summary, using the SE metric we compared the quality of our multiple partition combining algorithms, BP and MM, with MCLA. In addition to that, we also compared the quality of each of these multiple combining algorithms with clustering all the data at once in memory (global clustering), average quality of base clustering solutions (BC), and a single pass algorithm (SP). We will show the ensemble formed by our centroid based approach has almost the same quality (sometimes even better) when compared to the label vector based approach, MCLA. Further, it provides a

speed up of the order of hundreds of thousand times on medium or large data sets. We will also show that compared to global clustering (GC), the average base clustering solution (BC), and a single pass clustering algorithm (SP), our centroids based approach performs as well or better than the label vector based algorithm MCLA. We will quantitatively evaluate the difference in quality in each case. We will study multiple partition combining algorithms on medium or large data sets and evaluate scalability issues in realistic scenarios, in terms of time and space complexities. We have tested our algorithms on seven real data sets from various domains, both image and non-image.

3.4 Data Sets Used

- *The Iris Data set.* The Iris plant data set consists of 150 examples each with 4 numeric attributes [78] and 3 classes of 50 examples each. One class is linearly separable from the other two. We clustered this data set into 3 clusters.

- *The ISOLET6 Data set.* It is a subset of the ISOLET spoken letter recognition training set and has been prepared in the same way as done in [3]. Six classes out of 26 were randomly chosen and it consists of 1440 instances with 617 features [78]. We clustered this data set into 6 clusters.

- *The Pen-Based Recognition of Handwritten Digits.* It consists of 3498 examples, 16 features, and 10 classes [78]. The classes are pen-based handwritten digits with ten digits 0 to 9. We have named it the Pen Digits data set in our experiments. We clustered this data set into 10 clusters.

- *Plankton.* The plankton data consists of 419,358 samples of plankton from the underwater SIPPER camera which records 8 gray levels. There were 26 features extracted. The samples were taken from the twelve most commonly encountered

classes of plankton during the acquisition in the Gulf of Mexico. The class sizes range from about 11,337 to 74,053 examples. We clustered this data set into 12 clusters.

- *KDD.* This data set is the 1998 KDD contest data set [79]. This data set is about people who gave charitable donations in response to direct mailing requests. It was used in [21], and has been preprocessed in the same way. After processing the original data, it has 95,412 examples and 56 features. As done in [21], we clustered this data set into 10 clusters. The code for preprocessing it is available at http://www-cse.ucsd.edu/users/elkan/skm.html.

- *MRI-1.* The MRI-1 data set was created by concatenating 45 slices of MR images of a human brain each of size 256X256 from modalities T1, PD, and T2. The magnetic field strength was 3 Tesla. After air was removed using a threshold, there are slightly above 1 million examples (1,132,545 examples/pixels, 3 features). We clustered this data set into 9 clusters.

- *MRI-2.* The MRI-2 data set was created by concatenating 48 slices of MR images of a human brain each of size 512X512 from modalities T1, PD, and T2. The magnetic field strength was 1.5 Tesla. After air was removed using a threshold, there are slightly below 4 million examples (3,991,592 examples/pixels, 3 features). We clustered this data set into 9 clusters.

The values of m used for fuzzy clustering were $m = 2$ for Iris, Plankton, MR1-1, MRI-2; $m = 1.2$ for ISOLET6, KDD98, and $m = 1.5$ for the pen digits data set. The different values enabled reliable partitions with the full data to be obtained.

Experiments on the 3 small data sets, Iris, Pen Digits, and Isolet6 were run on an UltraSPARC-III+ with one 900 MHz processor with 1024 MB memory. As memory

requirements with the other 4 medium or large data sets would increase, they were run on a different machine. It was an UltraSPARC-III with 8 processors each of 750 MHz. There was 32GB of main memory. None of the programs were multithreaded, so each program used only one processor at a time.

3.5 Results and Discussion

On the small data sets, Iris, ISOLET6, and Pendigits, two types of experiments have been conducted by splitting them in two ways. For the Iris data set, we used $r \in \{3, 5\}$ i.e. divided randomly into 3 and 5 disjoint equal size subsets. It means in the first experiment, an ensemble of size 3, base partitions from 3 subsets, will be merged, and in the second experiment base partitions from 5 subsets will be merged. Similarly, for ISOLET6 and Pendigits it was r=5 and 10. As clustering time is longer for the other 4 large/medium large data sets, we split them in only one way. Plankton and KDD98 were divided into 15 disjoint equal size subsets (r=15), while MRI-1 and MRI-2 were divided into 20 disjoint equal size subsets (r=20). We choose these values because we believe a 10% (r=10) to 20% (r=5) sample size to be reasonable for small data sets and 5% (r=20) for medium or large data sets. One may split into other reasonable size subsets. In each experiment, all the base clusterings/partitions in an ensemble were formed either using hard-k-means or fuzzy-k-means. When all base clusterings in an ensemble were formed by hard-k-means, we will denote that experiment as a hard ensemble experiment, and when the ensemble were formed by fuzzy-k-means, we will denote as a fuzzy ensemble experiment. All experiments are conducted with 50 random initializations. So, all results in the tables report an average over 50 random experiments.

3.5.1 Hard Ensemble Experiments

Table 3.1 shows the quality, in the SE metric, of the multiple partition combining algorithms. That is, Bipartite Merger (BP), Metis Merger (MM), and Meta CLustering (MCLA). In Table 3.1 the first column indicates the data set name and into how many disjoint subsets it was divided. For example, Iris 3S means that a row contains results of Iris data divided into 3 subsets. In Table 3.1, the best SE value (the lowest) for each experiment is italicized. It was observed that out of 10 sets of experiments none was the clear winner in all cases; MM was best in 4 cases and BM and MCLA each best in 3 cases. In Table 3.2, we quantitatively computed the difference in quality of our BP and MM algorithms compared to MCLA by using the results from Table 3.1. As the SE value of partitions could vary by data sets, we expressed the difference in quality (DQ) as a percentage. The difference in quality of our algorithm BM and MM, compared to MCLA, is defined as follows:

$$DQ\_BM\_MCLA = \frac{(BM - MCLA)}{MCLA} * 100 \tag{3.5}$$

$$DQ\_MM\_MCLA = \frac{(MM - MCLA)}{MCLA} * 100 \tag{3.6}$$

BM, MM, and MCLA are the quality values in the 1st, 2nd, and 3rd column of Table 3.1 respectively.

A negative value (italicized in Table 3.2) indicates the algorithm in that column, BP or MM, is better compared to MCLA. Although MM was best on the majority of experiments, 4 times in Table 3.1, results in Table 3.2 indicate on average over the 10 sets of experiments MCLA is 0.18% and 0.45% better compared to MM and BM respectively. So, both of our centroids based ensemble algorithms have quality similar to MCLA; MM performs slightly better than BM. Nevertheless, we will show later

that BM and MM can be hundreds of thousand times faster, compared to MCLA, on the medium or larger data sets.

Table 3.1 Quality, computed in squared-error (SE) criterion, of Bipartite Merger (BM), Metis Merger (MM), and Meta Clustering algorithm (SP) computed using the SE metric. Base clustering formed from hard-k-means. Italicized indicates lowest SE for an experiment.

	BM	MM	MCLA
Iris, 3S	*82.18*	84.87	84.04
Iris, 5S	84.14	*82.36*	83.09
Isolet6, 5S	84900.56	85077.89	*84626.02*
Isolet6, 10S	85998.40	84581.65	*84379.66*
Pen Digits, 5S	15619284..38	*15421779.21*	15564776.64
Pen Digits, 10S	15859731.96	*15062673.23*	15306065.54
KDD, 15S	3947179.06	3985303.18	*3925332.21*
Plankton, 15S	*5075393.96*	5241279.29	5113725.15
MRI-1, 20S	4643171175.64	*4625631083.40*	4626282184.65
MRI-2, 20S	*29361808465.06*	29460000480.17	29630820407.62

Table 3.3 shows the quality of Global clustering (GC), average base clustering solutions (BC), and Single pass hard-k-means (SP) computed using the SE metric. Results of each of the multiple partition combining algorithms, BP, MM, and MCLA presented in Table 3.1, will be compared to the results in Table 3.3 separately.

Table 3.4 shows the difference in quality of BP, MM, and MCLA compared to clustering all the data at once, that is, global clustering (GC). The difference in quality is expressed as a percentage and is defined as follows:

$$DQ\_BM\_GC = \frac{(BM - GC)}{GC} * 100 \tag{3.7}$$

$$DQ\_MM\_GC = \frac{(MM - GC)}{GC} * 100 \tag{3.8}$$

Table 3.2 Difference in quality of BM and MM compared to MCLA (formed from hard c means). All values expressed in percentage. Italicized indicates an algorithm better than MCLA on an experiment.

	DQ_BM_MCLA %	DQ_MM_MCLA %
Iris, 3S	*-2.213232*	0.987625
Iris, 5S	1.263690	*-0.878565*
Isolet6, 5S	0.324416	0.533961
Isolet6, 10S	1.918401	0.239382
Pen Digits, 5S	0.350197	*-0.918725*
Pen Digits, 10S	3.617301	*-1.590169*
KDD, 15S	0.556561	1.527793
Plankton, 15S	*-0.749575*	2.494349
MRI-1, 20S	0.365066	*-0.014074*
MRI-2, 20S	*-0.907879*	*-0.576494*
average	0.452495	0.180508

$$DQ\_MCLA\_GC = \frac{(MCLA - GC)}{GC} * 100 \tag{3.9}$$

where BM, MM, and MCLA are the quality values in the columns of Table 3.1 and GC is the SE value of the first column of Table 3.3.

A negative value (italicized in Table 3.4) indicates that the particular algorithm in that column is better when compared to GC. The results indicate that none of the multiple combining algorithms is better compared to GC on all the data sets. Out of 10 sets of experiments, MCLA is better than GC in 5 cases, MM in 4 cases, and BM in only 2 cases. However, on average, over all data sets and experiments, all multiple partition combining algorithms are better compared to GC, that is, MCLA is 1.21% better followed by MM, 1.03% better, and BM 0.75% better. So, on average the difference in performance, compared to GC, between MM and MCLA is 1.21-1.03=0.18%, and between MCLA and BM is 1.21-0.75=0.46%.

Table 3.3 Quality, computed in squared-error (SE) criterion, of Global clustering (GC), average base clustering solutions (BC), and Single pass hard-k-means (SP). GC and BC formed from hard-k-means.

	GC	BC	SP
Iris, 3S	94.48	88.09	99.60
Iris, 5S	94.48	94.01	98.15
Isolet6, 5S	84697.03	85272.18	85280.31
Isolet6, 10S	84697.03	86165.56	85708.75
Pen Digits, 5S	15033257.03	15395919.25	15484834.63
Pen Digits, 10S	15033257.03	15566617.66	15563107.05
KDD, 15S	3868364.17	3883997.73	3879132.90
Plankton, 15S	4985295.44	5001070.40	4966087.91
MRI-1, 20S	4635407930.55	4647970372.54	4649262104.30
MRI-2, 20S	29001948851.85	28991615370.97	29005498243.05

Table 3.4 Difference in quality of BM, MM, and MCLA compared to GC (formed from hard c means). All values expressed in percentage. Italicized indicates an algorithm better than GC on an experiment.

	DQ_BM_GC %	DQ_MM_GC %	DQ_MCLA_GC %
Iris, 3S	*-13.018628*	*-10.171465*	*-11.049958*
Iris, 5S	*-10.944115*	*-12.828112*	*-12.055461*
Isolet6, 5S	0.240304	0.449673	*-0.083840*
Isolet6, 10S	1.536500	*-0.136227*	*-0.374712*
Pen Digits, 5S	3.898204	2.584418	3.535625
Pen Digits, 10S	5.497644	0.195674	1.814700
KDD, 15S	2.037422	3.022958	1.472665
Plankton, 15S	1.807285	5.134778	2.576170
MRI-1, 20S	0.167477	*-0.210917*	*-0.196870*
MRI-2, 20S	1.240812	1.579382	2.168377
average	-0.753710	-1.037984	-1.219330

Table 3.5 shows the difference in quality of BP, MM , and MCLA compared to the average quality of base clustering solutions in the ensemble (BC). Quality of a base clustering solution is computed over all the examples, that is, global data. The difference in quality, as before, is expressed as a percentage and is defined as follows:

$$DQ\_BM\_BC = \frac{(BM - BC)}{BC} * 100 \tag{3.10}$$

$$DQ\_MM\_BC = \frac{(MM - BC)}{BC} * 100 \tag{3.11}$$

$$DQ\_MCLA\_BC = \frac{(MCLA - BC)}{BC} * 100 \tag{3.12}$$

where BM, MM, and MCLA are the quality values in the columns of Table 3.1 and BC is the quality value of the second column in Table 3.3.

A negative value (italicized in Table 3.5) indicates that the algorithm in that column is better compared to BC. Out of 10 sets of experiments, both MM and MCLA are better compared to BC 6 times, while BM is better 5 times. On average, compared to BC, MCLA is 1.45 % better, MM 1.26 % better, and BM 1.02% better. Thus, the difference in performance of MCLA with MM is 1.45-1.26=0.19 % and 1.45-1.02=0.43 % with BM, which is quite close. It was also reported in [3] that none of the 5 multiple partitioning algorithms, SCEC, CSPA , MCLA , QMI , and MMEC, was always better when compared to BC. It should be noted that getting better results, compared to GC or BC, for a multiple partition combining algorithm may depend on the ensemble size and how the ensemble was created. In [3], the ensemble size was much bigger than we used. We created an ensemble from disjoint subsets to put a restriction on the size of the ensemble and provide a scalable framework. One

may increase the ensemble size, using overlapped partitions or the full data set or any other relevant method, both for centroid based and label based approaches. Our main purpose here was to show that using identical ensembles, one represented by centroids and one represented by label vectors, the centroid based approach performs comparably to a label vector based approach, while providing superiority in terms of time and space complexity, which we will discuss later. Because we are discussing combining large or very large data sets, in this work we created ensembles in a scalable framework, that is, from disjoint subsets.

For the single pass hard c means experiment (SP), each time the number of examples loaded is equal to the number of examples present in a subset. For example, for Iris 3S, which means it was divided into 3 subsets, SP will load exactly 1/3 of the examples into memory at a time. Table 3.6 shows the difference in quality of BM, MM , and MCLA compared to the average quality of the single pass hard-k-means algorithm (SP).

The difference in quality in percentage is defined as follows:

$$DQ\_BM\_SP = \frac{(BM - SP)}{SP} * 100 \tag{3.13}$$

$$DQ\_MM\_SP = \frac{(MM - SP)}{SP} * 100 \tag{3.14}$$

$$DQ\_MCLA\_SP = \frac{(MCLA - SP)}{SP} * 100 \tag{3.15}$$

where BM, MM, and MCLA are the SE values in the columns of Table 3.1 and SP is the quality value of the third column in Table 3.3. A negative value (italicized) in Table 3.6 indicates that the algorithm in that column is better compared to SP.

Out of 10 experiments, MM is 7 times better than SP followed by MCLA, 6 times, and BM, 4 times. Thus ensemble based approaches are not always better than SP; nevertheless, results indicate that all multiple clustering algorithms, on average over all data sets and experiments, are better compared to SP. Average improvement of MCLA, compared to SP, is 2.85%, MM is 2.67%, and for BM is 2.40%. The difference in performance between MCLA and MM is 2.85-2.67=0.18 % and between MCLA and BM 2.85-2.40=0.45 %, which shows they are comparable.

Table 3.5 Difference in quality of BM, MM, and MCLA compared to BC (formed from hard c means). All values expressed in percentage. Italicized indicates an algorithm better than BC on an experiment.

	DQ_BM_BC %	DQ_MM_BC %	DQ_MCLA_BC %
Iris, 3S	*-6.709048*	*-3.655352*	*-4.597571*
Iris, 5S	*-10.498883*	*-12.392299*	*-11.615786*
Isolet6, 5S	*-0.435805*	*-0.227847*	*-0.757762*
Isolet6, 10S	*-0.193999*	*-1.838217*	*-2.072638*
Pen Digits, 5S	1.450805	0.167966	1.096767
Pen Digits, 10S	1.882967	*-3.237341*	*-1.673788*
KDD, 15S	1.626709	2.608278	1.064225
Plankton, 15S	1.486153	4.803150	2.252613
MRI-1, 20S	*-0.103254*	*-0.480625*	*-0.466616*
MRI-2, 20S	1.276897	1.615588	2.204793
average	-1.021746	-1.263670	-1.456576

In Table 3.7 the average speed up of our centroid based ensemble algorithms, BM and MM, compared to the label vector based algorithm, MCLA, is shown. We excluded Iris as it is too small to benefit. The results in Table 3.7 show that the speed-up from BM and MM was very large on medium or large data sets. On the MRI-2 data set, which contains about 4 million examples, BM was more than six hundred thousand times faster than MCLA, while MM was over two hundred thousand

Table 3.6 Difference in quality of BM, MM, and MCLA compared to SP (formed from hard c means). All values expressed in percentage. Italicized indicates an algorithm better than SP on an experiment.

	DQ_BM_SP %	DQ_MM_SP %	DQ_MCLA_SP %
Iris, 3S	*-17.489960*	*-14.789157*	*-15.622490*
Iris, 5S	*-14.274070*	*-16.087621*	*-15.343861*
Isolet6, 5S	*-0.445296*	*-0.237358*	*-0.767223*
Isolet6, 10S	0.337947	*-1.315035*	*-1.550705*
Pen Digits, 5S	0.868265	*-0.407208*	0.516260
Pen Digits, 10S	1.905949	*-3.215514*	*-1.651608*
KDD, 15S	1.754159	2.736959	1.190970
Plankton, 15S	2.201049	5.541412	2.972908
MRI-1, 20S	*-0.131009*	*-0.508275*	*-0.494270*
MRI-2, 20S	1.228423	1.566952	2.155875
average	-2.404454	-2.671484	-2.859414

times faster. It seems from the results that BM and MM have almost constant time complexity, that is, independent of data set size, while with MCLA it grows with data size. Theoretical time complexity analysis will be discussed later. It should be noted that for MRI-2, the largest data set, MCLA on average took about 3.84 hours, while BM and MM took only 0.02 and 0.06 seconds respectively. Looking at the contrast, it seems MCLA and other similar label vector based multiple partitioning algorithms, having similar time complexity, might not be scalable for large or very large data sets.

3.5.2 Fuzzy Ensemble Experiments

In this section the base partitions, the ensemble, were generated using the fuzzy-k-means algorithm. We conducted comparison experiments similar to those with the hard ensembles. That is, the 3 multiple partition combining algorithms will be compared among themselves and with GC and BC also. The difference in

Table 3.7 Time computed in seconds. Speed up of BM and MM compared to MCLA (formed from hard c means). SU-BM, SU-MM mean the speed up using Bipartite merger or MM respectively.

	MCLA	BM	MM	SU-BM	SU-MM
Isolet6, 5S	0.5572	0.0062	0.0562	89.87	9.91
Isolet6, 10S	1.3232	0.0148	0.1244	89.40	10.63
Pen Digits, 5S	0.6784	0.0052	0.0138	130.46	49.15
Pen Digits, 10S	2.8664	0.0056	0.0298	511.85	96.18
KDD, 15S	223.63	0.0410	0.0798	5454.39	2802.38
Plankton, 15S	1424.38	0.0302	0.0760	47164.90	18741.84
MRI-1, 20S	3745.35	0.0214	0.0626	175016.35	59829.87
MRI-2, 20S	13840.25	0.0222	0.0634	623434.68	218300.47

quality (DQ) of the fuzzy results is evaluated using the same formulas as used in the hard-ensemble experiments, equations (3.5) to (3.12). The only difference from the hard ensemble experiments is that we do not compare the 3 multiple partition combining algorithms, BM, MM, and MCLA, with SP because SP is a variant of the hard-k-means algorithm. So, we believe, comparing fuzzy ensemble results with it is not appropriate.

As MCLA can only merge partitions in the form of label vectors, a crisp partition of the fuzzy centroids in the ensemble was obtained after assigning an example to the nearest centroid. For BM and MM, the fuzzy centroids in the ensemble were merged to obtain k global centroids. A crisp partition was then obtained by assigning an example to the nearest centroid. The quality of MCLA, BM, and MM was then evaluated using the SE metric.

Table 3.8 shows the quality in SE metric of the 3 multiple partition combining algorithms, BM, MM, and MCLA. Out of 10 sets of experiments over all data sets, both MM and MCLA were best 4 times and BM 2 times; it seems for fuzzy ensembles

both MM and MCLA performed better compared to BM. The best SE values (lowest) are italicized in the table. In Table 3.9, the differences in quality of BM and MM from MCLA were computed. Table 3.9 shows that out of the 10 sets of experiments, MM was better 5 times and BM only 3 times. When BM or MM was better compared to MCLA in the table, the values are italicized. It is interesting to note that unlike the hard-ensemble case, MM on average, over all data and experiments, was better compared to MCLA, by 0.20 %, while MCLA is slightly better than BM by 0.44%; however, the quality difference among them using fuzzy ensembles was also small.

Table 3.8 Quality, computed in squared-error (SE) criterion, of Bipartite Merger (BM), Metis Merger (MM), and Meta Clustering algorithm (SP) computed using the SE metric. Base clustering formed from fuzzy-k-means. Italicized indicates the lowest SE value in an experiment.

	BM	MM	MCLA
Iris, 3S	*80.48*	80.67	80.52
Iris, 5S	79.44	79.63	*79.38*
Isolet6, 5S	83349.57	83817.75	*83007.61*
Isolet6, 10S	83764.54	83763.36	*83590.10*
Pen Digits, 5S	15113629.69	*14964032.98*	15091708.56
Pen Digits, 10S	15320268.38	*14797649.95*	14804901.11
KDD, 15S	4043089.75	4048920.09	*4040212.63*
Plankton, 15S	8671138.59	*8441119.13*	8652913.68
MRI-1, 20S	5113693711.14	*5083170309.75*	5116494014.66
MRI-2, 20S	*29665330823.85*	29665330823.86	29666026271.38

Table 3.10 shows the average quality of GC and BC experiments. Table 3.11 shows the quality difference in percentage of BM, MM, and MCLA compared to GC. The cases when BM or MM or MCLA were better (negative number) compared to GC, are italicized. From the results, GC was better compared to all the multiple partition combining algorithms, BM, MM, and MCLA, for most cases. Out of 10

cases, MM was better than GC in 2 cases only, MCLA in 1 case, and BM in none. The average results over all data sets and experiments are shown in Table 3.11 also reflect that, GC was better than MM by 0.64%, MCLA by 0.85%, and BM by 1.30%; however, among the three MM was the best, closest to GC, followed by MCLA and BM. So, global fuzzy clustering seems to be quite robust when compared to global hard clustering.

Table 3.9 Difference in quality of BM and MM compared to MCLA (formed from fuzzy c means). All values expressed in percentage. Italicized indicates an algorithm better than MCLA on an experiment.

	DQ_BM_MCLA %	DQ_MM_MCLA %
Iris, 3S	*-0.049677*	0.186289
Iris, 5S	0.075586	0.314941
Isolet6, 5S	0.411962	0.975983
Isolet6, 10S	0.208685	0.207273
Pen Digits, 5S	0.145253	*-0.845998*
Pen Digits, 10S	3.481059	*-0.048978*
KDD, 15S	0.071212	0.215520
Plankton, 15S	0.210622	*-2.447667*
MRI-1, 20S	*-0.054731*	*-0.651300*
MRI-2, 20S	*-0.002344*	*-0.002344*
average	0.449763	-0.209628

When compared to BC in Table 3.12, it was observed that out of 10 cases both MM and MCLA were better than BC in 6 cases, and BM better than BC in 3 cases only. On average, over all data sets and experiments, all three multiple partition algorithms are better than BC. The best is MM followed by MCLA and BM. The difference in performance among the three, compared to BC, was small.

We also empirically evaluated the average speed up from BM and MM compared to MCLA. Similar to the hard ensemble experiments, the results in Table 3.13 show

Table 3.10 Quality, computed in squared-error (SE) criterion, of Global clustering (GC) and average base clustering solutions (BC). GC and BC formed from fuzzy-k-means.

	GC	BC
Iris, 3S	79.11	82.05
Iris, 5S	79.11	86.40
Isolet6, 5S	82774.46	83223.59
Isolet6, 10S	82774.46	83596.88
Pen Digits, 5S	14872713.34	15018288.06
Pen Digits, 10S	14872713.34	15134076.22
KDD, 15S	3991204.98	3990866.15
Plankton, 15S	8565044.10	8543395.94
MRI-1, 20S	5024958474.14	5039282201.40
MRI-2, 20S	29648952131.97	29666105675.18

Table 3.11 Difference in quality of BM, MM, and MCLA compared to GC (formed from fuzzy c means). All values expressed in percentage. Italicized indicates an algorithm better than GC on an experiment.

	DQ_BM_GC %	DQ_MM_GC %	DQ_MCLA_GC %
Iris, 3S	1.731766	1.971938	1.782328
Iris, 5S	0.417141	0.657313	0.341297
Isolet6, 5S	0.694792	1.260401	0.281669
Isolet6, 10S	1.196118	1.194692	0.985376
Pen Digits, 5S	1.619855	0.614008	1.472463
Pen Digits, 10S	3.009236	*-0.504705*	*-0.455951*
KDD, 15S	1.299978	1.446057	1.227891
Plankton, 15S	1.238692	*-1.446869*	1.025909
MRI-1, 20S	1.765890	1.158454	1.821618
MRI-2, 20S	0.055242	0.055242	0.057588
average	1.302871	0.640653	0.854019

that speed-up for BM and MM was very large, that is, of the order of hundreds of thousands times.

Table 3.12 Difference in quality of BM, MM, and MCLA compared to BC (formed from fuzzy c means). All values expressed in percentage. Italicized indicates an algorithm better than BC on an experiment.

	DQ_BM_BC %	DQ_MM_BC %	DQ_MCLA_BC %
Iris, 3S	*-1.913467*	*-1.681901*	*-1.864717*
Iris, 5S	*-8.055556*	*-7.835648*	*-8.125000*
Isolet6, 5S	0.151375	0.713932	*-0.259518*
Isolet6, 10S	0.200558	0.199146	*-0.008110*
Pen Digits, 5S	0.634837	*-0.361260*	0.488874
Pen Digits, 10S	1.230284	*-2.222972*	*-2.175059*
KDD, 15S	1.308578	1.454670	1.236485
Plankton, 15S	1.495221	*-1.197145*	1.281899
MRI-1, 20S	1.476629	0.870920	1.532199
MRI-2, 20S	*-0.002612*	*-0.002612*	*-0.000268*
average	-0.347415	-1.006287	-0.789321

In summary, we compared the three multiple partition combining algorithms among themselves and with GC, BC, and SP (for hard ensemble experiments only). For hard ensemble experiments, among BM, MM, and MCLA, MCLA was slightly better than MM, and MM slightly better than BM. For fuzzy ensemble experiments, among BM, MM, and MCLA, MM was slightly better than MCLA, and MM was slightly better than BM. For both hard ensemble and fuzzy ensemble experiments, compared to GC, BC, and SP (for hard ensembles only), centroid based combining algorithms and label vector based algorithms were comparable.

Looking at the time taken and speed up from using the centroids based algorithms, BM and MM, it seems that for very large or extreme data sets label vector based approaches may not be scalable. It is true that in many cases base clusterings in

Table 3.13 Time computed in seconds. Speed up of BM and MM compared to MCLA (fuzzy ensemble). SU-BM, SU-MM mean the speed up using Bipartite merger or MM respectively.

	MCLA	BM	MM	SU-BM	SU-MM
Isolet6, 5S	0.5612	0.0054	0.0566	103.92	9.91
Isolet6, 10S	1.3250	0.0144	0.1270	92.01	10.43
Pen Digits, 5S	0.6794	0.0056	0.01420	121.32	47.84
Pen Digits, 10S	2.8654	0.0068	0.0304	421.38	94.25
KDD, 15S	223.49	0.0420	0.0762	5321.19	2932.93
Plankton, 15S	1424.83	0.0310	0.0796	45962.25	17899.87
MRI-1, 20S	3741.78	0.0194	0.0632	192875.25	59205.37
MRI-2, 20S	13847.10	0.0212	0.0648	653165.09	213689.81

the form of a centroid vector may not exist (nominal features); however, there are many domains for which all features are numeric. For example, besides many generic data sets, image processing and many other multispectral imaging domains, including Magnetic resonance and satellite imaging almost always have only numeric features. We also compared the quality of multiple partitioning combining algorithms with a single pass algorithm (SP) to study the pros and cons of both frameworks. The results indicate that ensemble based algorithms, BM, MM and MCLA, on average over all data sets and experiments were better than single pass hard-k-means. The difference in quality with single pass was not very much, indicating single pass algorithms are a potentially good candidate for scaling to large data sets. So, we examined both scalability and quality issues of multiple partition combining algorithms for large or very large data sets by experimenting on 7 real data sets from various domains. We have shown that our centroid based ensemble merging algorithms are better or as good as a label vector based ensemble merging algorithm, MCLA, while being scalable. It shows label vectors, which are a high resolution representation of a clustering solution

containing a label for every example in a data set, do not have much advantage over a centroid based representation. A centroid based representation is likely to be a low resolution representation as number of clusters is typically much less than number of examples in large or very large data sets.

3.6 Performance Under Malformed Centroids and the Effect of Filtering

In all experiments, we assumed the data were randomly divided into equal size subsets i.e. each subset contains a uniform/slowly changing distribution from all classes. But it might not always be true, especially when data is already clustered. In this case we have to provide a knowledge reuse framework. Skewed distributions may also exist in physically distributed data, and there may be no chance of homogenizing the distributed sites due to privacy preserving issues [2, 33, 34]. Moreover, in some cases one or more clustering solutions in the ensemble may be noisy. We created a type of unbalanced distribution using the 3 small data sets Iris, Pen Digits, and Isolet as shown in Tables 14 to 16 respectively. We chose these data sets because they were small and hence easy to analyze plus they have true labels. All three data sets were divided into approximately equal sized subsets; r=5 for Iris, r=10 for Pen Digits, and r=10 for Isolet6. For all the data sets, 80% of the subsets, 4 subsets for Iris, 8 for Pen Digits and isolet6, have examples from at least one class missing and have an unbalanced distribution of examples, a kind of skewed distribution. There are various ways of creating an unbalanced distribution. We arbitrarily created one for each data set. The purpose is to provide an understanding of the performance of the algorithms in one type of noisy setting (malformed centroids). Figures 4 to 6 show the average results from 50 random experiments on Iris, Pen Digits, and the Isolet6 data sets respectively. Experimental results on ensembles formed from both hard-k-means and fuzzy-k-means are provided. In the figures, BM" means the ensemble

was merged using the Bipartite Merger with filtering options turned off, MM" means Metis merger with filtering option turned off, while BM and MM means as usual the filtering option was on. MCLA has no notion of filtering, so it is unchanged. We compared the results with BC and GC. With the filtering option turned on, the quality of BM and MM improved in all experiments, justifying the concept of filtering. Because the distribution was heavily skewed, as expected the average SE value of BC is poor. In all hard ensemble experiments, among the 3 multiple partition combining algorithms, Metis merger was the best on all data sets. It managed to recover good partition quality, even better than GC on Iris, with the filtering option turned on. This is not a trivial accomplishment because the distribution is heavily skewed, which is indicated by the poor SE value of BC. For fuzzy ensemble experiments, MM was the best except on Isolet6, where MCLA was slightly better than MM. BM with the filtering option turned on managed to get quality better than BC on all data sets; however, it performed poorly, except on Iris, compared to MM and MCLA. This might be because 80% of the subsets had an unbalanced distribution, which is not a favorable condition for grouping centroids using a one to one mapping constraint. BM optimally matches centroids between a pair of partitions, that is, proceeds by providing best matches locally. In many cases, this type of one to one constrained mapping might be useful, especially for mapping clusters of partitions whose distribution is slowly changing/evolving. For example, a one to one mapping constraint was used in [5] for addressing the clustering problem in perceptual organization, which was used to provide additional knowledge about a 3D object. They proposed path based clustering to group smooth curves and textured segmentations.

It should be noted that even without filtering, the quality of MM was always better (except on fuzzy ensemble of isolet6) than MCLA.

Table 3.14 Unbalanced distribution of Iris data set into 5 subsets. S1, S2 ,.., S5 are the subsets.

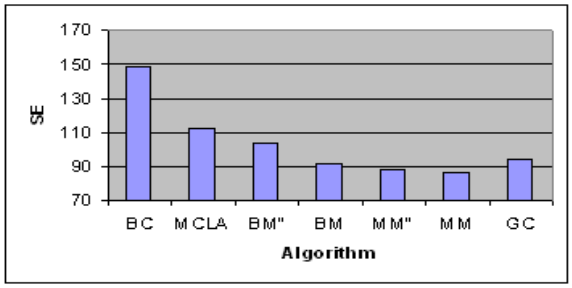
	S1	S2	S3	S4	S5
Class1	10	0	20	10	10
Class2	10	20	0	0	20
Class3	10	10	10	20	0

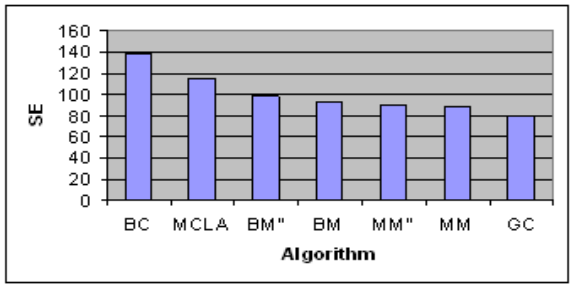
Table 3.15 Unbalanced distribution of Pen Digit data set into 10 subsets. S1, S2,.., S10 are the subsets.

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Class1	35	35	0	35	35	70	35	5	65	48
Class2	35	35	70	35	35	0	35	65	5	49
Class3	35	35	35	0	35	35	70	35	35	49
Class4	35	35	35	35	35	35	35	35	35	21
Class5	35	35	35	70	0	35	0	70	35	49
Class6	35	35	35	35	35	35	35	35	35	20
Class7	35	35	5	35	35	65	35	35	0	56
Class8	35	35	35	35	70	35	35	0	70	14
Class9	35	35	65	65	35	5	5	35	56	0
Class10	34	34	34	4	34	34	64	34	13	51

Table 3.16 Unbalanced distribution of Isolet6 data set into 10 subsets. S1, S2,.., S10 are the subsets.

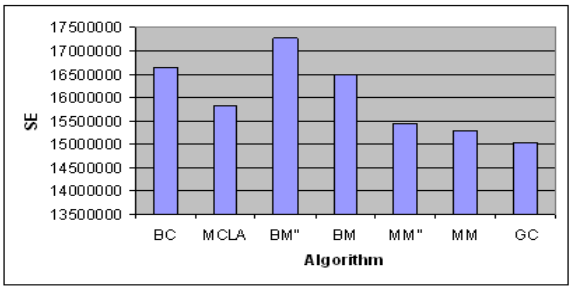
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Class1	24	24	0	24	4	48	24	44	48	0
Class2	24	24	24	0	44	24	48	4	24	24
Class3	24	24	48	24	24	0	24	24	0	48
Class4	24	24	24	48	24	24	0	24	24	24
Class5	24	24	4	24	48	44	24	0	24	24
Class6	24	24	44	24	0	4	24	48	24	24

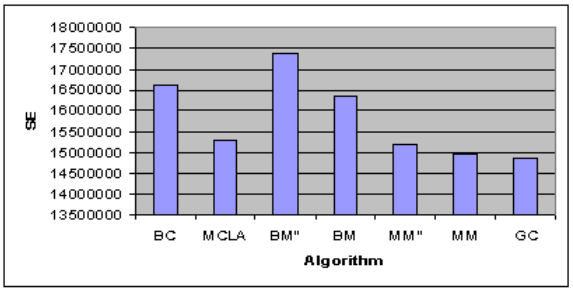




(a) (b)

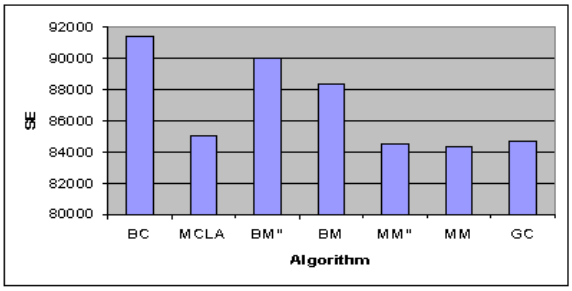
Figure 3.4 Experiment on unbalanced Iris data set. Experimental results both from (a) hard-k-means and (b) fuzzy-k-means are provided.

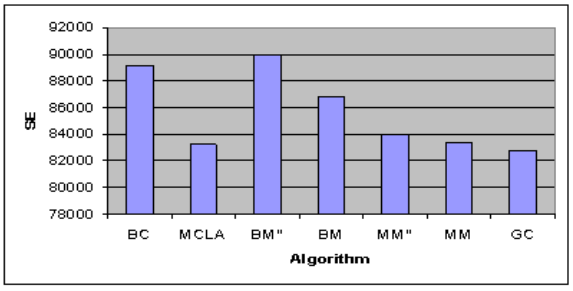




(a) (b)

Figure 3.5 Experiment on unbalanced Pen Digits data set. Experimental results both from (a) hard-k-means and (b) fuzzy-k-means are provided.





(a) (b)

Figure 3.6 Experiment on unbalanced Isolet6 data set. Experimental results both from (a) hard-k-means and (b) fuzzy-k-means are provided.

3.7 Analysis of Time and Space Complexity

We analyze the time complexity of the multiple partition combining algorithms. Let n be the number of examples in the whole data set, k the number of clusters, and r the number of base clusterings/partitions in the ensemble. Then, the time complexity of MCLA is $\mathrm{O}(nk^2r^2)$ [2]. For Bipartite Merger (BM) and Metis Merger (MM), centroids were used to represent the ensemble, thus the time complexity of our algorithms is free of n. The time complexity for BM is $\mathrm{O}(rk^2f + rk^3 + kr^2)$, where f is the number of features in the data, and for MM it is $\mathrm{O}(r^2k^2f + kr^2)$. The time complexity of other related label vector based multiple partition combining algorithms is also of interest. For SCEC it is $\mathrm{O}(tnr^2k)$, where t is the number of iterations required in SCEC and CSPA is $\mathrm{O}(n^2kr)$ [2]. It has been mentioned in [3] the QMI and MMEC have the same time complexity as SCEC. It seems that CSPA will be slower than MCLA as it depends on n^2. SCEC, hence QMI and MMEC, will be faster than MCLA if t, the number of iterations required in their algorithm, is less than k, which is the number of clusters. So, all have n in their time complexity equation, except our BM and MM.

We also analyze the space complexity of the representation of the two types of ensemble, that is, centroid based and label vector based. Assuming hyperedges are loaded in memory, the space complexity of the label vectors based algorithm, MCLA, is $\mathrm{O}(knr)$. For BM and MM it is free from n, that is, $\mathrm{O}(kfr)$, assuming all centroids are loaded in memory for merging. We did an empirical evaluation using our largest size data set, MRI-2. It contains 3,991,592 examples, 3 features, 9 clusters, and was divided into 20 subsets, that is, ensemble size r=20. For representing hyperedges, an indicator vector, in MCLA, we used char type memory allocation (1 byte). In MM and BM, floating point precision (4 bytes) was used to store cluster

centroids. For MCLA, 9*20=180 hyperedges were kept in memory for efficient construction of the meta-graph. So, the total memory requirement to store the hyperedges are 180*3991592*1=718,486,560 bytes (approx 685 MB). To keep the ensemble of centroids in memory, we need 9*3*20*4 bytes, only 2160 bytes. Thus the memory requirement of the label vector based algorithm, MCLA, is more than three hundred thousand times bigger, that is, 332,632.66. In a wide area network scenario, which might occur for merging clustering solutions of physically distributed data, transferring centroids would also be efficient in terms of network bandwidth cost. Thus, the centroid based ensemble algorithms not only provide a speed-up of hundreds of thousands of times but also require hundreds of thousands of times less memory compared to a label vector based algorithm, MCLA.

Compared to label vector based ensemble merger algorithms, our algorithms should scale extremely well, in terms of time and space complexity. The time complexity equations of our algorithms do not directly depend upon the size of the data, and for most cases the number of centroids is not likely to be large. Even for very large dimensional data sets if k remains relatively small our algorithm will be fast, provided f is not as large as the size of data. Because in many cases the number of clusters, k, size of ensemble, r, and the number of dimensions, f, are generally much smaller than the size of the data, the time complexity equations of Bipartite Merger and Metis Merger will simple reduce to $O(1)$, that is, constant time. Tables 3.7 and 3.13 also show that the time taken by Bipartite Merger and Metis Merger do not increase significantly with the size of the data, indicating approximately constant running time algorithms.

3.8 Sensitivity Estimation

Because the multiple partition combining algorithms are heuristic solutions to the r-partite graph partitioning problem, they may be sensitive to the order of selecting partitions in the ensemble. In [2], it was not stated that MCLA could be sensitive to the order in which the hyperedges, actually the vertices, are numbered in the meta graph. Order matters because the METIS graph partitioning package is a randomized algorithm. In this section some experiments are conducted to estimate sensitivity of the multiple partitions combining algorithms.

Each experiment with multiple combining algorithms consisted of 50 random initializations, where the order of selecting partitions was random for each merging operation. We re-ran the experiment 32 times with the same set of centroids obtained from 50 random initializations; each time the centroid combination order was random to estimate how sensitive the algorithms were to the order of random selection. Sensitivity was determined by computing the average of the absolute difference of quality in SE of each experiment from its mean. As SE values vary by data sets, we normalized by dividing by the mean and then multiplying by 100 to obtain a percentage. This will indicate how much the quality of partitions could vary from their mean quality on average. Sensitivity is computed as:

$Sensitivity = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{|p_i - m_i|}{m_i}\right) * 100$, where n is 32, p_i is the average quality in SE of each experiment, and $m_i = \sum_{i=1}^{n} p_i$.

Because each of the 32 experiments involves 50 random initializations, we used the 3 smaller data sets Iris, Isolet6, and Pen Digits for the sensitivity estimation experiments. The chosen value of r, the ensemble size, was a maximum for each of the data sets, that is, 5 for Iris, 10 for Isolet6, and 10 for Pen Digits. Tables 3.17 and 3.18 show the results expressed in percentage for the multiple partition combining

algorithms on each data set both for hard and fuzzy ensembles, respectively. The average sensitivity of the algorithms over all data sets is also given. On average, for both on hard and fuzzy ensemble experiments, BM was the most sensitive compared with MM and MCLA. On the hard ensemble experiments, MCLA was the least sensitive, while on fuzzy ensemble experiments MM was the least sensitive. From the tables, with the Iris data BM seemed to be much more sensitive compared to MM and MCLA; otherwise, all the three algorithms do not seem to have much difference among them. On average the sensitivity of all the three algorithms was low, that is, for MM and MCLA much less than 0.5%, while for BM slightly above 0.5%.

Table 3.17 Sensitivity experiments of BM, MM, and MCLA on Hard Ensembles. All values expressed in percentage change of SE.

	Iris, 5S (%)	Isolet6, 10S (%)	Pen Digits, 10S (%)	Average (%)
BM	1.304684	0.308255	0.453086	0.688675
MM	0.361843	0.145923	0.210061	0.239275
MCLA	0.096984	0.148035	0.369641	0.204886

Table 3.18 Sensitivity experiments of BM, MM, and MCLA on Fuzzy Ensembles. All values expressed in percentage change of SE.

	Iris, 5S (%)	Isolet6, 10S (%)	Pen Digits, 10S (%)	Average (%)
BM	1.139955	0.181290	0.315357	0.545534
MM	0.106712	0.178516	0.174228	0.153152
MCLA	0.040850	0.176785	0.254943	0.157526

CHAPTER 4

SINGLE PASS FUZZY C MEANS ALGORITHM

In this chapter we propose a modified fuzzy c means algorithm for large or very large data sets, which will produce a final clustering in a single pass through the data with limited memory allocated. We neither keep any complicated data structures nor use any complicated compression techniques. We will show that our simple single pass fuzzy c means algorithm (SPFCM) will provide quality (by loading as little as 1% of the data) which is highly similar to that obtained by clustering all the data at once using fuzzy c means (FCM). Moreover, we will also show our single pass fuzzy c means algorithm provides significant speed up even when compared with clustering a complete data set, which is less than the size of the memory.

4.1 Single Pass Fuzzy C means Algorithm Details

Suppose we intend to cluster a large or very large data set present on a disk. We assume for large or very large data sets that the data set size exceeds the memory size. As in [21], we assumed the data set is randomly scrambled on the disk. We can only load a certain percentage of the data based on the available memory allocation. If we load 1% of the data into memory at a time then we have to do it 100 times to scan through the entire data set. We call each such data access a partial data access (PDA). The number of partial data accesses will depend on how much data we load each time. In our approach, after the first PDA, data is clustered into c partitions using fuzzy c means. We note here that once in memory data may be

accessed multiple times. After clustering, the data in memory is condensed into c weighted points and clustered with new points loaded in the next PDA. We call them "weighted" points because they are associated with weights, which are calculated by summing the membership of examples in a cluster. This is the key difference from the crisp clustering case [21], where a fuzzy membership matrix is not present. In each PDA new singleton points are loaded into memory and clustered along with the past c weighted points obtained from the previous clustering. We will call this partial data clustering (PDC). After clustering these new singleton points along with the past c weighted points, they are condensed again into c new higher weighted points and clustered with examples loaded in the next PDA. This continues until all the data has been scanned once. The objective function of fuzzy c means was modified in a fashion similar to that in [22] to accommodate the effect of weights. We will discuss the calculation of weighted points and the modification of the objective function of fuzzy c means in detail later.

As an example, consider a large or very large data set of n examples. If n_1 examples are fetched in the first PDA and clustered into c partitions then all these n_1 examples in memory are condensed into c weighted points, whose weights will sum up to n_1. Condensation of n_1 examples into c weighted points frees the buffer. Next n_2 examples are loaded into memory in the next PDA. These new n_2 examples are then clustered along with the c weighted points. So, after the second PDA there will be $n_2 + c$ examples in memory for clustering, out of which c are weighted points and n_2 examples have weight one (singletons). We will call the new fuzzy c means which takes into account the weights of c weighted points the weighted fuzzy c means (WFCM). After clustering these $n_2 + c$ examples in memory using WFCM they are condensed again into c new weighted points. This time the weight of the c points sum up to $n_1 + n_2$ and thus they have more weight than before. This is because there were

already c weighted points, summed up to n_1, present when n_2 new singleton examples were loaded in the second PDA. Similarly, after completion of clustering in the third PDA, the weight of the new condensed c points will sum up to $n_1+n_2+n_3$. This means after the m^{th} PDA there will be n_m singleton points loaded in the memory along with c weighted points from a previous PDC, whose weights sum up to $n_1+n_2+n_3+...+n_{m-1}$. So, if the last PDA loads n_l examples, it essentially clusters the whole data set, where $n - n_l$ examples remain as c condensed weighted points and n_l as singleton points. Thus, our simple single pass fuzzy c means will partition the whole data in a single pass through the whole data set. To speed up clustering, we initialize each PDC with the final centroids obtained from the previous PDC. This knowledge propagation allows for faster convergence.

4.1.1 Weighted Point Calculation

The objective function (J_m) minimized by FCM is defined as follows:

$$J_m\left(U,V\right) = \sum\nolimits_{i=1}^{c}\sum\nolimits_{k=1}^{n} U_{ik}^{m} D_{ik}(x_k, v_i) \tag{4.1}$$

$U$and V can be calculated as:

$$U_{ik} = \frac{D_{ik}\left(x_k, v_i\right)^{\frac{1}{1-m}}}{\sum_{j=1}^{c} D_{jk}\left(x_k, v_j\right)^{\frac{1}{1-m}}} \tag{4.2}$$

$$v_i = \frac{\sum\limits_{j=1}^{n}\left(u_{ij}\right)^m x_j}{\sum_{j=1}^{n}\left(u_{ij}\right)^m} \tag{4.3}$$

Where,

U_{ik} : is the membership value of the k^{th} example, x_k, in the i^{th} cluster.

v_i : is the i^{th} cluster centroid.

n : is the number of examples

c: is the number of clusters

$D_{ik}(x_k, v_i) = \|x_k - v_i\|^2$: is the norm. We have used the Euclidean distance.

We will discuss weighted point calculation in this section. We will assume we have a weighted FCM algorithm (WFCM) that takes into account the weights of examples and clusters data into c partitions. The details of WFCM will be discussed in the next section.

Consider n_d examples which are loaded in memory in the d^{th} PDA.

4.1.1.1 Case1: d=1

If d is equal to one i.e. the first PDA, there will be no previous weighted points. In this case WFCM will be the same as FCM. After applying clustering, let v_i be the cluster centroids obtained, where $1 \leq i \leq c$. Let u_{ij} be the membership values, where $1 \leq i \leq c$ and $1 \leq j \leq n_d$. Let $\overrightarrow{w}$ be the weights of the points in memory. In this case all n_d points have weight 1 because no weighted points from the previous PDC exist. Now, memory is freed by condensing the clustering result into c weighted points, which are represented by the c cluster centroids v_i, where $1 \leq i \leq c$ and their weights are computed as follows:

$$w_i^{'} = \sum_{j=1}^{n_d} (u_{ij})\, w_j,$$

$$1 \leq i \leq c, \quad w_j = 1, \forall 1 \leq j \leq n_d.$$

The weight of the c points, after condensing the clustering results, in memory is as follows:

$$w_i = w_i^{'}, \quad 1 \leq i \leq c.$$

It should be noted that when n_d new singleton points (weight one) are loaded in all subsequent PDA (d>1), their indices associated with $\overrightarrow{w}$ will begin at $c+1$ and end at $n_d + c$ i.e.

$$w_j = 1, \forall c < j \leq n_d + c.$$

4.1.1.2 Case2: d>1

In this case, clustering will be applied on singleton points freshly loaded in the d^{th} PDA along with c weighted points obtained after condensation from the $(d-1)^{th}$ PDC. So, there will be $n_d + c$ points in memory for clustering using WFCM. The new n_d singleton points have weight one. After clustering, the data in memory (both singletons and weighted points) are condensed into c new weighted points. The new weighted points are represented by the c cluster centroids v_i, where $1 \leq i \leq c$ and their weights are computed as follows:

$$w_i^{'} = \sum_{j=1}^{n_d+c} (u_{ij})\, w_j, \quad 1 \leq i \leq c.$$

Then memory is freed up and the weight of the condensed clustering results in memory is updated as follows:

$$w_i = w_i^{'}, \quad 1 \leq i \leq c.$$

4.1.2 Weighted FCM (WFCM)

We modified the objective function of FCM (similar to [22]) to take into effect the weighted points. It must be noted that except for the first PDC all subsequent clustering will have c weighted points along with freshly loaded examples. Let us consider that n_d examples (weight one) are loaded from the disk in the d^{th} PDA and there are c weighted points obtained from condensation of examples after the previous

PDC and their union constitutes the set $X^{'}$. The cluster centroids and membership matrix for the WFCM are calculated as:

$$v_i = \frac{\sum_{j=1}^{n_d+c} w_j (u_{ij})^m x_j^{'}}{\sum_{j=1}^{n_d+c} w_j (u_{ij})^m}, \quad 1 \leq i \leq c, \quad x_j^{'} \in X^{'}.$$

The weight of each of the n_d examples loaded will be 1. The c weighted points have weight calculated from condensation after the previous PDC.

$$u_{ij} = \left[\sum_{l=1}^{c} \left(\frac{\left\| x_j^{'} - v_i \right\|}{\left\| x_j^{'} - v_l \right\|} \right)^{\frac{2}{m-1}} \right]^{-1}, \quad 1 \leq i \leq c \text{ and } 1 \leq j \leq n_d + c.$$

It should be noted that if we have enough memory to load all the data then our simple single pass fuzzy clustering algorithm will reduce to FCM.

4.2 Data Sets

There were nine real data sets used. They are Iris, ISOLET6, Pen Digits, KDD, Plankton, and four magnetic resonance image data sets (MRI-1, MRI-3, MRI-4, and MRI-5). The Iris, ISOLET6, Pen Digits, KDD, Plankton, and MRI-1 dat sets were discussed in Chapter 3. The Iris data set was clustered into 3 clusters, ISOLET6 into 6, Pen Digits into 10, KDD into 10, Plankton into 12, and MRI-1 into 9. Below we describe the three other data sets, MRI-3, MRI-4, and MRI-5.

- *MRI-3 Data.* This data set was created by concatenating 2 Volumes of human brain MRI data. Each Volume consists of 144 slices of MR images of size 512X512 from modalities T1, PD, and T2. The magnetic strength is 1.5 Tesla. For this data set air was not removed and the total size of data set is slightly above 75 million (75,497,472 examples, 3 features). We clustered this data set into 10 clusters.

- *MRI-4 Data.* This data set was created by concatenating 96 slices of MR images, T1 weighted, of size 512X512 from a single human brain. The magnetic field strength was 1.5 Tesla. After air and skull were removed using the brain extraction tool (BET2) [80], there were 3,621,971 examples. The code for the BET2 is available at http://www.fmrib.ox.ac.uk/analysis/research/bet/. We clustered this data set into 3 clusters.

- *MRI-5 Data.* This data set was created by concatenating 144 slices of MR images, T1 weighted, of size 256X256 from a single human brain. The magnetic field strength was 3 Tesla. After air and skull were removed using the brain extraction tool (BET2) [80], there were 1,248,595 examples. Intensity homogeneity on this data set was corrected using an implementation of the bias correction algorithm in [81]. We clustered this data set into 3 clusters.

4.3 Experimental Setup

In [82], a reformulated optimization criteria R_m , which is mathematically equivalent to J_m (equation 4.1) was given.

$$R_m(V) = \sum_{k=1}^{n} \left(\sum_{i=1}^{c} D_{ik}(x_k, v_i)^{\frac{1}{(1-m)}} \right)^{(1-m)} \tag{4.4}$$

The new formulation has the advantage that it does not require the U matrix and can be directly computed from the final cluster centroids. For large data sets, where all the data cannot be loaded into memory, R_m can be computed by incrementally loading examples from the disk. The values of m used for fuzzy clustering were $m = 2$ for Iris, Plankton, MRI-1, MRI-3, MRI-4, and MRI-5, $m = 1.2$ for ISOLET6

and KDD, and $m = 1.5$ for the Pen digits data set. The different values enabled reliable partitions with the full data to be obtained.

Results on Iris, ISOLET6, Pen Digits, MRI-1 were averaged over 50 experiments, each starting with a random initialization. Due to the computational time involved in global clustering, results on the data sets KDD, Plankton, MRI-4, and MRI-5 were averaged over 32 random experiments. The MRI-3 data set had examples over 75 million, so results on this data set were averages over 10 random experiments only.

Both FCM and SPFCM were initialized with the same centroids. Each data set was clustered using FCM, loading all the data into memory, and single pass fuzzy c means (SPFCM), loading a chosen percentage of data into memory. For measuring quality, we compute the mean R_m value of FCM and SPFCM. Generally this is the criteria minimized in FCM. In single pass hard c means [21], a similar metric was used to compute quality, that is, the sum of the squared distance between each example and the cluster centroid to which it belongs. This criteria is minimized in hard c means. Both for SPFCM and FCM, the R_m value of the data set can be computed using the final V matrix (equation 4.4). We compare quality by computing the difference between the mean R_m value of FCM and SPFCM and then expressing it as a percentage. For example, if we denote the mean R_m value of experiments with FCM as m_1 and mean R_m value of experiments with SPFCM as m_2 then

Difference in Quality (DQ) is:

$$DQ = \left(\frac{m_2 - m_1}{m_1}\right) * 100 \tag{4.5}$$

So, a positive value means the average value of R_m of FCM is better (lower) while a negative value means R_m of SPFCM is better.

We also compare the speed-up obtained by SPFCM compared with FCM. For FCM, as stated earlier, we load the entire data set into memory before clustering. Thus the speed-up reported in this paper is the minimum speed up SPFCM can achieve compared to FCM. This is because for very large data sets the time required by FCM will become more and more due to disk accesses per iteration. Thus we will show that even if we have enough memory to load all the data SPFCM will be significantly faster than FCM while providing almost the same quality partition.

All experiments were run on UltraSPARC-III with 8 processors each of 750 MHz. There was 32GB of main memory. None of the programs were multithreaded, so each program used only one processor at a time.

4.4 Results and Discussion

In a single pass experiment, if we load n% of the data in each PDA, it is denoted by SPFCMn. Experimental results on the 8 real data sets are shown in Table 4.1. For small data sets i.e. Iris, Pen digits, and Isolet6 with SPFCM10 (10% data loaded), we got excellent quality partitions over all 50 experiments with different random initializations i.e. on average the difference in quality from FCM is 0.05%, 0.24%, 0.28% for Iris, Pen Digits, and Isolet6 respectively. We also performed experiments on these small data sets under a more stringent condition i.e. loading as little as 5% (loading only 8 of 150 examples each time) for Iris and 1% for Pen digits (loading only 35 of 3498 examples each time) and Isolet6 (loading only 15 of 1440 examples each time). As seen in Table 4.1 for Pen Digits and Isolet6 we observed acceptable differences on average of 4.80% and 3.18% respectively from FCM over 50 experiments. For Iris with 5% (SPFCM5) data loaded, the difference in quality compared to FCM is a little higher. After we examined the distribution of extrema, we found that out of 50 random experiments 42 (84%) are excellent partitions whose

difference in quality with mean R_m value of FCM is only 0.09% and the other 8 got stuck in some not so good extrema whose R_m value is high. We show this in Figure 4.1 and 4.2, which show the extrema distribution both for FCM (Figure 4.1) and SPFCM5 (Figure 4.2) on the Iris data set over the 50 random experiments. In Figure 4.1 and Figure 4.2, the x-axis indicates the experiment number and y-axis the R_m value. It clearly shows that FCM in all 50 experiments got one type of extremum whose R_m value is near 60, while SPFCM5 found 8 poorer partitions ($R_m$value near 105) and the other 42 were very similar R_m (about 60) values when compared to FCM. In Figure 4.3, we have visualized this poorer partition using 2 out of the 4 features of Iris. The two features used were petal length and petal width as they contain most of the information about the classes. It looks from Figure 4.3 that the two overlapped classes of Iris came out as a single cluster and the separable one got split into 2 clusters.

SPFCM appears quite stable even under stringent conditions, which are not very realistic. However, it can break if a "too small" percentage is loaded. In summary, on small data sets with 10% data loaded we got excellent partitions, with average quality almost the same as FCM, on all data sets and generally acceptable partitions under stringent conditions.

As our single pass fuzzy c means algorithm is mainly meant for large or very large data sets, results on the 3 magnetic resonance image data sets, MRI-1,MRI-4, and MRI-5, and the KDD and the Plankton data sets will help us better assess our algorithm. The results (Table 4.1) show that SPFCM achieved excellent partitions whose average quality difference (in percentage) with FCM on all random experiments was below 0.5%. So, we believe that for medium/large data sets 1% data loaded may be enough to get an excellent partition, that is, with average quality almost the same as FCM.

Table 4.2 indicates the average speed-up of SPFCM compared to FCM on the data sets. We excluded Iris as it is quite small. Although, SPFCM is meant for large scale data we reported speed-up for Pen Digits and Isolet6 also. Similar to [21], it seems in general loading and clustering 1% of the data (SPFCM1) is faster than loading 10% of the data (SPFCM10) in each PDA. It may be due to the reason that approximate model of the data is built in the initial few PDAs. So, loading and clustering 1% data in initial PDAs will be faster than loading 10% data. As shown in Table 4.2, large/medium size data sets achieved good speed-up with SPFCM1. On the small data sets, the speed up achieved was also significant. It should be noted that FCM loaded all data in memory, so the speed-up reported here for SPFCM is the minimum. For large data sets (assuming it won't fit in memory), FCM will require multiple disk accesses making it slower. The results indicate that SPFCM can be used for speeding up fuzzy c means even if the data can be loaded fully into memory.

The MRI-3 experiment consists of 75,497,472 examples and it takes about 4 to 5 days for FCM to complete each experiment. We clustered the whole data 10 times using FCM. So, we compare it with the first 10 experiments of SPFCM and thus the average results are computed over 10 experiments only. On this data set we loaded only 1% of the data for SPFCM. The average quality difference of FCM with SPFCM1 is 0.0053%. FCM on average took 102.72 hours, above 4 days, to partition the whole data set, while SPFCM took only 1.47 hours. Thus, the speed up obtained is 69.87 times. The quality difference and speed up obtained on this 75 million example data set were also excellent.

In summary, fuzzy c means is widely used in image segmentation and for other data partitioning purposes. Looking at the excellent quality and speed-up achieved by SPFCM, besides using it for clustering large generic data sets, it can also be used to accurately segment, compared to FCM, large volumes of image data quickly.

For example MRI-1, which is an entire volume of MRI data of the human brain, is segmented into 9 clusters accurately in less than a minute on average with our not so fast processor. So, our SPFCM algorithm could help segment the huge amounts of data involved in 3D modeling. For example, to our knowledge segmenting the whole volume of MRI at once using multiple features (generally it is done slice by slice) is done rarely, if ever, but with our SPFCM this can be explored/studied.

Table 4.1 Difference in quality of FCM with SPFCM. All values expressed in percentages.

Data Sets	Quality Difference
Iris (SPFCM5)	11.960000 %
Iris (SPFCM10)	0.050000 %
Pen Digits (SPFCM1)	4.800000 %
Pen Digits (SPFCM10)	0.240000 %
Isolet6 (SPFCM1)	3.180000 %
Isolet6 (SPFCM10)	0.280000 %
MRI-1 (SPFCM1)	0.120000 %
MRI-1 (SPFCM10)	0.020000 %
MRI-4 (SPFCM1)	0.005187 %
MRI-4 (SPFCM10)	0.001512 %
MRI-5 (SPFCM1)	0.005062 %
MRI-5 (SPFCM10)	0.000759 %
KDD (SPFCM1)	0.134279 %
KDD (SPFCM10)	-0.029527 %
Plankton (SPFCM1)	0.0069842 %
Plankton (SPFCM10)	0.0022635 %

4.5 Complexity Analysis

In [21], time, memory, and disk input/output complexity of single pass hard c-means and the hard-c-means algorithm were discussed. Similarly, we discuss the time, memory, and disk input/output complexity of our single pass fuzzy c means and FCM algorithm. We use the following notation in our discussion.

i number of iterations of FCM over the full data set

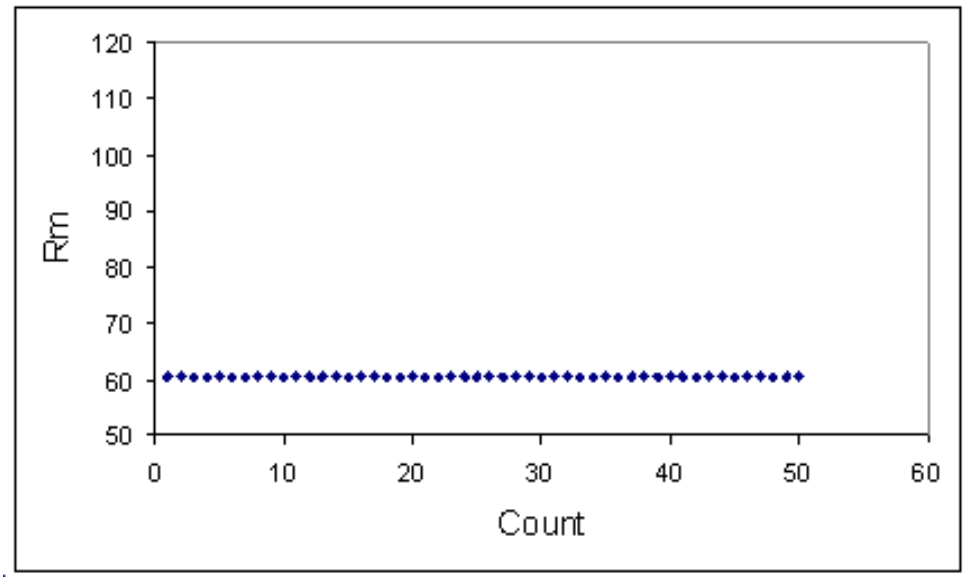


Figure 4.1 Extrema distribution of FCM on the Iris data set.

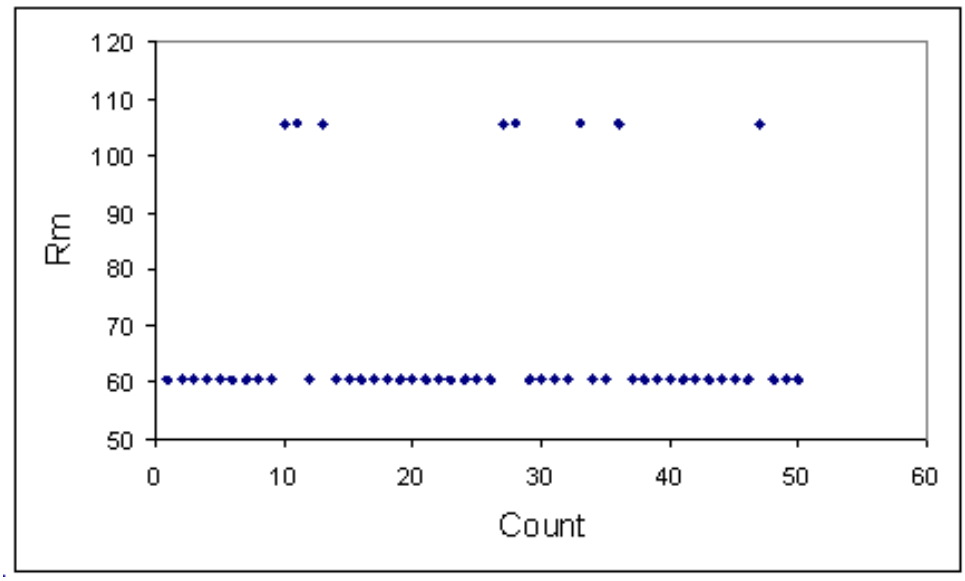


Figure 4.2 Extrema distribution of SPFCM5 on the Iris data set.

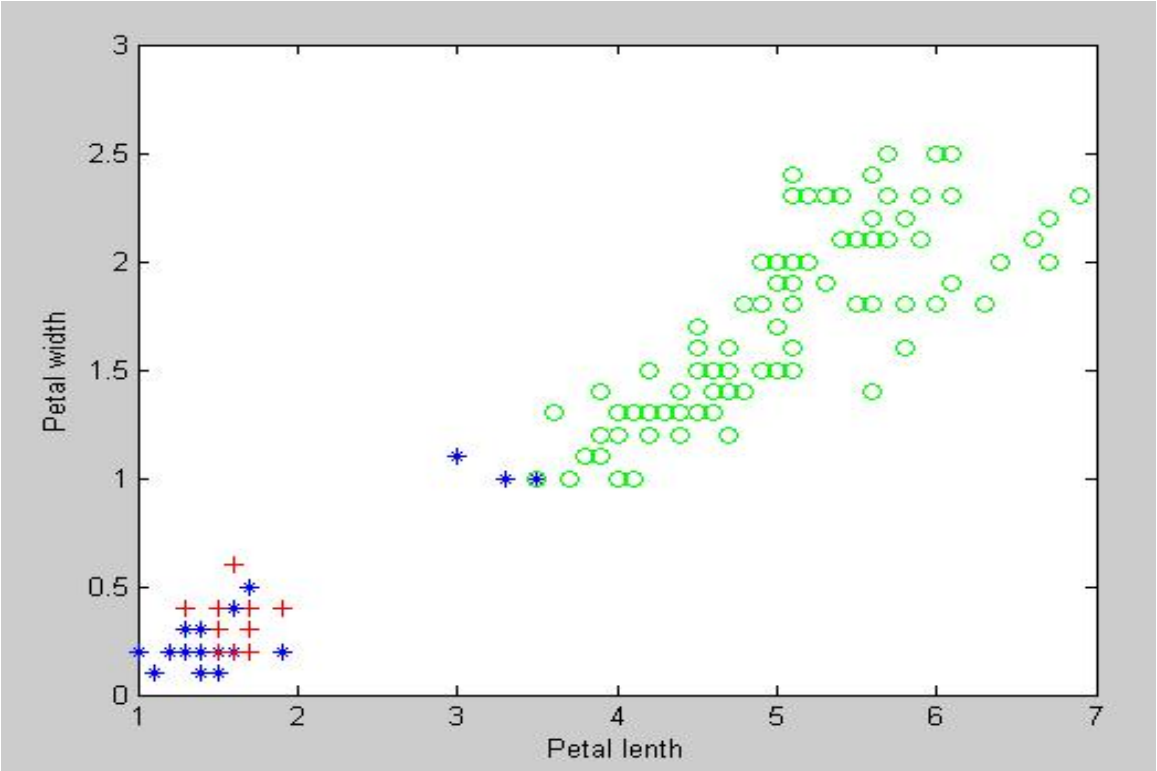


Figure 4.3 Plotting of the poor extrema obtained on Iris with SPFCM5 using the dimensions petal length and petal width. Symbols "o", "*", and "+" represent the three clusters.

Table 4.2 Speed up (SU) of SPFCM compared to GC. Time computed in seconds.

	Global Time(S)	SPFCM10 Time(S)	SPFCM1 Time(S)	SU (SPFCM10)	SU (SPFCM1)
Pen Digits	10.38	1.60	0.81	6.48	12.81
Isolet6	7.06	2.12	1.47	3.33	4.80
MRI-1	2408.60	181.84	54.93	13.24	43.48
MRI-4	227.67	50.33	26.95	4.52	8.44
MRI-5	100.18	21.53	11.09	4.65	9.02
KDD	541.89	69.31	27.02	7.81	20.05
Plankton	1522.98	112.15	48.18	13.57	31.60

$i^{'}$ average number of iterations of SPFCM in each PDC

n number of examples/data points

f number of dimensions

p fraction of data loaded in each PDA

d number of PDAs required by SPFCM

c number of clusters

The time complexity of FCM is $\mathrm{O}\left(nfc^2i\right)$. For the SPFCM, time complexity in each PDC will be $\mathrm{O}\left(pnfc^2i^{'}\right)$. As there will be d PDCs, the time complexity to cluster the whole data set is $\mathrm{O}\left(pnfc^2i^{'}d\right)$. Because $d = \frac{1}{p}$, it reduces to $\mathrm{O}\left(nfc^2i^{'}\right)$. As stated in [21], we also found $i^{'}$ to be typically smaller compared to i because after the first PDC we get an approximate model (centroids), and as we pass this knowledge to subsequent PDCs by initializing with previous cluster centroids, the clustering of freshly loaded singleton points along with past weighted points converges quickly. It has been shown before that initializing with "good" centroids helps FCM converge quickly [29].

The memory complexity of FCM is $\mathrm{O}\left(nf + nc\right)$, where nf is the size of data set and nc the size of U matrix. For SPFCM, the memory complexity is $\mathrm{O}\left(pnf + pnc\right)$. As $p < 1$, the memory complexity of SPFCM is less than FCM.

For data sets which cannot be loaded into memory, FCM will have disk accesses every iteration. Thus the disk input/output complexity will be $\mathrm{O}\left(nfi\right)$. It is likely that for those data sets the U matrix cannot be kept in memory too. Thus, it will increase the disk input/output complexity further. In the case of SPFCM, input/output complexity in each PDA is $\mathrm{O}\left(pnf\right)$. Thus overall complexity for clustering the whole data set is $\mathrm{O}\left(pnfd\right)$, which is $\mathrm{O}\left(nf\right)$.

4.6 Clustering Data Coming in an Unpredictable Order

In all experiments above we assumed data in each PDA has a sufficient number of representative examples from each class. We achieved this by randomly reordering the data sets. However, data might not always come in a "good" mixture of all classes. We might have to process data as it comes, which is especially true for streaming/online algorithms. We performed experiments on some of the data sets, in which SFCM processed data in the order it comes, that is, data was not randomly reordered to create a "good" mixture of all classes. We choose the MRI-4 (1.5T data set), MRI-5(3T data set), and the KDD data set. We choose the MRI data sets because they are known to evolve as we go up or down the brain in the axial plane. The KDD data set was also used because it is available publicly and is a well known data set to test incremental algorithms. Table 4.3 shows the difference in quality obtained on the all data sets with 10% data loaded. All experimental results were an average of 32 times random experiments. On the KDD data set, we got excellent quality, with average quality of the single pass fuzzy c means algorithm slightly better than average quality of FCM. However, on the MRI-4 data set we got poor quality, about 20% difference from FCM. On the MRI-5 data set, we got a comparable difference in quality, about 3%. It seems that the quality obtained was unpredictable and depends from data set to data sets. Thus, single pass fuzzy c means can be used to obtain excellent quality when data comes in a "good" mixture, but may give unpredictable results for data coming in an unpredictable order. Thus, this kind of algorithm may not be suited for clustering streaming data where we have to process data as it comes. In the next Chapter, we will process a streaming variant of the FCM algorithm. We will also propose a generalized single pass fuzzy c means algorithm, known as online

FCM, that will provide clustering quality as good as clustering the full data set and will be able to process data as it comes.

Table 4.3 Single Pass Fuzzy C Means experiment without scrambling data.

	Diff in Quality (%)
KDD(SPFCM10)	-0.001394
MRI-4(SPFCM10)	20.529071
MRI-5(SPFCM10)	2.961599

CHAPTER 5

ALGORITHMS FOR CLUSTERING DATA STREAMS

In this chapter we propose a streaming variant of the fuzzy c means algorithm (SFCM). As stated in the previous chapter, the methods applied to the crisp algorithms may not be easily generalized to fuzzy case [44]. This is because soft clustering algorithms try to optimize different objective functions compared to the objective functions minimized by crisp clustering algorithms. Our contribution in this work is two fold. First, we propose a streaming variant of the fuzzy c means algorithm, and second we empirically study the tradeoff involved between responding to an evolving distribution and summarization of data seen so far by varying past history usage in clustering streaming data. The performance of our algorithm with noise is also studied. In streaming algorithms will process data as it comes.

5.1 Fuzzy Streaming Algorithm

Due to the constraints of limited memory and computation time, a streaming algorithm will be able to load only a relatively small amount of the data at a time depending upon the speed of the stream and hardware capability. As in [43], we assume data is both arriving and processed in chunks, that is, n_1 data points arrive at time t_1, n_2 at t_2, and so on. After clustering n_1 data at time t_1, memory is freed by condensing the clustering result in memory into c weighted points. The c weighted points are represented by the c cluster centroids obtained after clustering. The summarization of clustering results involves the U matrix, which indicates the

fuzziness of examples associated with each cluster. This is one key difference in our summarization with the streaming algorithm of crisp cases.

For example if there are n_1 examples in memory at time instant t_1 then after clustering, memory is freed by summarizing the clustering result by c weighted centroids, whose weights are calculated using the membership matrix as follows:

$$w_i = \sum_{j=1}^{n_1} u_{ij}, 1 \leq i \leq c \tag{5.1}$$

At time t_2 the c weighted points, summarized result of t_1, are clustered with n_2 freshly arrived examples. The use of weighted points from previous clustering represents the use of history. After clustering, memory is freed by representing the clustering solution at time t_2 by the cluster centroids obtained. Similarly to (5.1), the weight of the cluster centroids are obtained only by summing the membership values of points which arrived at t_2. This is one key difference from the single pass approach [21,43,63], where weights are accumulated over time, which might make streaming algorithms insensitive to evolving distribution. Similarly at t_3, weighted centroids of t_2 are used to cluster with freshly arrived singleton points. This continues until the end of the stream. In the above method we are using only weighted centroids from the previous time instant. One may use weighted centroids from the previous two (or more) time instants also. This is equivalent to using more history. Obviously, using more history means clustering newly arrived data with more than c weighted points from previous time instants. For example if we use the history of the previous two time instants at any time instant t_n, there will be c weighted points from time instant $t_n - 1$ and c weighted points from $t_n - 2$ along with newly arrived singleton points at time t_n. So, if the streaming algorithm is using the history of the previous p time instants, it needs to keep $p*c$ weighted points in memory. The weighted centroids, with a limited

history, certainly will not occupy much memory and can be efficiently stored. So, the maximum usage of history will depend on the available buffer size. We will later show that one need not keep a large amount of history in memory for obtaining good partition quality. The weighted centroids of all time instants could be saved on disk too. As these centroids are summarized results of time instants, they could be used offline to analyze patterns of clusters obtained over different time horizons [57, 60]. Clustering of each chunk is initialized with centroids obtained from the previous clustering. This knowledge propagation helps in faster convergence of clustering when the distribution is slowly evolving. It should be noted that at the beginning of the stream all weighted centroids of past p time instants won't be available until time instant t_{p+1}. So, till then it would use the maximum amount of history available, which is none at t_1.

It should be noted that at any time t_n if we use the weighted centroids of the previous time instant $(t_n - 1)$, it does not mean that we are using the information of examples at $t_n - 1$ only. This is because cluster centroids at $t_n - 1$ were not only obtained using the singleton points at that time instant but also weighted centroids from $t_n - 2$. Similarly, cluster centroids at $t_n - 2$ were obtained using weighted centroids at $t_n - 3$ and so on. So, the information is getting propagated from one time instant to another time instant. Obviously, the more history we use the more information from past clustering will get propagated.

The weighted FCM algorithm is similar to that discussed for the single pass FCM algorithm (SPFCM) in the previous chapter. However, for SPFCM there were c weighted points only, but for the streaming algorithm there may be more than c weighted points, which depends on the amount of history used. Still, we discuss below the weighted FCM again in the context of streaming algorithm to make this chapter self sufficient.

5.1.1 Weighted FCM

The objective function (J_m) minimized by FCM is defined as follows:

$$J_m\left(U,V\right)=\sum_{i=1}^{c}\sum_{k=1}^{n}U_{ik}^{m}D_{ik}(x_k,v_i) \tag{5.2}$$

$U$and V can be calculated as:

$$U_{ik}=\frac{D_{ik}\left(x_k,v_i\right)^{\frac{1}{1-m}}}{\sum_{j=1}^{c}D_{jk}\left(x_k,v_j\right)^{\frac{1}{1-m}}} \tag{5.3}$$

$$v_i=\frac{\sum_{j=1}^{n}\left(u_{ij}\right)^m x_j}{\sum_{j=1}^{n}\left(u_{ij}\right)^m} \tag{5.4}$$

Where,

U_{ik} : is the membership value of the k^{th} example, x_k, in the i^{th} cluster.

v_i : is the i^{th} cluster centroid.

n : is the number of examples

c: is the number of clusters

$D_{ik}\left(x_k,v_i\right)=\left\|x_k-v_i\right\|^2$: is the norm. We have used the Euclidean distance.

The reformulated optimization criteria R_m (equation 4.4) , which is mathematically equivalent to J_m (equation 5.2) was given. So, if we need to compute the objective function value of any partition, we will compute the R_m value.

The clustering result at any time instant is summarized and memory is freed by representing the clustering solution as weighted centroids. In subsequent time instants these weighted points along with newly arrived singleton points can be clustered. We need to modify the fuzzy objective function (similar to [22]) to take into account the effect of these weighted points.

Let us consider that n_d singleton points (weight one) are in the buffer at time t_n and there are h weighted points obtained from clustering of previous time instant/instants. The value of h will depend on how much previous history we want to use and their union with singleton points constitutes the set $X^{'}$. The cluster centroids for the weighted FCM (WFCM) are calculated as:

$$V_i = \frac{\sum_{j=1}^{n_d+h} w_j (u_{ij})^m x_j^{'}}{\sum_{j=1}^{n_d+h} w_j (u_{ij})^m}, \quad 1 \leq i \leq c, \quad x_j^{'} \in X^{'}.$$

The weight of n_d examples loaded will have weight 1 and the weight of h weighted points are calculated from condensation/summarization of clustering at previous time instant(s).

The membership matrix is calculated as follows.

$$u_{ij} = \left[\sum_{l=1}^{c} \left(\frac{\left\| x_j^{'} - v_i \right\|}{\left\| x_j^{'} - v_l \right\|} \right)^{\frac{2}{m-1}} \right]^{-1}, 1 \leq i \leq c \text{ and } 1 \leq j \leq n_d + h$$

It should be noted that modification of objective function does not change the convergence property of FCM because a weighted point can be thought of as many singleton points of identical value.

5.2 Data Sets Used

Artificial Data: As true centroids for artificial data are available, artificial data makes it easier to study how the streaming algorithm responds to an evolving distribution. An evolving distribution is simulated both by varying the rate of the change of centroids from slow to rapid. The artificial data has 2 features and 4 clusters. A total of 100,000 examples were generated in 20 different phases, 5000 examples from each phase. So, there will be 20 sets of generating functions/true centroids, one for each phase. In each phase centroids of Gaussians from which the examples were

generated were determined based on the centroids of the previous phase. They may either be changing slowly (centroids moved by 2 to 5% of the value of the standard deviation value in each dimension) or remain unchanged or change sharply (centroids moved by 50 to 60% of the value of the standard deviation value in each dimension) compared to the centroids of previous phase. The sharp change can happen only at any 2 phases, randomly chosen between phase 2 to 19. Centroids for the other phases were obtained by either moving the centroids slowly or not changing them at all relative to the centroids in the previous phase with a probability of 50%. When we move centroids, we either add or subtract the change in each dimension of the centroids. This choice was made randomly in each dimension. The means of the Gaussians for each of the clusters in the initial phase were (300, 300), (800,300), (300,800), and (800,800) (all slightly overlapped). Figure 5.1 shows the 5000 examples generated in the first phase. The standard deviations of the mixtures were 100 in both dimensions. We also added Gaussian noise with a signal to noise ratio of 5:1. This data set was clustered into 4 clusters. The code for generating data from a gaussian distribution was obtained from http://ftp.arl.mil/random/.

To have a visual understanding of how the centroids moved in each phase, we computed the sum of the Euclidean distance of the centroids at each phase from its corresponding centroid at the previous phase. This will show the relative movement of the centroids. As data from these phases will arrive in chunks later in our experiments, we attempt to show how the distribution evolved with time. The x-axis in Figure 5.2 shows the phases and the y-axis the relative movement of the centroids. When the curve in the figure coincides with the x-axis, it indicates centroids did not move in that phase relative to the previous phase. A sharp rise indicates a sudden change in centroids, a spike, which we see occurred at phase 3 and phase 9. It should be noted that change of centroids may not always mean a change in distribution. For example,

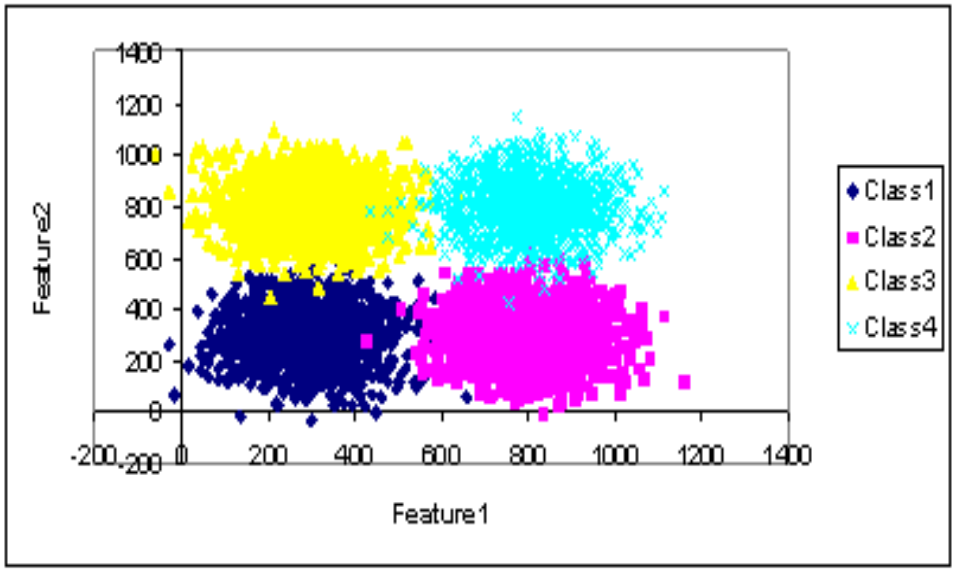


Figure 5.1 Plotting of data generated in the first phase.

if the 4 cluster centroids are moved in such a way that the relative distance between them remains the same, it is just a change of scale. As we are changing centroids randomly in each dimension, such an occurrence is unlikely.

We also used six real data sets, MRI-4, MRI-5, MRI-6, MRI-7, MRI-8, and MRI-9. We chose MRI because the distribution of tissues in human brain naturally evolves as we go up or down along the axial plane. There will be also different amounts of tissue at different locations. We fetched data for this experiment along the axial plane, from the bottom of the brain (neck) to the top of the skull. So, we believe it will be a good data set to study the streaming algorithm in a real life scenario. Two data sets, MRI-4 and MRI-5, were previously discussed in Chapter 4. The value of m used for all these data sets was 2.

- *MRI-6 Data.* This data set was created by concatenating 96 slices of MR images, T1 weighted, of size 512X512 from a single human brain. The magnetic field strength was 1.5 Tesla. After air and skull were removed using the brain

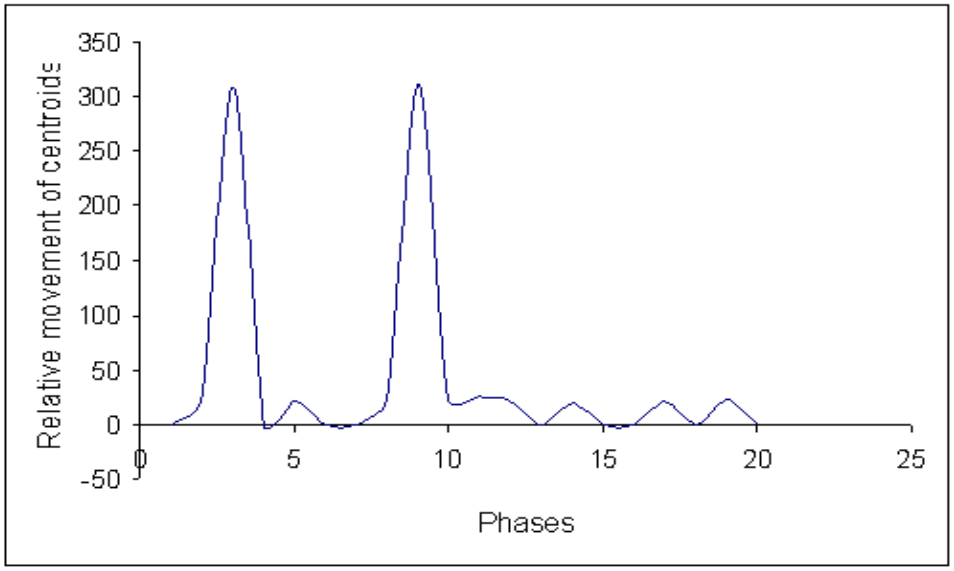


Figure 5.2 Plotting of the movement of centroids in each phase.

extraction tool, BET2 [80], there were 4,948,180 examples. We clustered this data set into 3 clusters.

- *MRI-7 Data.* This data set was created by concatenating 96 slices of MR images, T1 weighted, of size 512X512 from a single human brain. The magnetic field strength was 1.5 Tesla. After air and skull were removed using the brain extraction tool, BET2, there were 4,031,593 examples. We clustered this data set into 3 clusters.

- *MRI-8 Data.* This data set was created by concatenating 144 slices of MR images, T1 weighted, of size 256X256 from a single human brain. The magnetic field strength was 3 Tesla. After air and skull were removed using the brain extraction tool, BET2, there were 1,236,969 examples. Intensity homogeneity on this data set was corrected using an implementation of the bias correction algorithm in [81]. We clustered this data set into 3 clusters.

- *MRI-9 Data.* This data set was created by concatenating 144 slices of MR images, T1 weighted, of size 256X256 from a single human brain. The magnetic field strength was 3 Tesla. After air and skull were removed using the brain extraction tool, BET2, there were 1,504,594 examples. Intensity homogeneity on this data set was also corrected. We clustered this data set into 3 clusters.

All experiments were done on an UltraSPARC-III with 8 processors each of 750 MHz. There was 32GB of main memory. None of the programs were multithreaded.

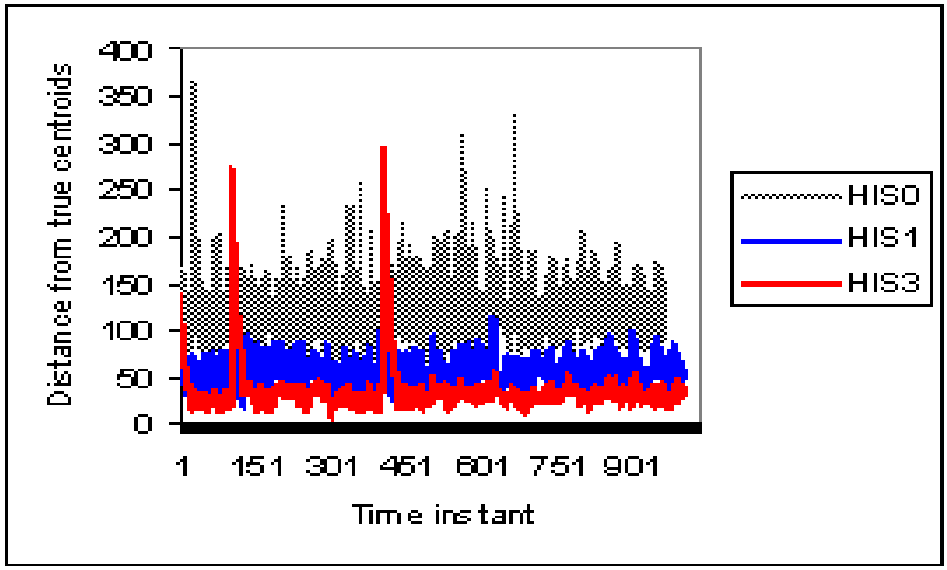
5.3 Results and Discussion

The chunk size, the amount of data that arrived at each time instant, was kept to 100 examples for the artificial data experiment. By time instant we mean the moment at which a fresh chunk arrives. As the total size of the artificial stream was 100,000 examples, there will be 1000 time instants. True centroids are available for the artificial data, so quality is evaluated by computing the sum of the Euclidean distance between centroids generated by the streaming algorithm at any time instant and the true centroids from which the data was generated at that time. For computing a distance between centroids one needs to solve the correspondence problem. We matched the centroids using the minimally weighted perfect bipartite matching algorithm [76], which optimally matches the centroids with minimum overall cost. The code for bipartite matching (Hungarian method) was obtained from http://reptar.uta.edu/. All experimental values were averaged over 32 experiments, each started with centroids randomly initialized at time instant 1.

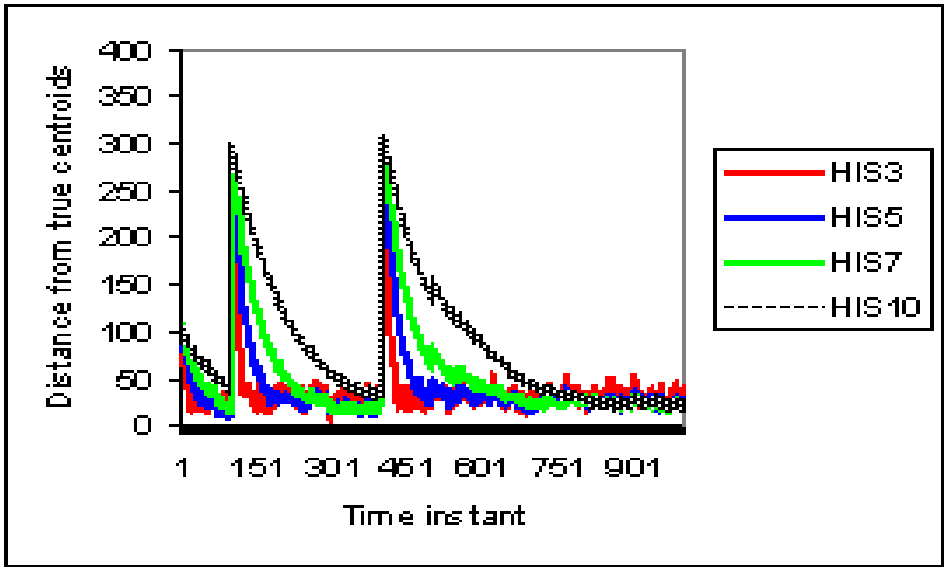
We conducted experiments by varying the amount of history. For example HIS1 means at any time instant only weighted centroids from the immediate previous time instant were used. Similarly, HIS2 means only weighted points from the immediate

previous 2 time instants were used. For HISn weighted points from previous n time instants were used. HIS0 indicates only FCM was applied at any time instant without using any history. Figure 5.3 shows the results using different amounts of history. For clarity purposes, we have split the results into Figures 5.3 (a) and (b). Figure 5.3 (a) contains results of HIS0, HIS1, and HIS3. Figure 5.3 (b) contains results of HIS3, HIS5, HIS7, and HIS10. HIS3 was repeated because the plots were split at this point. So, it will serve as a reference. In the figures the x-axis is the time instant and the y axis is the distance from the true centroids. The streaming quality using HIS0 is the worst because it is simply FCM applied at any time instant. As we increase the usage of history we can see quality improves, that is, centroids generated by the streaming algorithm become closer to the true centroids. We also observe that the streaming algorithm responded well, HIS1 to HIS5, to the sharp changes which occurred at time instants 100 (phase 3) and 400 (phase 9). After the sharp changes, HIS1 and HIS3 take near about 20 time instants for error to become low. HIS5 takes about 80 time instants, while HIS7 takes near about 150 time instants. HIS10 takes longer than 350 time instants. It was observed that more the history we used the more slowly error settles after the sharp changes. This was expected because past history dominates for some time. On this data set and with this chunk size this happens mainly after HIS5, that is, for HIS7 and HIS10. So, it seems that usage of too much history may not be good for a rapidly changing distribution. The tradeoff will vary by data set and may vary upon application. We empirically study the tradeoff here, but in the future theoretical analysis would be valuable.

For the real data set experiments, MRI volumes, we do not know the true centroids. So, the previous quality metric could not be used. Instead we evaluate quality using the fuzzy objective function value, R_m. We clustered the whole MRI data (entire volume) using FCM and computed the objective function value, R_m. The objective



(a)



(b)

Figure 5.3 a) Comparison of HIS0, HIS1, and HIS3. b) Comparison HIS3, HIS5, HIS7, and HIS10.

function value has also been used to evaluate streaming quality in [43, 57]. Because their algorithms were crisp, they computed the sum of squared distance (SSQ). As this R_m is computed clustering all the data at once (Global R_m value), this will provide a bound against which we can compare the quality of the streaming algorithm at different time instants. At any time instant using the centroids produced by the streaming algorithm, the R_m value is computed over all the examples seen so far. A good streaming algorithm should be able to summarize information with time, so at the end (last time instant) it should produce an R_m value comparable to the R_m value obtained by clustering all the data at once. As the number of examples seen so far will vary by time instant, the R_m value was normalized by dividing by the number of examples from which it was computed. The global R_m value was also normalized by dividing by the size of the entire data set. We set the chunk size to approximately 5% of the total data set size, so there will be 20 time instants. We performed experiments by varying the amounts of history used from one to five chunks (HIS1 to HIS5).

Figures 5.4 to 5.9 show the results using different amounts of history on the six data sets, MRI-4, MRI-5, MRI-6, MRI-7, MRI-8, and MRI-9 respectively. In the figures the x-axis is the time instant and the y axis is the normalized R_m value. All experimental values are an average of 32 experiments each started with random initialization of centroids at time instant 1. As stated earlier, in Figure 5.4 to 5.9 GLOBAL is the normalized R_m value obtained by clustering the full data set. We plotted this value for all time instants to compare with the R_m values of the streaming algorithm. As we can see at the end (time instant 20) all streaming experiments, on all data sets, using a history of up to 2 chunks (HIS2) performed better than HIS0. On MRI-4, MRI-5, and MRI-6 the results with HIS1 and HIS2 almost touched the GLOBAL line at the end indicating that the final centroids obtained at the end of the stream, time instant 20, were almost same as those obtained by FCM clustering

the full volume at once. It should be noted that usage of a history greater than 2 on these volumes, MRI-4, MRI-5, and MRI-6, shifts the R_m value away from GLOBAL, indicating an overuse of history. On MRI-7, MRI-8, and MRI-9 overuse appears to happen with usage of a history greater than 1.

In Figures 5.4 to 5.9, the performance of HIS0 on all data sets detoriated suddenly at the end of the stream. It was quite expected because the quality of HIS0 depends on chance, and it must have gotten a bad sample at the end of the stream, which was non representative of the total number of examples seen so far. However, a streaming algorithm with appropriate usage of history, generally HIS1 and HIS2, steadily minimizes the normalized R_m value with time. This is obvious from the fact that at time instant 20 (at the end) even if the quality of HIS0 sharply detoriated, the quality of streaming algorithm did not change rapidly. At the beginning (time instant 1), some streaming experiments, on data sets MRI-4, MRI7, MRI-8, and MRI-9, had a lower normalized R_m value compared to global normalized R_m. This is because in the streaming algorithms data do not arrive in a random way, and the normalized R_m value for the streaming experiments at time instant 1 was computed from data seen till time instant 1 only, that is approximately the first 5% of the entire volume, which was non representative of an overall distribution. As time goes on more data is seen, so estimation of R_m becomes more representative of the overall distribution. So, it seems with usage of an appropriate amount of history it may be possible to produce clustering quality, at the end of the stream, as good as clustering the whole data set at once by FCM.

We saw in the artificial data experiments that the more history used the more slowly error settles after sharp/significant changes, as outdated history dominates for a while. As discussed earlier in the context of the artificial data experiments, for a quickly changing distribution usage of less history is desirable. It should be noted

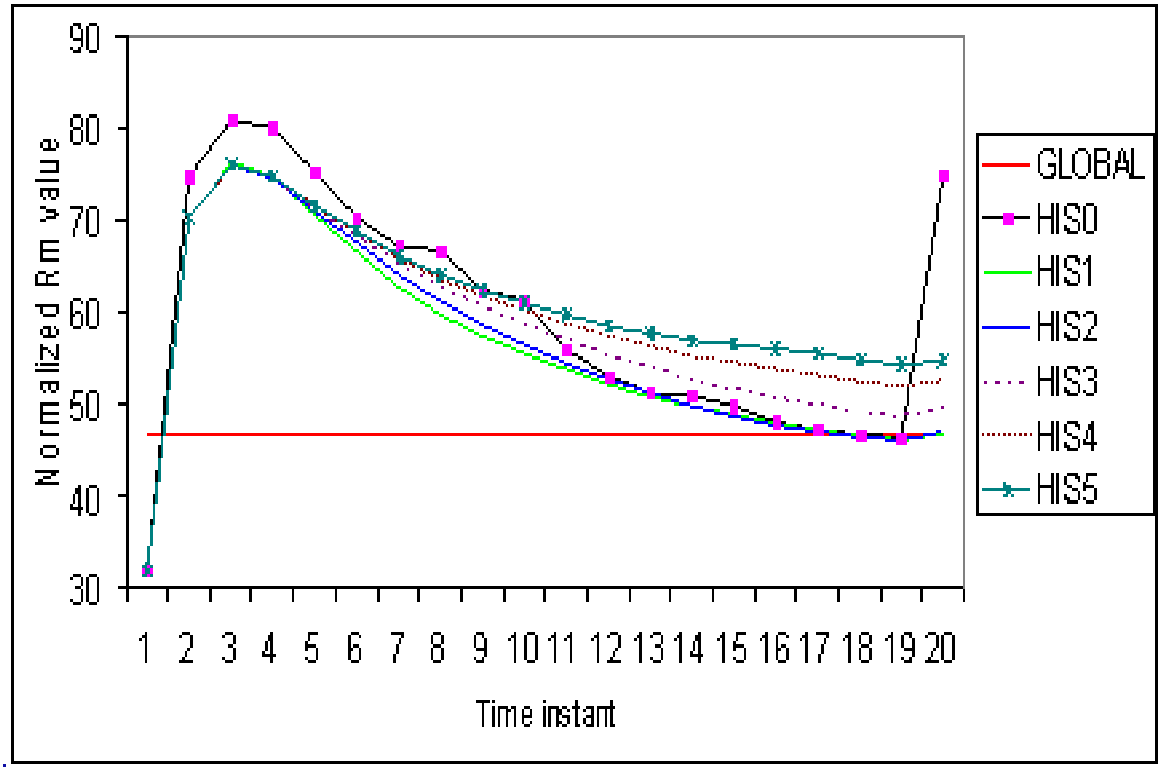


Figure 5.4 Streaming quality on the MRI-4 data set with chunk size set to 5%.

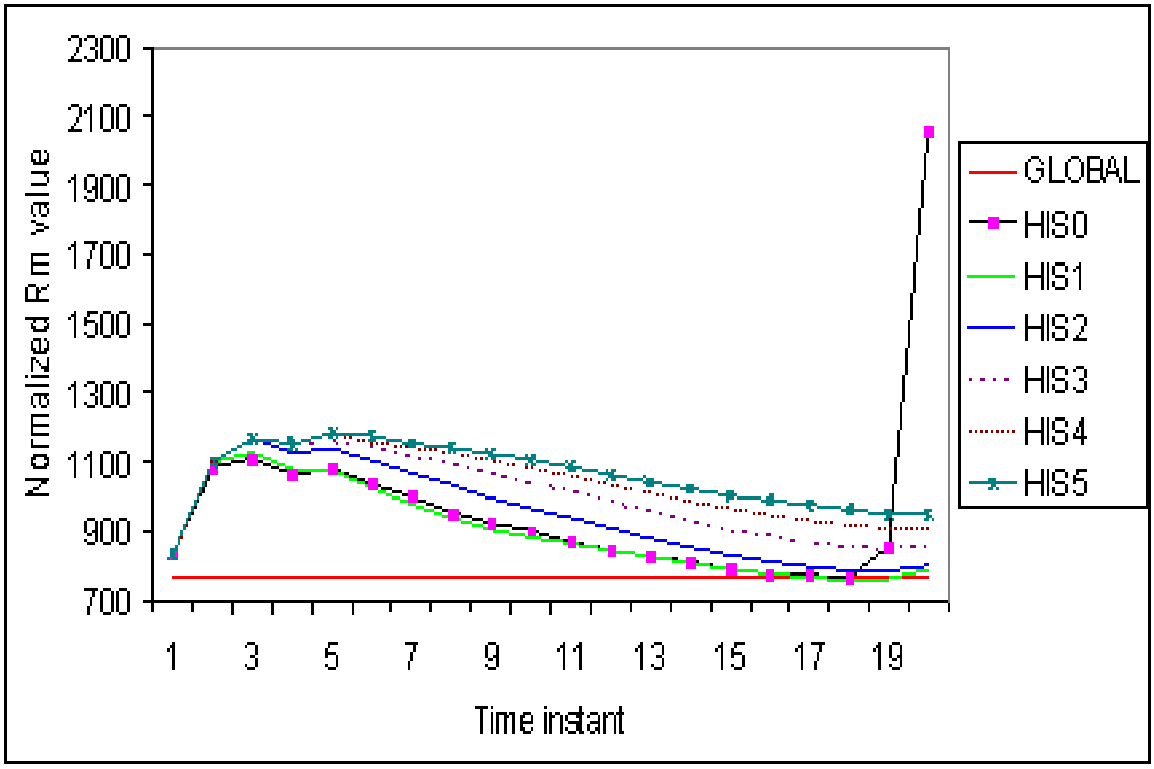


Figure 5.5 Streaming quality on the MRI-5 data set with chunk size set to 5%.

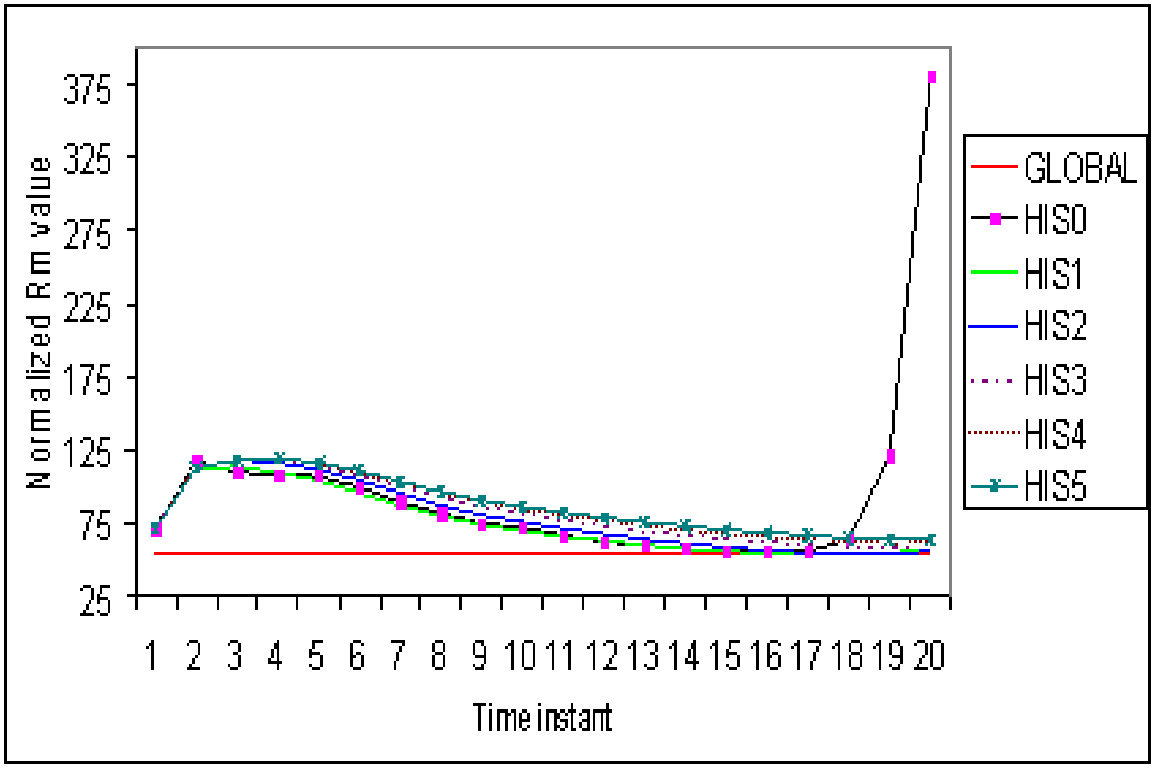


Figure 5.6 Streaming quality on the MRI-6 data set with chunk size set to 5%.

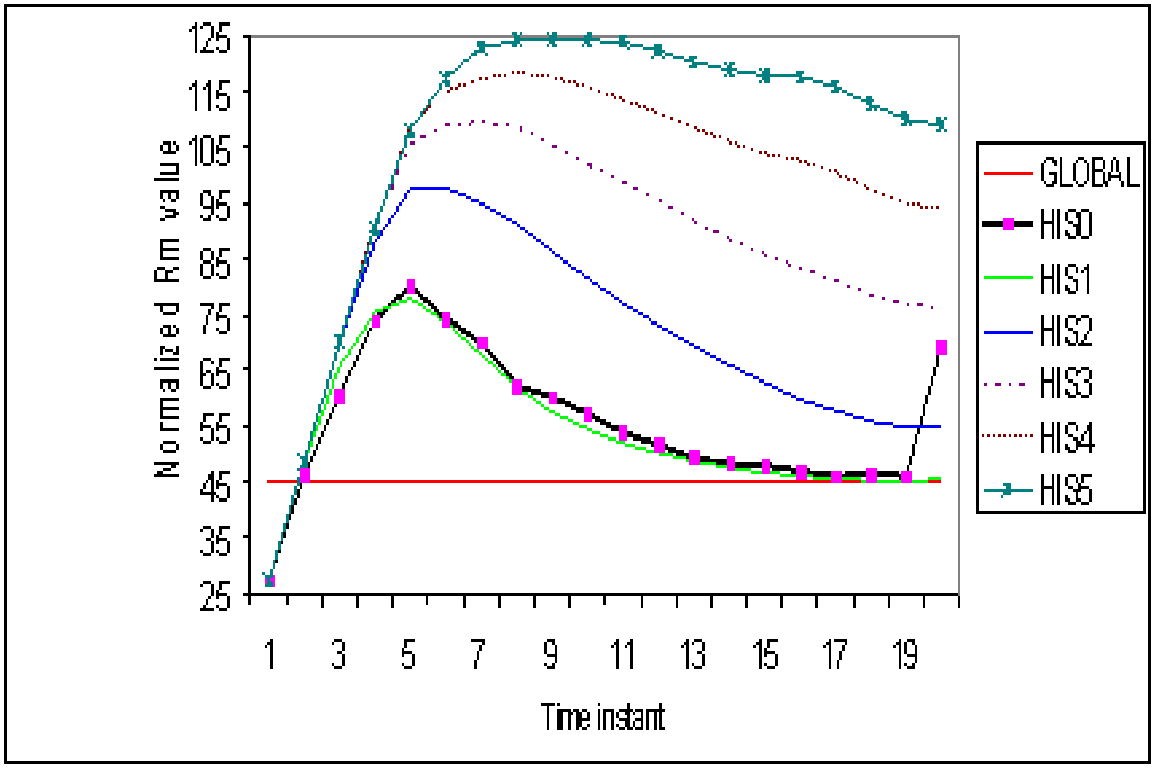


Figure 5.7 Streaming quality on the MRI-7 data set with chunk size set to 5%.

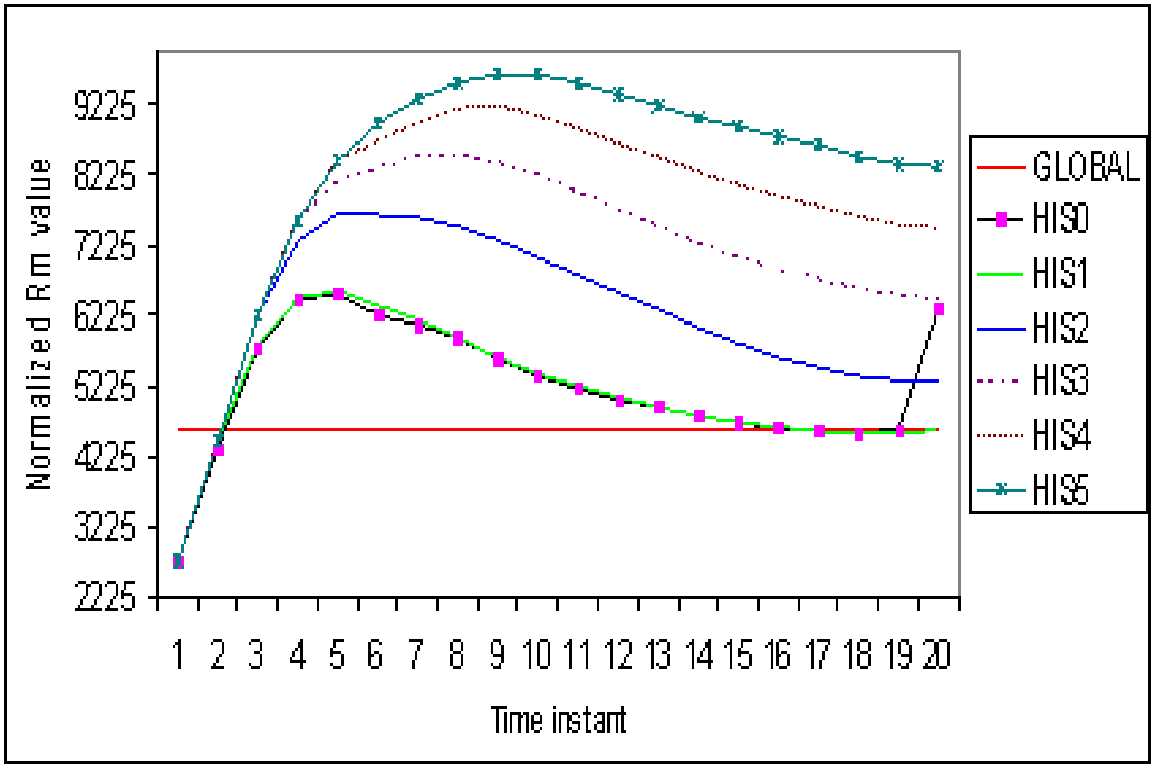


Figure 5.8 Streaming quality on the MRI-8 data set with chunk size set to 5%.

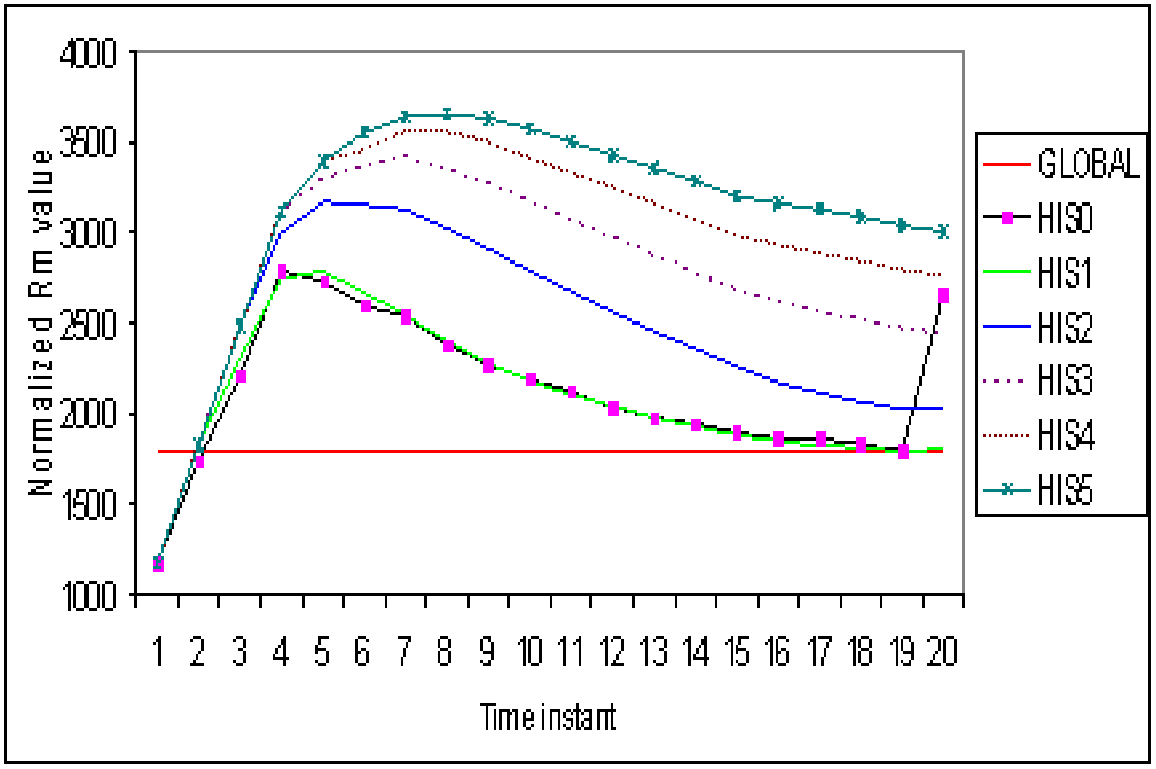


Figure 5.9 Streaming quality on the MRI-9 data set with chunk size set to 5%.

that for the same data set, the chunk size will determine how quickly the streaming algorithms will face such sharp changes. For example, consider a data stream created from a finite size data set. Let the stream on average undergo significant/sharp changes in distribution after every 20% of the data it generates, that is, the generating function/true centroids changes every 20% of the data. If we fetch 5% of the full data in each chunk, the algorithm will see significant changes after every 4 chunks. Now for the same data set, if we fetch 1% data in each chunk, the algorithm would face significant changes after every 20 chunks. In the later case sharp changes will occur less quickly compared to the former case. Thus, we hypothesize that in the later case (when 1% is fetched) more chunks of history can be used before an overuse of history occurs. This hypothesis was tested on the MRI-4 and MRI-9 data sets, from 1.5T and 3T respectively. We chose these 2 data sets because the other 4 have data sets had similiar curves (Figures 5.4 to 5.9) for history usage. The chunk size was set to 1% of the total data (previously using 5%), and the results were an average of 20 random experiments, each streaming experiment using different amounts of history started with the same initialization. Figures 5.10 and 5.11 show the results on data sets MRI-9 and MRI-4 respectively. With the chunk size set to 1%, the overuse of history occurred after HIS3 on data set MRI-9, whereas previously with chunk size set to 5% it occurred after HIS1 (Figure 5.9). Similarly, on data set MRI-4 the overuse of history occurred after HIS4 (Figure 5.11) compared to HIS2 when chunk size was set to 5% (Figure 5.4). These experiments seems to validate our hypothesis that for the same data set using a smaller chunk size more history can be integrated before overuse occurs. It should also be noted that at the end of the stream, time instant 100, HIS1 tended to change more than experiments using history greater than one. As discussed in the context of the artificial data experiments, it is because the distribution changed rapidly at the end, which is evident from the sudden rise in the

normalized R_m values of HIS0 in Figures 5.10 and 5.11, and with only 1% history it got adapted to that change quickly.

It seems that for the experiment with chunk size set to 5% (Figures 5.4 to 5.9), the data sets MRI-4, MRI-5, and MRI-6 evolved less quickly, fetching data from the bottom of the brain to the top, compared to data sets MRI-7, MRI-8, MRI-9, as overuse of history occurred quicker with the former (after HIS2) compared to the later (after HIS1). A mathematical model determining the amounts of information summarized with different chunk sizes and different amounts of history with time would be valuable in the future for accurately designing streaming algorithms for different applications.

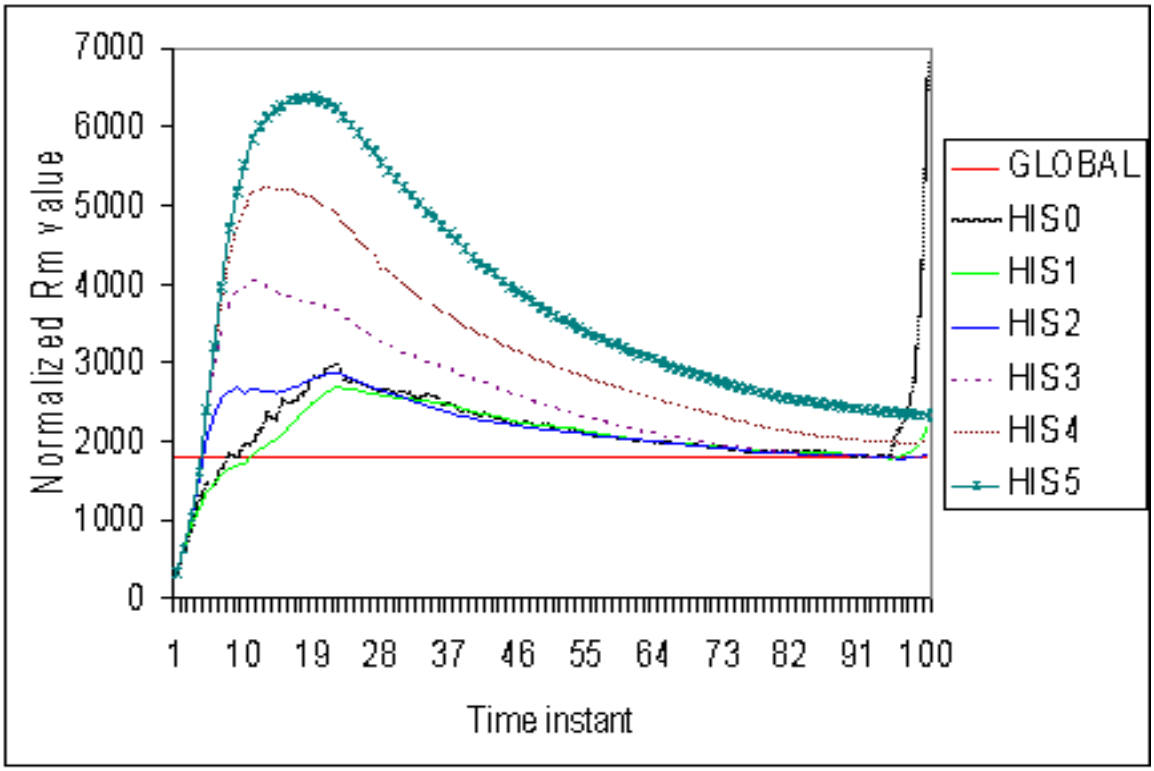


Figure 5.10 Streaming quality on the MRI-9 data set with chunk size set to 1%.

The purpose of a streaming algorithm is two fold; responding to an evolving distribution and summarizing the data seen so far. So, if at some point in time

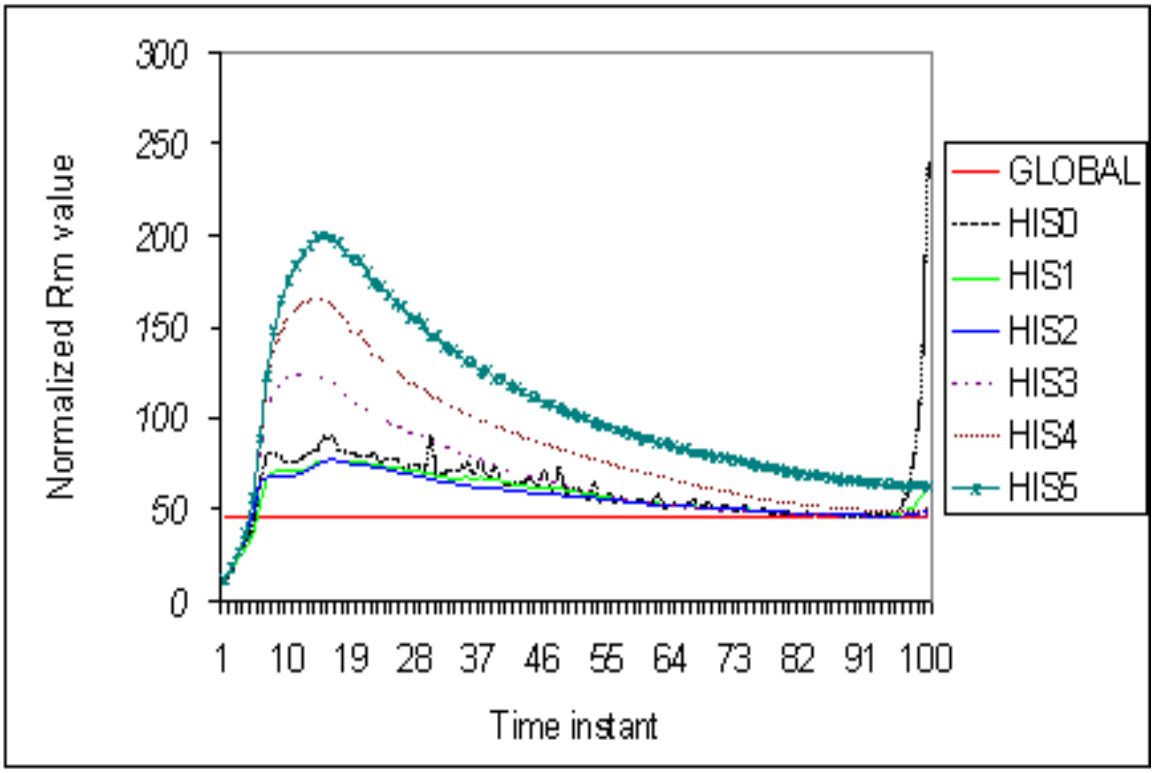


Figure 5.11 Streaming quality on the MRI-4 data set with chunk size set to 1%.

the distribution changes rapidly, a streaming algorithm will respond to that change. This characteristic is desirable for any streaming algorithm, but it may not be always suitable for producing clustering quality (at the end) as good as clustering the entire stream in memory. This is because the clustering quality of a streaming algorithm will be influenced more by current chunks, whereas for producing clustering quality as good as clustering the whole stream at once, each portion of the data should have an equal contribution. A streaming data set might flow for months, and the flexibility in the usage of different amounts of history may be useful because based on a particular application a user might choose to use more or less history. But, it might be also difficult to determine exactly how much history to use to produce clustering quality as good as clustering the entire stream, in which all examples in the stream should contribute equally. For this purpose, in the next section we propose a special purpose streaming algorithm which can produce clustering quality as good as clustering the whole data stream at once. This kind of algorithm will be useful because large data sets on disk can be clustered by a streaming approach to produce clustering quality as good as clustering the whole data set. As discussed earlier, for this purpose all chunks of data will contribute equally in obtaining a final clustering solution, and it would be able to process the data as it comes. We will call the new algorithm online FCM (OFCM) and its design issues and performance will be discussed in the next section.

5.4 Online Fuzzy C Means

Consider a full data set has c classes (the maximum that can be present). Due to the constraints of limited memory and computation time, an online algorithm will be able to load only a small amount of data at a time depending upon the speed of data arrival and hardware capability. As in streaming algorithms, we assume data

is both arriving and processed in chunks, that is, n_1 data points arrive at time t_1, n_2 at t_2, and so on. Then in the worst case in a given chunk data might come from one class only and in the best case data might come from all n classes. So, if we set the number of clusters to be always c (highest resolution) what effect will it have on different orders?

Case A: If less than c classes arrive in a chunk, then we are overclustering. Overclustering does not cause any information loss. Information loss only occurs when we undercluster.

Case B: If exactly c classes come in a chunk, then we are partitioning the data correctly, that is, neither overclustering nor underclustering.

It seems that setting the number of clusters to be always equal to c, the maximum number of classes in the data set, will not cause any information loss. So, we set the number of clusters to be c in each chunk. After clustering data arriving at each time instant by FCM (a chunk), memory is freed by condensing the clustering solution in the memory into c weighted points. The c weighted points are represented by the c cluster centroids obtained after clustering. The summarization of clustering results involves the U matrix, which indicates the fuzziness of examples associated with each cluster, a key difference in our summarization with the algorithms for crisp cases.

For example if there are n_i examples in memory at time instant t_i then after clustering, memory is freed by summarizing the clustering result by c weighted centroids as done in equation 5.1.

The weighted centroids are saved. At the end, the weighted centroids of all chunks form an ensemble of weighted clustering solutions. The ensemble is then merged into c final clusters. The merging operation is done by clustering all the weighted centroids in the ensemble using their weights. The weighted FCM (WFCM), discussed before, has been used for this purpose. The only difference is that it has no singleton points

now. To speed up clustering, we initialize the clustering process for each chunk with the final centroids obtained from clustering the previous chunk. This knowledge propagation allows for faster convergence, provided the distribution does not change rapidly, which might often be the case.

The size of the ensemble of weighted centroids is not likely to be large because it consists of only weighted centroids. If in any case it becomes large, similar to [43] the weighted centroids from the ensemble can be incrementally loaded and reclustered into c weighted centroids, which can be retained in memory only. This will decrease the ensemble size, which can be finally merged into c partitions in memory. It should be noted that while clustering each chunk, we do not use any history in OFCM. This allows each chunk to equally influence the final clustering solution, which is a necessary condition for obtaining clustering quality as good as clustering the entire stream at once.

Our online FCM algorithm is somewhat similar to the crisp streaming algorithm proposed in [43]. The difference between it and ours lies in the fact that we condensed the clustering solutions in a fuzzy way. Our method of summarizing clustering solutions of each chunk involves the fuzzy membership matrix, which does not exist for the crisp cases. We believe, condensing clustering solutions using fuzzy centroids and membership matrix is important; otherwise, the summarized clustering solutions will simply reduce to a form of crisp clustering solutions, which is not the goal. The idea can be extended to create clustering solutions at intermediate time instants as done for clustering (in a crisp way) evolving data streams in [57]. As discussed earlier, in [57] each chunk was over clustered to create micro clusters and then they were merged into macro clusters over a user defined time frame. A similar idea using over clustering has been applied in [56] too, where data is incrementally loaded and modeled using gaussians. At the first stage the gaussians models are allowed to overfit

the data and then later merged at a second stage to output the final models. However, FCM tries to optimize a different objective function and methods to summarize and merge fuzzy clustering solutions will not be the same as for other algorithms. It should be noted that the online FCM (OFCM) algorithm is only concerned about producing clustering quality, at the end of the stream, as good as clustering the entire stream. So, it is not concerned about producing clustering solutions at intermediate time instants like a typical streaming algorithm [57]. It could be converted into a purely streaming algorithm, as done in [57], by merging intermediate clustering solutions over a given time horizon. Unlike in [57], this merging could be done without user intervention. However, that would require the dynamic knowledge of the number of macro clusters as time changes.

Cluster validity algorithms, which estimate the number of clusters, may be integrated to obtain information on how the number of clusters changes with time. As discussed earlier, our streaming FCM (SFCM) handles the case of evolving distribution by using varying amounts of history, but assumes the number of clusters to be fixed. Obviously, the idea of integrating a cluster validity metric algorithms could be extended to SFCM also. It will then handle the scenario where the number of clusters changes with time. We will compare the performance of SFCM with OFCM and will show that SFCM still performs reasonably well, with appropriate usage of history, when compared to clustering the entire stream even if data comes in problematic order, where the number of classes vary from chunk to chunk. Obviously, integrating cluster validity metric algorithms in the future may be a way to improve performance.

5.4.1 Results and Discussion

Experiments were performed on 9 real data sets. Six of them, MRI-4, MRI-5, MRI-6, MRI-7, MRI-8, and MRI-9 were the same as used for the streaming FCM

(SFCM) in the previous section. Data was fetched exactly as done for the SFCM experiments, that is 5% data loaded in each chunk. The other three data sets were the Iris, KDD, and the Plankton data sets. The Iris, KDD, and Plankton data sets were described in Chapters 3 and 4. We clustered Iris into 3 clusters, KDD into 10, and Plankton into 12 clusters. The value of m used both for Iris and Plankton was 2, where 1.2 was used for the KDD data set. Like the MRI data sets, 5% of the data was fetched in each chunk for the KDD and the Plankton data sets. For the Iris experiment, both for SFCM and OFCM, we fetched 25 examples in each chunk. So, it required 6 time instants to fetch the full data set. In the Iris data set the first 50 examples were from class1, second 50 from class 2, and the third 50 from class 3. So, fetching 25 examples in each chunk means at any time instant data will come from one class only. This will be a hard case for any algorithm. We will also compare the performance of SFCM and OFCM under this setting also. We also compared the results of the single pass FCM (SPFCM) algorithm on these data sets with the same chunk size as used for SFCM and OFCM experiments. The single pass algorithm (SPFCM) was run with and without scrambling the data. Results of both experiments will be reported.

The results of OFCM and SPFCM were compared with the clustering quality obtained at the end of the stream for the SFCM algorithm. In Table 5.1, we show the difference in quality computed according to (4.5), that is, the difference in R_m value expressed in percentage, of the OFCM, SPFCM, and SFCM algorithms from the quality obtained by clustering all the data at once. All results are average of 32 random experiments, each starting with a random initialization at the beginning of the stream. On each data set all algorithms had the same random initializations. In the table HISn means SFCM using history of n chunks. For the single pass experiments, in the table SPFCM denotes clustering was done on randomly reordered data sets (as

discussed in Chapter 4), while SPFCM" means data was clustered the way it comes; the way SFCM and OFCM algorithms fetches data. We already saw in Chapter 4 that SPFCM gives unpredictable results with unpredictable data order. Here we study it in detail to understand the limitations of SPFCM under a stream setting, where data might come in an unpredictable order and one may have to process it as it comes.

In Table 5.1, we see SPFCM, as expected, provides unpredictable clustering quality when it processes data as it comes (SPFCM" column in the table). When the same data sets were scrambled, it always produced excellent quality (SPFCM column in the table) as discussed in Chapter 4. So, if a data stream comes in a "good" order, created here by randomly reordering the examples, SPFCM could be used; otherwise, SPFCM may give unpredictable results. For processing data in a typical stream setting (processed as it comes), either SFCM (with appropriate history) or OFCM can be used. The results in Table 5.1 show that OFCM is always superior to SFCM in producing a clustering solution as good as clustering the full stream at once. OFCM always obtained good quality partitions; even for the Iris experiment the quality difference is only 0.21661%. It seems on average, except on the Plankton data set, SFCM produces quality comparable to the FCM with usage of history equal to 1. It is interesting to note that on the Plankton data set quality improves with more history. Generally, usage of history greater than or equal to 2 resulted in poor partitions, at least in this context of producing clustering quality (at the end of stream) as good as clustering the entire data stream at once. On the KDD data sets, any amount of history usage gives good quality; however, with HIS1 and HIS2 average quality was even better than the average quality of FCM. However, the quality of OFCM always looks better than SFCM in producing clustering quality as good as clustering the full data set. Thus, OFCM can be thought of as a generalized single pass FCM algorithm

that like streaming algorithms can process data as it comes, while at the end of the stream it can produce clustering quality as good as clustering the entire data stream.

In summary, SPFCM can be used to produce excellent quality partitions, provided there is a way to ensure data comes in a "good" mixture. One way to create a "good" mixture is to randomly reorder it, but streaming data has to be processed as it comes. If the data stream is not well mixed or reordering is difficult, OFCM can be used to produce good quality partitions, as it processes data as it comes. While SFCM with appropriate amounts of history may also be used to produce comparable clustering quality as when clustering the entire stream, its design goals were different. It was designed to cluster evolving data streams, where clustering solutions at each time instant will be more influenced by current data rather than history. It also has the flexibility of using varying amounts of history.

Table 5.1 Difference in Quality (in percentage) of the SFCM, OFCM, and SPFCM algorithms compared to clustering all the stream at once. SPFCM" means clustering without scrambling the data.

	HIS1 (%)	HIS2 (%)	HIS3 (%)	HIS4 (%)	HIS5 (%)	SPFCM' (%)	SPFCM (%)	OFCM (%)
MRI-4	0.7082	1.0378	6.6540	12.9819	17.6392	8.8818	0.0026	0.17447
MRI-5	2.4084	3.8948	11.1541	18.0348	23.1885	10.4976	0.0011	0.17691
MRI-6	6.7014	4.2827	10.2577	15.7393	19.5325	8.2708	0.0009	1.1098
MRI-7	1.2444	22.0437	69.0189	109.1186	141.9229	84.720	0.0065	0.4390
MRI-8	0.5840	15.7915	41.5251	63.6055	82.3348	47.623	0.0027	0.2398
MRI-9	0.5464	13.0416	35.9483	53.7082	67.0518	40.582	0.0141	0.2995
Iris	5.2772	2.3517	90.0830	91.2483	91.5650	79.6733	0.1117	0.21661
KDD	-0.0567	-0.0585	0.0169	0.0127	0.0098	-0.1315	-0.0324	-0.07934
Plankton	14.2393	11.7439	10.1547	8.7612	8.6569	4.02337	0.0046	2.95274

5.5 Generalizing to Other Soft Clustering Algorithms

In this Chapter and in Chapter 3, we presented the single pass FCM, streaming FCM, and the online FCM algorithms for large/streaming data sets. In this section we

provide mathematical equations which would enable other iterative fuzzy/possibilistic clustering algorithms to be turned into similar algorithms as was done for FCM. We introduce transformed equations for summarizing clustering solutions for possibilistic c-means (PCM) [83], and the Gustafson-Kessel algorithm (GK) [84] also. Usage of history/weighted points then can be done in the same way as we did for the single pass, streaming, and the online FCM algorithm.

5.5.1 Possibilistic C Means

The objective function (J_m) minimized by PCM is defined as follows:

$$J_m\left(U,V,\eta\right)=\sum_{i=1}^{c}\sum_{k=1}^{n}U_{ik}^{m}D_{ik}(x_k,v_i)+\sum_{i=1}^{c}\eta_i\sum_{k=1}^{n}(1-U_{ik})^m \tag{5.5}$$

$U$and V can be calculated as:

$$U_{ik}=[1+(\frac{D_{ik}^2}{\eta_i})^{\frac{1}{m-1}}]^{-1} \tag{5.6}$$

$$v_i=\frac{\sum\limits_{j=1}^{n}(u_{ij})^m x_j}{\sum_{j=1}^{n}(u_{ij})^m} \tag{5.7}$$

Where,

U_{ik} : is the membership value of the k^{th} example, x_k, in the i^{th} cluster.

v_i : is the i^{th} cluster centroid.

n : is the number of examples

c: is the number of clusters

$D_{ik}\left(x_k,v_i\right)=\left\|x_k-v_i\right\|^2$: is the norm.

$\eta=(\eta_i,\eta_2,\ldots,\eta_c)^{\mathbf{T}};\eta_i\in\Re^+$ is the i^{th} penalty term. Each η_i is a fixed user specified positive weight.

The weighted version of PCM (allowing examples to have weights) is:

$$Jw_m\,(U,V) = \sum_{i=1}^{c}\sum_{k=1}^{n} U_{ik}^m w_k D_{ik}(x_k, v_i) + \sum_{i=1}^{c}\eta_i \sum_{k=1}^{n} w_k (1-U_{ik})^m \quad (5.8)$$

$U$and V can now be calculated as:

$$U_{ik} = [1 + (\frac{D_{ik}^2}{\eta_i})^{\frac{1}{m-1}}]^{-1} \quad (5.9)$$

$$v_i = \frac{\sum_{j=1}^{n} (u_{ij})^m w_j x_j}{\sum_{j=1}^{n} w_j (u_{ij})^m} \quad (5.10)$$

where w_j is the weight of the j^{th} example.

5.5.2 Gustafson-Kessel (GK) Clustering

This is the same as FCM except that $D_{ik}\,(x_k, v_i) = \|x_k - v_i\|_{A_i}$:,

where $A_i = [\rho_i det(C_i)]^{1/p} C_i^{-1}, 1 \leq i \leq c$ and C_i is the fuzzy covariance matrix of cluster i,

$$C_i = \sum_{k=1}^{n} U_{ik}^m (x_k - v_i)(x_k - v_i)^{\mathbf{T}} / \sum_{k=1}^{n} U_{ik}^m \quad (5.11)$$

where $1 \leq i \leq c; m \geq 1$

$\rho_i = 1, \forall i$ is a typical choice. The change to C_i shown in (5.12) below takes care of the weighting issue for the distance function.

$$C_i = \sum_{k=1}^{n} U_{ik}^m w_k (x_k - v_i)(x_k - v_i)^{\mathbf{T}} / \sum_{k=1}^{n} w_k U_{ik}^m \quad (5.12)$$

where $1 \leq i \leq c; m \geq 1$.

CHAPTER 6

CONCLUSIONS

In Chapter 3, we proposed a method for merging clustering solutions in an ensemble in a scalable framework in terms of time and space complexity. The centroid based ensemble merger algorithms, Bipartite merger (BM) and Metis merger (MM), were shown to be competitive or better than a label vector based ensemble merger algorithm, MCLA. Quantitative differences in the quality of our algorithms compared to MCLA shows that for fuzzy ensembles, MM, on average is better than MCLA , and for hard ensembles it is as good as MCLA. Nevertheless, the overall quality difference among BM, MM, and MCLA in all cases was low. We also compared the ensemble merger algorithms, BM, MM, and MCLA, with the GC (global clustering), BC (base clustering), and SP (a single pass clustering algorithm for hard ensembles only). Performance of BM and MM, when compared to GC, BL, and SP, is better than or as good as MCLA. Among BM and MM, quantitative evaluation on average indicates MM is slightly better than BM. In summary, our centroids based ensemble merging algorithms were better than or as good as the label vector based ensemble merging algorithm, MCLA, while providing very large speed ups of the order of hundreds of thousands times on medium or large data sets. The memory requirements of our algorithms were also hundreds of thousand times less than MCLA.

Although, we form the ensemble from disjoint subsets, we have shown, on average, a final clustering of better or equivalent quality can be achieved compared to global clustering and average base clustering solutions in the ensemble; however, global

clustering using fuzzy-k-means seems to be slightly better, on average, than any ensemble merging algorithm.

Comparison of multiple partition combining algorithms with SP reveals that the single pass hard c means algorithm was competitive to ensemble merging algorithms for scaling to large data sets, provided the whole data set is available on a single disk; though, on average over all data sets and experiments all ensemble merging algorithms were slightly better than SP. Our algorithms are also capable of detecting and removing malformed clusters from the ensemble to provide a robust framework. In this work, our focus was to show that using the same base clustering solutions a centroid based ensemble was competitive or better compared to a label vector based ensemble. It is not always necessary to create an ensemble from a disjoint data set, but we did it to restrict the size of ensemble, a way of scaling it. Future experiments could be done to create an ensemble in a different way, that is, from the full data set or overlapped examples to see how they affect quality compared to global clustering (GC), average base clustering solutions (BC), and single pass clustering algorithm (SP). Future work could also include partitioning each subset with a different number of clusters, and then partitioning the ensembles by graph partitioning packages or methods that cluster them into unequal size chains/groups. It will then address those scenarios in which the number of clusters in each base clustering solutions are not the same.

In Chapter 4, a single pass fuzzy c means algorithm (SPFCM) was introduced for large or very large data sets, which will produce a final clustering in a single pass through unloadable data on disk with limited memory allocated. We neither kept any complicated data structures nor use any complicated compression techniques, yet achieved excellent quality partitions, with an average quality almost the same as FCM, by loading as little as 1% of the data for medium/large data sets and 10% for

small data sets. Moreover, our simple single pass fuzzy c means algorithm provides significant speed up even when compared to clustering all the data at once in memory. So, besides using it for clustering large data sets it can also be used for speed up purposes even for data that can be fully loaded into memory. As SPFCM would enable segmenting/partitioning a large amount of data accurately and quickly, it might open new opportunities for fuzzy 3D modeling/segmentation. The single pass fuzzy c means algorithm (SPFCM) produces excellent quality partition when data comes in a "good mixture" of all classes, here we created by randomly reordering the data. Unpredictable data orders may result in unpredictable cluster quality. So, we proposed streaming algorithms which can process data as it comes.

In Chapter 5 algorithms to cluster data streams were provided. Single pass algorithms may be useful for scaling purposes, but a single pass approach is not suitable for clustering stream data. This is because a data stream might evolve over time and the outdated history in a single pass approach might dominate incoming data. A good streaming algorithm should be able to respond to an evolving distribution and summarize the data seen so far. We proposed a streaming variant of the fuzzy c means algorithm (SFCM) and showed that a tradeoff exists between summarization of data seen and responsiveness to an evolving distribution. The tradeoff depends upon the amount of history used. Future work could include providing theoretical justification for this tradeoff. We tested both on a real and artificial data set to evaluate our algorithm and compared with FCM. One advantage of the streaming algorithm is that it can process data as it comes and is a true online algorithm in the sense that it does not require random data access, which many single pass or other scalable algorithms assume. As fuzzy c means is widely used for segmentation purposes, this streaming variant may be embedded directly into hardware for real time processing. We also proposed a special purpose streaming algorithm, known as online FCM (OFCM), to

produce clustering quality as good as clustering the entire stream. This kind of algorithm will be useful because large data sets on disk can be clustered by a streaming approach to produce clustering quality as good as clustering the whole data set. Unlike typical streaming algorithms, where clustering solutions are generally dominated by currently arriving data, in this algorithm (OFCM) all data chunks will contribute equally in obtaining final clustering solution, and like streaming algorithms it would be able to process data as it comes. Finally, we generalized our methods to other iterative fuzzy/possibilistic clustering algorithms by introducing transformed equations for possibilistic c-means (PCM), and the Gustafson-Kessel algorithm (GK) also.

The single pass FCM (SPFCM) algorithm can be used to produce excellent quality partitions, if data comes in a "good" mixture. Data streams may not always come well mixed and sometimes reordering may be difficult. Sometimes, advance knowledge about the order of the data may not be available, which is typical for data streams. In those cases, OFCM can be used , as it produces good quality partitions by processing data as it comes. SFCM has the flexibility of using varying amounts of history. It was designed to cluster evolving data streams, where clustering solutions at each time instant will be more influenced by current data chunks. With appropriate amounts of history, it may also be used to produce comparable partition quality as obtained by custering an entire stream.

Clustering algorithms are an important tool for data analysis. Thus, different frameworks/algorithms have been proposed to address scalability issues in different settings. For example, cluster ensembles addresses merging multiple clustering solutions generated from large data sets, the single pass algorithm addressed clustering large or very large data sets on disk in a single scan, and streaming algorithms dealt with clustering infinite data streams, where data might evolve with time and may

come in any order. Our ideas may be extended to other unsupervised learning algorithms also.

REFERENCES

[1] AK Jain and RC Dubes. Algorithms for Clustering Data. Prentice Hall. *Englewood Cliffs NJ, USA*, 1988.

[2] A. Strehl and J. Ghosh. Clusters ensembles- a knowledge reuse framework for combining multiple partitions. *Journal of Machine learning Research*, 3:583–617, 2002.

[3] B. Long, Z. (Mark) Zhang, and Philip S. Yu. Combining Multiple Clusterings by Soft Correspondence. *ICDM*, pages 282–289, 2005.

[4] X.Z Fern and C.E Brodley. Solving cluster ensemble problem by bipartite graph partitioning. *ICML*, 2004.

[5] B. Fischer and J.H Buhmann. Path-based clustering for grouping of smooth curves and texture segmentation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 25(4):513–518, 2003.

[6] A. Topchy, A.K Jain, and W. Punch. A mixture model for clustering ensembles. *In Proc. SIAM Intl. Conf. on Data Mining (SDM)*, pages 379–390, 2004.

[7] S. Dudoit and J. Fridlyand. Bagging to improve the accuracy of a clustering procedure. *Bioinformatics*, 19(9):1090–1099, 2003.

[8] A. Topchy, A.K Jain, and W. Punch. Combining Multiple Weak Clusterings. *IEEE Intl. Conf. on Data Mining*, pages 331–338, 2003.

[9] A.L.N Fred. Finding Consistent Clusters in Data Partitions. *Proc Int. Workshop on multiple Classifier Systems*, pages 309–318, 2002.

[10] A. Topchy, A. K. Jain, and W. Punch. Clustering Ensembles: Models of Consensus and Weak Partitions. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 27(12):1866–1881, 2005.

[11] Muna Al-Razgan and Carlotta Domeniconi. Weighted Clustering Ensembles. *SDM*, 2006.

[12] Tilman Lange and Joachim M. Buhmann. Combining partitions by probabilistic label aggregation. *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 147–156, 2005.

[13] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. Clustering Aggregation. *ICDE*, pages 341–352, 2005.

[14] P. Hore, Lawrence Hall, and Dmitry Goldgof. A Cluster Ensemble Framework for Large Data sets. *IEEE International Conference on Systems, Man, and Cybernetics*, 2006.

[15] Alexander P. Topchy, Martin H. C. Law, Anil K. Jain, and Ana L. Fred. Analysis of Consensus Partition in Cluster Ensemble. *ICDM*, pages 225–232, 2004.

[16] Ana L.N. Fred and A. K. Jain. Combining Multiple Clusterings Using Evidence Accumulation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 27(6):835–850, 2005.

[17] Behrouz Minaei-Bidgoli, Alexander Topchy, and William F. Punch. Ensembles of partitions via data resampling. *ITCC*, pages 188–192, 2004.

[18] P. Viswanath and Karthik Jayasurya. A Fast and Efficient Ensemble Clustering Method. *ICPR*, pages 720–723, 2006.

[19] H. Spath. Cluster Analysis Algorithms for Data Reduction and Classification. *Ellis Horwood, Chichester, UK*, 1980.

[20] I. Davidson and A. Satyanarayana. Speeding up K Means Clustering Using Bootstrap Averaging. *Proc. IEEE ICDM 2003 Workshop on Clustering Large Data Sets*, page 1625, 2003.

[21] F. Farnstrom, J. Lewis, and C. Elkan. Scalability of Clustering Algorithms Revisited. *SIGKDD Explorations*, pages 51–57, 2000.

[22] S.Eschrich, J.Ke, L.O.Hall, and D.B. Goldgof. Fast Accurate Fuzzy Clustering through Data Reduction. *IEEE Transactions on Fuzzy Systems*, 11:262–270, 2003.

[23] V. Ganti, Gehrke, J., and R. Ramakrishnan. Mining very large databases. *Computer*, pages 38–45, 1999.

[24] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH : An Efficient Data Clustering Method for Very Large Databases. *Proc. ACM SIGMOD Int'l. Conf. on Management of Data, ACM Press*, pages 103–114, 1996.

[25] V. Ganti, R. Ramakrishnan, J. Gehrke, A. L. Powell, and J. C. French. Clustering Large Datasets in Arbitrary Metric Spaces. *Proc. 15th Int'l. Conf. on Data Engineering*, IEEE CS Press, pages 502–511, 1999.

[26] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. *Proc. 4th Int'l. Conf. Knowledge Discovery and Data Mining, AAAI Press*, pages 9–15, 1998.

[27] P. Domingos and G. Hulten. A General Method for Scaling Up Machine Learning Algorithms and its Application to Clustering. *Proc. Eighteenth Int'l. Conf. on Machine Learning*, pages 106–113, 2001.

[28] N.R. Pal and J.C. Bezdek. Complexity reduction for large image processing. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, pages 598–611, 2002.

[29] Cheng T.W., Goldgof D.B., and Hall L.O. Fast fuzzy clustering. *Fuzzy Sets and Systems*, pages 49–56, 1998.

[30] P. Mitra, C.A. Murthy, and Sankar K. Pal. Density-Based Multiscale Data Condensation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 24(6):734–747, 2002.

[31] Jun Liu, Jim P.Y. Lee, Lingjie Li, Zhi-Quan Luo, and K. Max Wong. Online Clustering Algorithms for Radar Emitter Classification. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 27(8):1185–1196, 2005.

[32] Inderjit S. Dhillon and Dharmendra S. Modha. A Data-Clustering Algorithm On Distributed Memory Multiprocessors. *Proc. of Large-scale Parallel KDD Systems Workshop, ACM SIGKDD*, pages 245–260, 1999.

[33] Joydeep Ghosh and Srujana Merugu. Distributed Clustering with Limited Knowledge Sharing. *Proc. 5th International Conference on Advances in Pattern Recognition*, pages 48–53, 2003.

[34] Joydeep Ghosh, Alexander Strehl, and Srujana Merugu. A Consensus Framework for Integrating Distributed Clusterings Under Limited Knowledge Sharing. *Proc. National Science Foundation (NSF) Workshop on Next Generation Data Mining*, pages 99–108, 2002.

[35] Eshref Januzaj, Hans-Peter Kriegel, and Martin Pfeifle. Towards Effective and Efficient Distributed Clustering. *Proc. Int. Workshop on Clustering Large Data Sets, 3rd IEEE International Conference on Data Mining*, pages 49–58, 2003.

[36] Matthias Klusch, Stefano Lodi, and Gianluca Moro. Distributed Clustering Based on Sampling Local Density Estimates. *Proc. Eighteenth International Joint Conference on Artificial Intelligence*, pages 485–490, 2003.

[37] H. Kriegel, P. Kroger, A. Pryakhin, and M. Scubert. Effective and Efficient Distributed Model-based Clustering. *ICDM*, pages 258–265, 2005.

[38] Huidong Jin, Man-Leung Wong, and K.-S. Leung. Scalable Model-Based Clustering for Large Databases Based on Data Summarization. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 27(11):1710–1719, 2005.

[39] P. Hore and L. Hall. Scalable Clustering: A Distributed Approach. *FUZZ-IEEE*, pages 143–148, 2004.

[40] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An Efficient Clustering Algorithm for Large Databases. *In Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 73–84, 1998.

[41] Raymond T. Ng and Jiawei Han. CLARANS: A Method for Clustering Objects for Spatial Data Mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

[42] Chetan Gupta and Robert Grossman. GenIc: A Single Pass Generalized Incremental Algorithm for Clustering. *Proceedings of the Fourth {SIAM} International Conference on Data Mining (SDM)*, pages 22–24, 2004.

[43] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-Data Algorithms for High-Quality Clustering. *Proceedings of IEEE International Conference on Data Engineering*, 2002.

[44] Richard J. Hathaway and James C. Bezdek. Extending Fuzzy and Probabilistic Clustering to Very Large Data Sets. *Journal of Computational Statistics and Data Analysis*, 2006.

[45] David Altman. Efficient Fuzzy Clustering of Multi-spectral Images. *IEEE FUZZ*, 1999.

[46] John F. Kolen and Tim Hutcheson. Reducing the Time Complexity of the Fuzzy C-Means Algorithm. *IEEE Transactions on Fuzzy Systems*, 10:263–267, 2002.

[47] Christian Borgelt and Rudolf Kruse. Speeding Up Fuzzy Clustering with Neural Network Techniques. *Fuzzy Systems*, 2:852–856, 2003.

[48] A. Bensaid, J.Bezdek, L.O. Hall, and L.P. Clarke. Partially Supervised Clustering for Image Segmentation. *Pattern Recognition*, 29(5):859–871, 1996.

[49] S. Rahmi, M. Zargham, A. Thakre, and D. Chhillar. A Parallel Fuzzy C-Mean Algorithm for Image Segmentation. *Fuzzy Information*, 1:234–237, 2004.

[50] Alfredo Petrosino and Mauro Verde. P-AFLC: a parallel scalable fuzzy clustering algorithm. *ICPR*, 1:809–812, 2004.

[51] S. Asharaf and M. Narasimha Murty. An adaptive rough fuzzy single pass algorithm for clustering large data sets. *Pattern Recognition*, 36, 2003.

[52] Traven H. G. C. A neural network approach to statistical pattern classification by semiparametric estimation of probability density functions. *IEEE Transcations on Neural Networks*, 2:366–377, 1991.

[53] Bo Thiesson, Christopher Meek, and David Heckerman. Accelerating EM for Large Databases. *Machine Learning Journal*, 45:279–299, 2001.

[54] R. Neal and G. Hinton. A View of the EM Algorithm that Justifies Incremental, Sparse, and other Variants. *Learning in Graphical Models*, pages 355–368, 1998.

[55] Bradley, P.S., U.M. Fayyad, and Reina C.A. Clustering very large databases using EM mixture models. *ICPR*, 2:76–80, 2000.

[56] Karkkainen and Franti. Gradual model generator for single-pass clustering. *ICDM*, pages 681–684, 2005.

[57] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. *In Proc. of VLDB*, 2003.

[58] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for projected clustering of high dimensional data streams. *In Proc. of VLDB*, 2004.

[59] J. Yang. Dynamic clustering of evolving streams with a single pass. *In Proc. of ICDE*, 2003.

[60] Feng Cao, Martin Estery, Weining Qian, and Aoying Zhou. Density-Based Clustering over an Evolving Data Stream with Noise. *SDM*, 2006.

[61] O. Nasraoui, C. Cardona, C. Rojas, and F. Gonzlez. Tecno-streams: tracking evolving clusters in noisy data streams with a scalable immune system learning model. *In Proc. of ICDM*, page 235242, 2003.

[62] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. OCallaghan. Clustering data streams:theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, page 515528, 2003.

[63] Prodip Hore, Lawrence Hall, and Dmitry Goldgof. Single Pass Fuzzy C Means. *FUZZ-IEEE (accepted)*, 2007.

[64] B. Dai, J. Huang, M. Yeh, and M. Chen. Clustering on demand for multiple data streams. *In Proc. of ICDM*, page 367370, 2004.

[65] Jurgen Beringer and Eyke Hullermeier. Online clustering of parallel data streams. *Data Knowl. Eng*, 58:180–204, 2006.

[66] Chunyu Yang and Jie Zhou. HClustream: A Novel Approach for Clustering Evolving Heterogeneous Data Stream. *ICDM*, pages 682–688, 2006.

[67] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *In Proc. of KDD*, 1996.

[68] Kyungmin Cho, Sungjae Jo, Hyukjae Jang, Su Myeon Kim, and Junehwa Song. DCF: An Efficient Data Stream Clustering Framework for Streaming Applications. *Book: Database and Expert Systems Applications: Publisher Springer Berlin / Heidelberg*, 4080:114–122, 2006.

[69] Modified Fuzzy C-Means Clustering Algorithm for Real-Time Applications. *Book Field-Programmable Logic and Applications: Publisher Springer Berlin / Heidelberg*, 2778, 2003.

[70] Peter X. Liu and Max Q.-H. Meng. Online Data-Driven Fuzzy Clustering With Application To Real-Time Robotic Tracking. *IEEE Transaction On Fuzzy Systems*, 12, 2004.

[71] Richard Nock and Frank Nielsen. On Weighting Clustering. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 28(8):1223–1235, 2006.

[72] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. *Technical Report TR 98-019, Department of Computer Science, University of Minnesota*, 1998.

[73] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96129, 1998.

[74] G. Karypis and V. Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 1998.

[75] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. *34th Design and Automation Conference*, page 526529, 1997.

[76] H.W Kuhn. The Hungarian Method for the Assignment Problem. *Naval. Res. Logist. Quart*, 2:83–97, 1995.

[77] R. Duda, P. Hart, and D. Stork. Pattern Classification. *A Wiley-Interscience Publication, Second edition*, 2002.

[78] C.J. Merz and P.M. Murphy. *UCI Repository of Machine Learning Databases Univ. of CA., Dept. of CIS, Irvine, CA., http://www.ics.uci.edu/˜ mlearn/MLRepository.html.*

[79] Kdd cup data http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html. 1998.

[80] M. Jenkinson, M. Pechaud, and S. Smith. BET2: MR-based estimation of brain, skull and scalp surfaces. *In Eleventh Annual Meeting of the Organization for Human Brain Mapping*, 2005.

[81] MS Cohen, RM DuBois, and MM Zeineh. Rapid and effective correction of RF inhomogeneity for high field magnetic resonance imaging. *Human Brain Mapping*, 10:204–211, 2000.

[82] Richard J. Hathaway and James C. Bezdek. Optimization of Clustering Criteria by Reformulation. *IEEE Transactions on Fuzzy Systems*, 3:241–245, 1995.

[83] J. B. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press*, pages 281–297.

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

ABOUT THE AUTHOR

Prodip Hore received a Bachelors Degree in Computer Science and Engineering from Institute of Engineering & Management (IEM), Kolkata in 2002 and a M.S. in Computer Science and Engineering from University Of South Florida in 2004. During his masters thesis he worked on designing scalable clustering algorithms and continued on as a Ph.D. student.

Besides working on his dissertation, he also worked on a Magnetic resonance Image segmentation project as a research assistant. He was actively involved in designing various scalable pattern recognition algorithms/systems, some of them have been publicly released. He has also coauthored five publications on scalable algorithms.