Fuzzy Based Scalable Clustering Algorithms for Handling Big Data Using Apache Spark

Neha Bharill, *Member, IEEE*, Aruna Tiwari, *Member, IEEE*, and Aayushi Malviya, *Member, IEEE*

Abstract—A huge amount of digital data containing useful information, called Big Data, is generated everyday. To mine such useful information, clustering is widely used data analysis technique. A large number of Big Data analytics frameworks have been developed to scale the clustering algorithms for big data analysis. One such framework called Apache Spark works really well for iterative algorithms by supporting in-memory computations, scalability etc. We focus on the design and implementation of partitional based clustering algorithms on Apache Spark, which are suited for clustering large datasets due to their low computational requirements. In this paper, we propose Scalable Random Sampling with Iterative Optimization Fuzzy c-Means algorithm (SRSIO-FCM) implemented on an Apache Spark Cluster to handle the challenges associated with big data clustering. Experimental studies on various big datasets have been conducted. The performance of SRSIO-FCM is judged in comparison with the proposed scalable version of the Literal Fuzzy c-Means (LFCM) and Random Sampling plus Extension Fuzzy c-Means (rseFCM) implemented on the Apache Spark cluster. The comparative results are reported in terms of time and space complexity, run time and measure of clustering quality, showing that SRSIO-FCM is able to run in much less time without compromising the clustering quality.

Index Terms—Fuzzy clustering, iterative computation, big data, Apache Spark, distributed machine learning

1 Introduction

A huge amount of data gets collected everyday due to the increasing involvement of humans in the digital space. We share, store and manage our work and lives online. For example, Facebook stores more than 30 Petabytes of data, and Walmart's databases contain more than 2.5 petabytes of data [1]. Such huge amount of data containing useful information is called Big Data. It is becoming increasingly popular to mine such big data in order to gain insights the valuable information that can be of great use in scientific and business applications. Clustering is the promising data mining technique that is widely adopted for mining valuable information underlining unlabeled data. Over the past decades, different clustering algorithms [2], [3] have been developed based on various theories and applications. Among them, partitional algorithms [4] are widely adopted due to their low computational requirements, they are more suited for clustering large datasets [6].

One of the most widely used partitional clustering algorithm is the Fuzzy c-Means (FCM) clustering algorithm proposed by Bezdek [7]. The Fuzzy c-Means clustering algorithm attempts to partition the data points in the set of c fuzzy clusters such that an objective function of a dissimilarity measure is minimized. Many approaches are proposed by the researchers based on partitional clustering for handling large dataset. For example, the literal Fuzzy c-Means with alternating optimization (LFCM/AO) [8] algorithm is one such approach, which performs clustering on entire dataset but it does not work well for big data.

In the literature, many approaches are designed to address the clustering of large datasets. Some of the approaches have been designed that takes the entire data into account during clustering, known as incremental fuzzy clustering algorithms, inspired by the well known Fuzzy c-means. Examples of such algorithms are the single-pass FCM referred to as SPFCM [9] and online FCM called OFCM [10]. These algorithms divide the data into subsets and obtain information from each of these subsets to estimate the c centroids for the entire data. The kernel versions of spFCM and oFCM, called spkFCM and okFCM respectively, are also developed to cluster large data efficiently. Recently, incremental fuzzy clustering for relational data has also been proposed [11]. Two algorithms are developed based on fuzzy c medoids (FCMD) [12] called as On-line fuzzy c medoids (OFCMD) and history-based online fuzzy c medoids (HOFCMD) to cluster large relational datasets. However, some approaches are designed based on the sampling methods are CLARA [13], coresets [14] and CURE [15] algorithms. These algorithms work by randomly selecting a sample of data from a huge dataset and computing cluster centers of this sampled data. The results obtained on this sample are extended to approximate the cluster center of the entire dataset. However, such algorithms result in overlapping cluster centers if the sampled data is not representative of the entire data.

In addition to this, another sampling method designed to address Very Large data is the random sampling plus extension Fuzzy c-Means (rseFCM) [8]. But like other sampling approaches, rseFCM too suffers from overlapping cluster centers since, it performs clustering only on a sample of the Very Large data which may not be representative of the entire data. The problem of overlapping cluster centers is overcome by Random Sampling with Iterative Optimization

• *The authors are with the Department of Computer Science and Engineering, Indian Institute of Technology, Indore 453552, India.*
E-mail: {phd12120103, artiwari, cse1200101}@iiti.ac.in.

Manuscript received 4 Jan. 2016; accepted 18 Oct. 2016. Date of publication 26 Oct. 2016; date of current version 29 Dec. 2016.
Recommended for acceptance by C. Baru.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TBDATA.2016.2622288

Fuzzy c-Means (RSIO-FCM) [16]. It creates various subsets by randomly selecting objects without replacement from the data, thus covering all objects of large data. It then performs clustering on each subset and uses the final cluster center values of one subset to initialize the cluster centers for the next subset. The process is repeated until clustering is performed on all the subsets. However, this approach does not take into consideration the fact that one subset may contain the sample belong to one class and another subset may contain the sample belong to the different class. Therefore, in this case, if the cluster center of the previous subset is fed as input for the clustering of current subset, then it results in a sudden increase in the number of iterations during the clustering of current subset. Furthermore, the obtained location of the cluster center for the current subset is far more deviated from the actual location.

Despite of the rapid development of the clustering algorithms aimed for handling large data, there is a lack of adoption of these techniques in the wider data mining and other application communities for big data problems. A likely reason for this is that these algorithms are not scalable. This prevents many state-of-the-art clustering algorithms from being widely applied at Big-Learning scales. To design such scalable algorithms, big data analytics frameworks are required. Recently, a large number of computing frameworks [17], [18], [19], [20], [21], [22] have been developed for big data analysis, the most famous being MapReduce [17]. This framework is widely used because of its simplicity, generality and maturity. However, MapReduce does not perform well for iterative algorithms [22]. Many frameworks such as Apache Flink [18], Haloop [19] Apache Mahout [20], i^2MapReduce [21] and Apache Spark [22] etc. have emerged in order to overcome the drawbacks of MapReduce. Apache Flink [18] is an open source platform for distributed Big Data analytics. It bridges the gap between MapReduce-like systems and shared-nothing parallel data based systems. Haloop [19] is a modified version of Hadoop MapReduce framework. It inherits the basic distributed computing model and architecture of Hadoop. Apache Mahout [20] is an open source project that is primarily used for creating scalable machine learning algorithms. It uses the Apache Hadoop library to scale effectively in the cloud. i^2MapReduce [21] is an incremental processing extension to MapReduce and widely used framework for mining big data. It relies on the disk-based systems for computation. Apache Spark [22] is an open source cluster computing framework, developed to optimize large-scale interactive computation. Out of the above stated frameworks, Apache Spark works really well for iterative algorithms by supporting in-memory computations, while retaining the scalability, fault tolerance of MapReduce. Spark can perform up to 100 times faster than Hadoop MapReduce and significantly faster than other frameworks [22]. Thus, we focus on the design and implementation of an iterative clustering algorithm based on Apache Spark.

In this paper, we propose Scalable Random Sampling with Iterative Optimization Fuzzy c-Means algorithm (SRSIO-FCM) implemented on an Apache Spark framework. It is designed to deal with the challenges associated with fuzzy clustering for handling big data. The algorithm starts by randomly partition the data into various subsets. Initially, the cluster centers are randomly initialized, for the clustering of first subset. After this, it computes the cluster centers, and membership information for the first subset. The final cluster centers obtained after clustering of first subset are used as an input for the clustering of second subset. Then it finds the cluster centers, and membership information for the second subset. But unlike RSIO-FCM, it does not use these cluster centers as an input for the clustering of third subset. Instead, it combines the membership information of first and second subset to compute the new cluster centers. These cluster centers are then fed as input for the clustering of third subset. This procedure is repeated for the clustering of all the subsequent subsets. In comparison with RSIO-FCM, where the cluster center of one subset is fed as input for the clustering of next subset, it might be possible that two subsets would consist of samples belongs to a distinct class. Due to this reason, the cluster center of one subset fed as an input for the clustering of current subset, will result in slow convergence by taking the higher number of iterations for that subset. Furthermore, it also produces the highly deviated cluster centers for that subset. This problem has been overcome in the proposed SRISO-FCM, because this approach uses the cluster centers obtained with the combined membership information of all the processed subset for the clustering of current subset. The combined membership information of all the processed subset covers the wider sample space. Thus, using the cluster centers obtained with the membership information of wider sample space will avoid the problem of using the highly deviated cluster for the clustering of current subset. In addition to this, it also results in faster convergence by taking the lesser number of iterations for clustering of current subset.

The SRSIO-FCM is implemented on an Apache Spark Cluster so that it can work well for significantly bigger data sets. To establish the comparison, we designed and implemented the scalable model of existing LFCM/AO and rseFCM algorithm on Apache Spark Cluster, termed as SLFCM and srseFCM. The experimental results show that the proposed SRSIO-FCM approach is significantly better than the previous existing approaches.

The remainder of this paper is organized as follows: Section 2, present the details of the related incremental fuzzy approaches reported in the literature. Section 3, presents the brief review of the Apache Spark framework which helps in understanding the working of scalable algorithms. In Section 4, the details of the proposed SRSIO-FCM approach is presented along with the scalable model of LFCM/AO and rseFCM. Section 5, present the time and space complexity of these algorithms. Experiments on several big data sets are conducted and the results are analyzed in Section 6. Section 7 concludes the paper with directions for future work.

2 Related Work

The one of the most widely used fuzzy clustering algorithm is Fuzzy c-Means, proposed by Bezdek [7]. It divides the n-sized data $X = \{x_1, x_2, \dots x_n\}$ into c groups by optimizing the following objective function:

$$J_m(U, V') = \sum_{i=1}^{n} \sum_{j=1}^{c} u_{ij}^m \parallel x_i - v'_j \parallel^2 . \tag{1}$$

Each data-point x_i is an object and has a set of membership degrees $\{u_{ij} : j \in (1, c)\}$ associated with it. Membership degree represents the association between a data-point and a cluster. The membership degrees of all data-points together constitute the membership matrix U. Each cluster is represented by a cluster center, which is nothing but a weighted mean of all data-points belonging to that cluster. The set of cluster centers is represented by $V = \{v_1, v_2, \dots v_c\}$. In this section, three well known fuzzy based clustering algorithms called LFCM/AO, rseFCM and RSIO-FCM is discussed.

2.1 Literal Fuzzy c-Means (LFCM/AO)

LFCM/AO [8], partition the data into clusters such that data-points in a cluster are more similar to each other than to data-points belonging to other clusters. It tries to optimize the objective function in Eq. (1) using alternating optimization [23]. The number of cluster centers c for LFCM/AO is decided by the user and the initial cluster centroids for c clusters are chosen randomly.

We may ease the notation of LFCM/AO in the further discussion by dropping the "/AO" notation unless clarity demands it [8]. The LFCM algorithm is illustrated in Algorithm 1.

Algorithm 1. *LFCM Iteratively Minimize* $J_m(U, V')$

```
Input: $X, V, c, m$
Output: $U, V'$
1: Randomly initialize cluster centers $V = \{v_1, v_2, \dots, v_c\}$.
2: Compute cluster membership.
       $$u_{ij} = \frac{\| x_i - v_j \|^{\frac{-2}{m-1}}}{\sum_{k=1}^{c} \| x_i - v_k \|^{\frac{-2}{m-1}}}, \forall i, j. \quad (2)$$
3: Check the constraint.
       $$\sum_{j=1}^{c} u_{ij} = 1. \quad (3)$$
4: Compute the cluster centers.
       $$v'_j = \frac{\sum_{i=1}^{n} [u_{ij}]^m x_i}{\sum_{i=1}^{n} [u_{ij}]^m}, \forall j. \quad (4)$$
5: if $\| V' - V \| < \epsilon$ then stop, else go to 2.
```

Where, $U = [u_{ij}]_{n \times c}$ is the partition matrix composed of membership degree of data point x_i to each cluster v_j, $m > 1$ is a fuzzification parameter which drastically affects the clustering results, and ϵ denotes a predefined constant which is used as the stopping criteria. We use $\epsilon = 10^{-3}$ in our experiments.

2.2 Random Sampling Plus Extension Fuzzy c-Means (rseFCM)

rseFCM [8] is a sampling based technique which performs clustering on a small subset of the large data and extends the results to the large data. rseFCM assumes that the subset chosen is representative of the entire data set. It selects the sample data points randomly from the data set and calculates the cluster centers for that subset using LFCM. The membership matrix U for all the data-points are calculated using Eq. (3) and final cluster centers are derived using Eq. (4). Algorithm 2, outlines the steps involved in rseFCM.

Algorithm 2. *rseFCM Approximately Minimize* $J_m(U, V')$

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

Note: Once the extension step is completed, the partition U on the full data set is known, then by using U in Eq. (4), the cluster centers V is evaluated.

rseFCM algorithm, does not perform well for big data sets because the randomly chosen data points in a subset is not able to fully capture the object space represented by entire Big Data. The cluster centers of the sampled data may be very different from the cluster centers that would have been derived by performing LFCM over the entire Big Data. The cluster centers on the sampled data may not be able to minimize the objective function.

2.3 Random Sampling with Iterative Optimization Fuzzy c-Means (RSIO-FCM)

Inspired by FCM, RSIO-FCM [16] has been designed to overcome the drawbacks of rseFCM. It is a partitioning based approach which takes into account the entire data while determining the cluster centers. In RSIO-FCM, the entire data is divided into various subsets $X = \{X_1, X_2, \dots X_s\}$. In this approach, the cluster centers for the first subset is chosen randomly from the entire data. Then, final cluster centers, and membership matrix for the first subset, denoted as V_1 and U_1 respectively, are found by applying LFCM. V_1 is used as the initial cluster centers for clustering of the second subset. This procedure is repeated until all the subsets are clustered. The final cluster centers for the entire data is determined by combining the membership matrix of all the subsets and using Eq. (4). Algorithm 3, highlights the steps involved in RSIO-FCM.

Algorithm 3. *RSIO-FCM Iteratively Minimize* $J_m(U, V')$

```
Input : $X, V, c, m$
Output: $U, V'$
1: Load $X$ as $n_s$ sized randomly chosen subsets $X = \{X_1, X_2 \dots\dots X_s\}$.
2: Sample $x_1$ from $X$ without replacement.
3: $U_1, V_1 = LFCM(X_1, V, c, m)$
4: for $p = 2$ to $s$ do
       $U_p, V_p = LFCM(X_p, V_{p-1}, c, m)$
   end for
5: Compute the partition on full data set
       $$U = \sum_{i=1}^{s} U_l. \quad (5)$$
6: Compute cluster center $V'$ with partition on full data set using Eq. (4).
7: Compute the objective function using Eq. (1).
8: Return $U, V'$
```

In this approach, the data set is randomly partitioned into various subsets. It may be possible that the two subsets

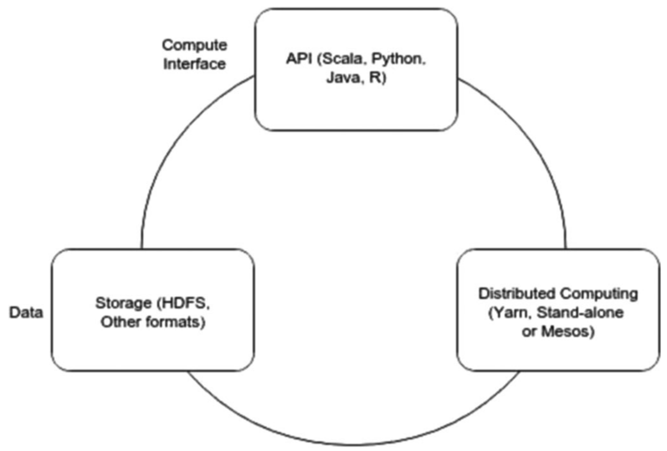

Compute Interface
API (Scala, Python, Java, R)
Data
Storage (HDFS, Other formats)
Distributed Computing (Yarn, Stand-alone or Mesos)

Fig. 1. Features of Apache Spark.

contain the data points belongs to the distinct class. Therefore, the cluster center of one subset may be significantly different from the cluster centers of the other subset. If the final cluster centers of one subset are used as the initial cluster centers for the clustering of current subset, then the number of iterations during clustering of current subset will increase significantly.

To overcome the problem of RSIO-FCM algorithm, in Section 4.2, we propose the Scalable Random Sampling with Iterative Optimization Fuzzy c-Means (SRSIO-FCM) algorithm for handling big data. Before presenting the description of proposed work, we move on to description of Apache Spark framework, which helps to understand the working of the proposed approach on Apache Spark cluster.

3 APACHE SPARK

Apache spark is originally developed at UC Berkeley in 2009, it is a powerful open source processing engine built for sophisticated data analysis. It has quickly become one of the most well recognized tools, giving tough competition to Hadoop MapReduce. It was adopted by IT giants such as Yahoo, Baidu and Tencent eagerly deployed Spark on a massive scale. They collectively process petabytes of data on their Spark clusters.

Spark is optimized for iterative algorithms and interactive data analysis, which perform cyclic operations on the same set of data. Fig. 1 shows the features of Spark. There are various advantages of using Spark [22], listed as follows:

1) Speed: Sparks exploits basic features such as in-memory computation, RDDs and several other optimizations making it up to 100 times faster than Hadoop MapReduce for large scale data processing. It's also considerably faster when the data is stored on the disk [24].
2) Ease of Use: Spark has easy-to-use APIs in Python, Scala, Java and R. It also has data frame APIs for manipulating semi-structured data.
3) A Unified Engine: Spark provides a rich support for SQL queries, machine learning, streaming data and graph processing through its higher-level libraries.

Spark cluster consists of many machines, each of which is referred to as a node. There are two main components of Spark: Master and Workers. There is only one master node and it assigns jobs to the slave nodes. However, there can be any number of slave nodes. Data can be stored in HDFS [25] or in local machine.

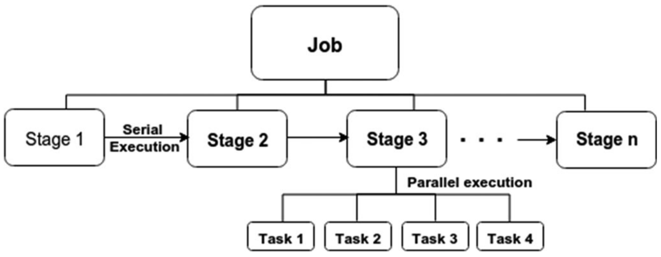


Fig. 2. Running a job over Apache Spark.

A job reads data from the local machine/HDFS, performs computations on it and/or writes some output. Fig. 2 describes how the master node divides the job among the worker nodes. A Driver process executing on the master node is responsible for running this job over the Spark Engine. Each job is partitioned into a number of stages. Each stage can either be a map or a reduce stage. One stage may be dependent on the results of a previously occurring stage. All stages are executed serially. The data are logically divided into several chunks, and these chunks are assigned to different worker nodes. A task performs the set of operations, defined by a stage, on a chunk. These tasks are performed by executor processes on a worker node. Only one task is performed on one partition by each executor.

Spark can run over a variety of cluster managers, a simple cluster manager included in Spark itself known as Standalone Scheduler, on Hadoop YARN [26], or on Apache Mesos [27]. For distributed data storage, spark can interface with a wide variety, including Hadoop Distributed File System (HDFS) [28], HBase [29], Hive [30] Tachyon [31], and any Hadoop data source. In this paper, for the implementation of the proposed algorithm, we use Spark on YARN and HDFS for distributed data storage.

Spark performs better than MapReduce due to the following reasons:

1) Spark does in-memory computations, i.e., it copies the data from the distributed physical storage to the much faster logical RAM memory. This helps Spark to eliminate the disc I/O time, making it 100 times faster than Hadoop [24].
2) Resilient Distributed Databases (RDD) [22]: It is the representation of data in object format on which computations can be performed. There are two types of operations that can be performed on an RDD:
    a) Transformations: Transformations are operations on an RDD which result into another RDD. For example, my_data.map(lambda x: 2x) gives an RDD where each element is twice the value of each element in my_data.
    b) Actions: These are operations that are performed only when the program needs to answer a question. For example, my_data.count() counts the number of elements that are present in my_data.
3) Lazy Approach: Spark follows a lazy approach such that it loads and pushes an RDD into the RAM only when an action needs to be performed, not when it encounters an RDD.

For the faster execution of iterative algorithm, in this paper, we choose Spark on Hadoop to implement the proposed algorithm.

4 Proposed Work

In this paper, we proposed a Scalable Random Sampling with Iterative Optimization Fuzzy c-Means named as SRSIO-FCM. It is a scalable model of RSIO-FCM with necessary modifications to tackle the challenges associated with fuzzy clustering of Big Data. Similar to RSIO-FCM, the proposed approach divides the data into various subsets. In this approach, for the clustering of first subset, cluster centers are initialized randomly. After the clustering of first subset, the cluster centers and membership information corresponding to first subset are obtained. For the clustering of second subset, the final cluster centers of first subset is used as the initial cluster centers. After the clustering of second subset, the cluster centers and membership information are obtained. For the clustering of further subsets, the procedure is stated as follows: First, the membership Information of all the processed subsets are combined to find the new cluster centers. Now, these cluster centers are used as the initial cluster centers for the clustering of next subset. Then, after clustering of that subset, the cluster centers and membership information corresponding to that subset are found. The same procedure is repeated for the clustering of the rest of the subsets. The detailed description of the proposed SRSIO-FCM is presented in Section 4.2.

Since SRSIO-FCM is a scalable algorithm, for comparing the performance of SRSIO-FCM with LFCM and rseFCM, we have designed and implemented the scalable model of these algorithms, termed as Scalable Literal Fuzzy c-Means (SLFCM) and Scalable random sampling plus extension Fuzzy c-Means (SrseFCM) respectively. All these algorithms are implemented on Apache Spark. The following sections discuss the design and implementation details of these algorithms. But, before presenting the proposed SRSIO-FCM algorithm, We will demonstrate the SLFCM algorithm which is used in SRSIO-FCM algorithm for the parallel computation of membership information and cluster centers.

4.1 Scalable Version of Literal Fuzzy c-Means (SLFCM)

The SLFCM algorithm is implemented on Apache Spark. Algorithm 4, describes the working of SLFCM Algorithm. In order to design the scalable version of LFCM, we first need to identify the computations in the algorithm, which could be done in parallel and the computations which can be done only in a serial manner. The most computation intensive part of LFCM is calculation of membership degrees of each point with respect to all the cluster centers. For the computation of membership degree, only the data-point and cluster center values are required. In Line 2 of Algorithm 1, the membership degree computation of each data-point is independent of other data-points. Therefore, the computation of the membership degree of different data-points can be performed in parallel on various machines. So, in Line 2 of Algorithm 4, we present the parallel computation of membership information of all the data points using map and reduceByKey function, discussed in Sections 4.1.1 and 4.1.2. Furthermore, in Line 4 of Algorithm 1, membership degrees of all datapoints are needed to update the cluster center values. Thus, Line 4 of Algorithm 1 must be executed after membership degrees of all points have been determined. So, according to this, in Algorithm 4, the membership degree of all the datapoints are combined at the master node and stored in terms of membership information $I^{'}$, which is used in Eq. (6) to update the cluster center v_j'. Then, the difference between the previously initialized cluster center and currently updated cluster center values is computed using Line 4 of Algorithm 4. Repeat this procedure, until no significant change in the values of cluster centers are observed according to Line 5 of the Algorithm 4. Then, all the iterations are performed serially since the updated cluster centers are needed as input for the next iteration. Fig. 3 depicts the workflow of SLFCM.

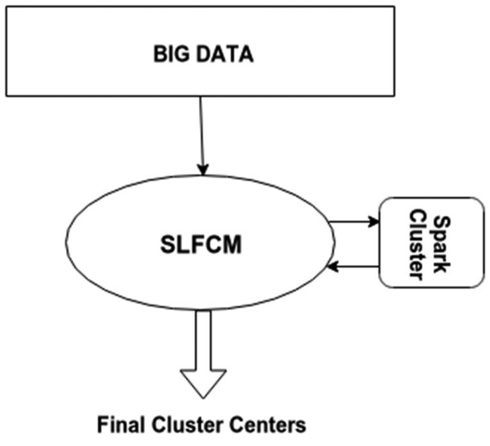


Fig. 3. Workflow of SLFCM algorithm.

Algorithm 4. *SLFCM to Iteratively Minimize* $J_m(U, V^{'})$

```
Input: $X, V, c, m$
Output: $V^{'}, I^{'}$
1: Initialize $V$ by randomly choosing some data points from $X$.
2: Compute membership information.
   $I' = X.map(V).reduceByKey()$
3: Compute Cluster Centers.
   for $< j, < sum\_d_jx, sum\_d_j > >$ in $I^{'}$ do
       $v_j^{'} = \frac{sum\_d_jx}{sum\_d_j}.$   (6)
   end for
4: Calculate change in Cluster Center values.
   $\epsilon' = \| V' - V \|$
5: If $\epsilon' < \epsilon$ : Stop; Else: $V = V'$. Go to 2;
6: Return $V^{'}, I^{'}$
```

In order to reduce the space requirements in the proposed SLFCM algorithm, we avoid storing the membership matrix of data points. For example: In Algorithm 1, the membership matrix U is needed to compute the cluster centers $V^{'}$ using Eq. (4). Instead of storing the huge membership matrix, we compute the value of the parameter present in the numerator and denominator of Eq. (4), i.e., $[u_{ij}]^m x_i$ and $[u_{ij}]^m$ for every point x_i and cluster center v_j, which is represented as $d_{ij}x_i$ and d_{ij} respectively. This avoids the need of storing the huge

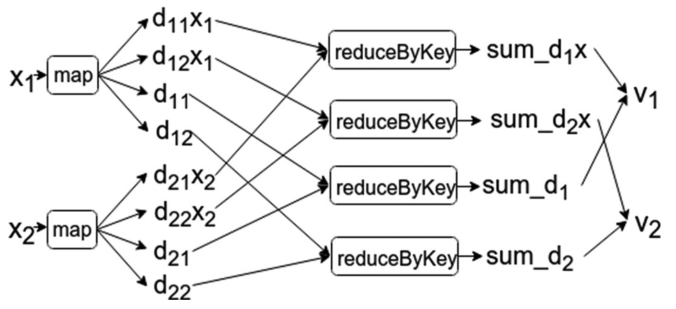


Fig. 4. Storage space optimization by avoiding storage of membership matrix of subsets.

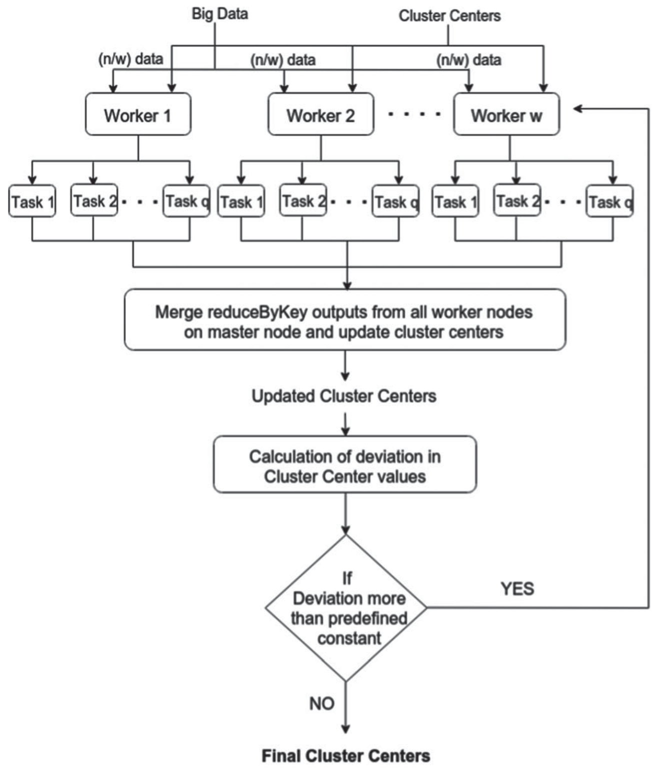
membership martix U. Then, we do the summation of all the $d_{ij}x_i$ values and all the d_{ij} values of the data-points corresponding to cluster center v_j to compute the numerator and denominator of Eq. (4) which is represented as sum_d_jx and sum_d_j respectively and stored in terms of membership information in a variable $I^{'}$. Then, we access these values in Line 3 of Algorithm 4 to compute the new cluster centers using Eq. (6). Due to this, we save a significantly huge amount of space and computational time [32]. Fig. 4 demonstrates the entire procedure of storage space optimization. where, X is an array of data-points such that $X = \{x_1, x_2, \ldots x_n\}$, $V^{'}$ represents the set of final cluster centers and $I^{'}$ represents the membership information of all the data points.

Algorithm 5. *Map(x, V)*

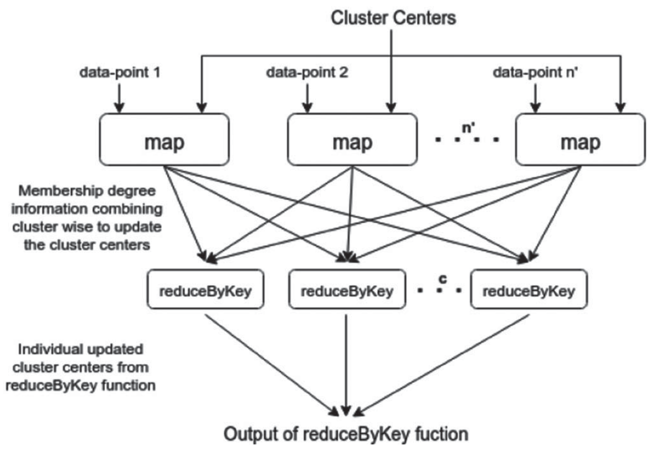
```
Input: $x_i, V$
Output: $< j, <d_{ij}x_i, d_{ij}>>$
1: for each $v_j$ in $V$ do
2:     $j$ = index of cluster center $v$.
3:     $d_{ij}$ = membership degree of $x_i$ with respect to $v_j$.
4:     $d_{ij}x_i = d_{ij} * x_i$
5:     yield $< j, < d_{ij}x_i, d_{ij} >>$
6: end for
```

4.1.1 Map-Function

The worker nodes process small chunks of data which are formed by logically splitting the data on the master node. The membership degrees of a point are determined by using the data-point itself and the cluster center values. Thus, the calculation of membership degrees of a data-point is independent of the calculation of the membership degrees of any other data-point. Therefore, we perform this operation independently for each data-point on a worker node and combine the resulting values on the master node. This produces the same result as that obtained by performing this operation for all data-points on a single machine. The map function defined in Algorithm 5, calculates the membership degree of a data-point with respect to each cluster center and returns each of them individually. Thus, each map function gives as many outputs as the number of clusters. These results are stored in the form of RDDs [22]. Algorithm 5, describes the set of operations performed during a map function to obtain $[u_{ij}]^m x_i$ and $[u_{ij}]^m$ for every point x_i and cluster center v_j as discussed in Line 4 of Algorithm 1. In this algorithm, $[u_{ij}]^m x_i$ and $[u_{ij}]^m$ are represented as $d_{ij}x_i$ and d_{ij} respectively. Fig. 5a shows the overview of working of SLFCM on Apache Spark


Big Data
Cluster Centers
(n/w) data
(n/w) data
(n/w) data
Worker 1
Worker 2
Worker w
Task 1
Task 2
Task q
Merge reduceByKey outputs from all worker nodes on master node and update cluster centers
Updated Cluster Centers
Calculation of deviation in Cluster Center values
If Deviation more than predefined constant
YES
NO
Final Cluster Centers

(a) Overview of working of SLFCM


Cluster Centers
data-point 1
data-point 2
data-point n'
map
n'
Membership degree information combining cluster wise to update the cluster centers
reduceByKey
c
Individual updated cluster centers from reduceByKey function
Output of reduceByKey fuction

(b) Steps performed on a chunk of data given to a task

Fig. 5. SLFCM algorithm implemented on Apache Spark cluster.

cluster. It describes how the data is divided, sent to various worker nodes and the results are collected from them to update the cluster centers. Fig. 5b demonstrates the operations which are performed on a chunk of data.

Where, x_i is the data-point, V is the set of cluster centers represented as $V = \{v_1, v_2, \ldots v_c\}$ and *yield* is used when a function has multiple return values.

4.1.2 ReduceByKey-Function

The Map function results in many key-value pairs having the same key value. ReduceByKey performs operations on the key-value pairs that have the same key. To simplify things, here we describe the operations that are performed on two such key-value pairs. Spark takes care of performing the same operations on all the key-value pairs. To update the cluster center values, we need to find the numerator and denominator in Line 4 of Algorithm 1. Now that we have calculated $[u_{ij}]^m x_i$ and $[u_{ij}]^m$ as $d_{ij}x_i$ and d_{ij} respectively, for every point x_i and cluster center j, during the map phase, we need to add all these values. This is done using the ReduceByKey function which is described in Algorithm 6. This gives us the numerator and denominator of Eq. (4) for every cluster center v_j as sum_d_jx and sum_d_j respectively.

Algorithm 6. *reduceByKey*(a, b)

```
Input: $a, b$ such that $a = < j, <(d_{ij}x_i)_a, (d_{ij})_a >>$, $b = < j, <(d_{ij}x_i)_b, (d_{ij})_b >>$.
Output: $< j, < sum\_d_jx, sum\_d_j >>$
1: $sum\_d_jx = (d_{ij}x_i)_a + (d_{ij}x_i)_b$
2: $sum\_d_j = (d_{ij})_a + (d_{ij})_b$
3: return $< j, < sum\_d_jx, sum\_d_j >>$
```

Where, a and b are output of two map functions, with respect to cluster center j, $(d_{ij}x_i)_a$ and $(d_{ij}x_i)_b$ denotes the value of $d_{ij}x_i$ corresponding to map function outputs a and b respectively, $(d_{ij})_a$ and $(d_{ij})_b$ denotes the value of d_{ij} corresponding to map function outputs a and b respectively. The output of the reduceByKey function is used to calculate the new cluster center values using Eq. (6) on the master node.

4.2 Scalable Random Sampling with Iterative Optimization Fuzzy c-Means (SRSIO-FCM)

SRSIO-FCM works by partitioning the data into various subsets and then performing clustering over all the subsets. The subsets are created by randomly selecting data-points from the entire data set without replacement. The data points present in one subset are distinct from the data points present in another subset. Algorithm 7, discusses the steps involved in SRSIO-FCM. For the clustering of first subset, the cluster centers are randomly initialized. SRSIO-FCM computes cluster centers, and membership information for the first subset X_1, denoted as V' and I' respectively, by applying SLFCM. It then uses V' as initial cluster centers for clustering of second subset. The membership information and cluster centers computed for the second subset X_2, is denoted as I and V', by applying SLFCM.

But unlike RSIO-FCM, SRSIO-FCM does not use the final cluster center of the second subset (V'), as the initial cluster centers for clustering of third subset. This is because, SRSIO-FCM takes into account the fact that random partitioning may result in the subsets containing the data points of distinct classes. Therefore, the cluster centers of these two subsets will be significantly different from each other. So to use a better approximation of cluster centers initialization for the clustering of any subset, SRSIO-FCM avoids using such cluster centers as initial cluster centers. Instead, it combines the membership information of all the processed subsets. Like, here the membership information of first two processed subsets, denoted as I' and I are combined and the new cluster centers are evaluated using the Eq. (6). These cluster centers provide a better estimation to the actual cluster centers, since they are calculated using the combined membership information of a larger number of data points which covers the wider sample space.

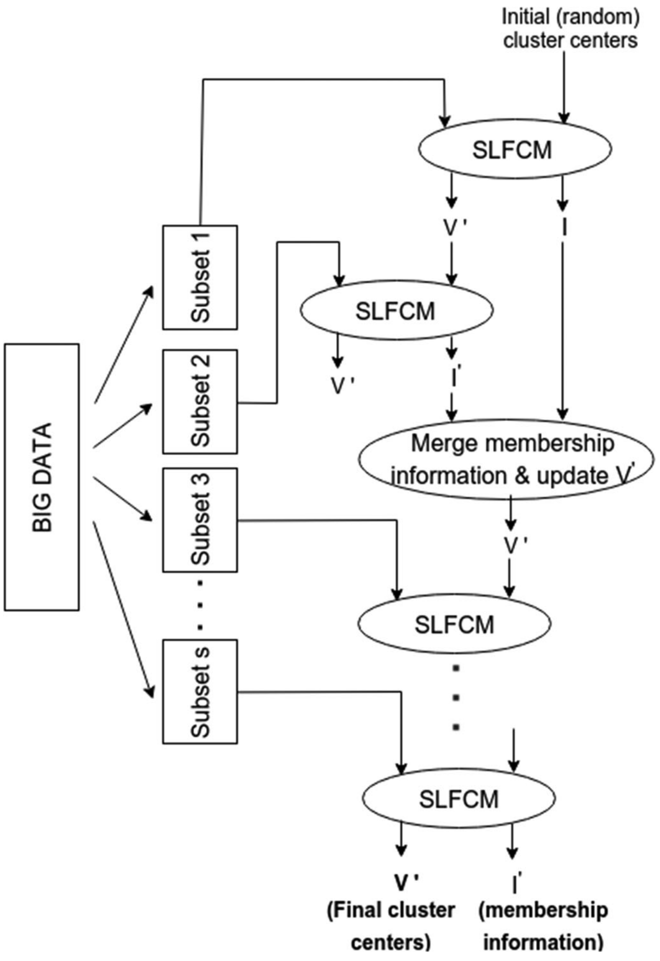


Fig. 6. Workflow of SRSIO-FCM algorithm.

Algorithm 7. *SRSIO-FCM to Iteratively Minimize* $J_m(U, V')$

```
Input : $X, V, c, m$
Output: $V', I'$
1: Load $X$ as $n_s$ sized randomly chosen subsets $X = \{X_1, X_2 \ldots X_s\}$.
2: Sample $X_1$ from $X$ without replacement.
3: $V', I' = SLFCM(X_1, V, c, m)$
4: for $p = 2$ to $s$ do
      4.1: $V', I = SLFCM(X_p, V', c, m)$
      4.2: Merge the partition of all processed subsets
         for $j = 1$ to c do
            $I'_j = < j, < (sum\_d_jx)_{I_j} + (sum\_d_jx)_{I'_j}, (sum\_d_j)_{I_j}, + (sum\_d_j)_{I'_j} >>$
         end for
      4.3: Compute new cluster centers $v'_j$ using $< j, < sum\_d_jx, sum\_d_j >>$ in $I'$ by Eq. (6) $\forall j \in [1, c]$
   end for
5: Compute the objective function using Eq. (1).
6: Return $V', I'$
```

Fig. 6 shows the complete workflow of SRSIO-FCM which makes use of SLFCM for calculating the membership information and cluster centers of all the subsets. It shows that how the data are randomly divided into various subsets and how the initial cluster centers is randomly selected for the clustering of subset1. It also shows that, how the

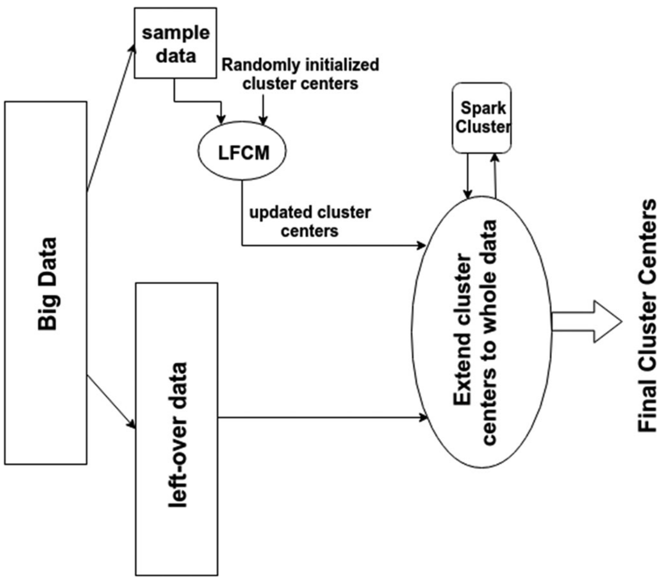


Fig. 7. Workflow of SrseFCM algorithm.

cluster centers and membership information for the subset1 is found using SLFCM. In addition to this, it also shows that how the cluster centers obtained from subset1 are used as the initial cluster centers for the clustering of subset2 using SLFCM. Then, the cluster centers and membership information of the second subset is obtained using SLFCM. Furthermore, the workflow show that how the membership information of subset1 and subset2 are combined to compute the new cluster centers. The new cluster centers is used as initial center for the clustering of subset3. The same procedure is repeated for the clustering of the rest of the subsets.

These cluster centers are then fed as input for the clustering of third subset. This same procedure is repeated for the clustering of all the subsequent subsets. SRSIO-FCM works by dividing the entire data set into various subsets and it continues until the clustering is performed on all the subsets. In this way, it ensures that it covers the complete object space represented by the entire data set. Thus, the results produced by SRSIO-FCM are equivalent to that produced by performing the clustering over the entire data at once.

4.3 Scalable Version of Random Sampling Plus Extension Fuzzy c-Means (SrseFCM)

SrseFCM is a scalable version of rseFCM, which takes a sample from the entire data set and performs LFCM over it. The cluster centers obtained are then extended to the entire partition. This is done in parallel to ensure scalability. The map and reduceByKey functions are applied on the entire data and the final cluster centers are obtained using Eq. (6). To compute the final cluster centers in Eq. (6), we do not store the membership matrix. Instead, we compute the numerator and denominator as sum_d_jx and sum_d_j respectively, while finding the cluster memberships of the data-points. This makes our implementation space efficient. Algorithm 8, presents the step-wise procedure for SrseFCM and Fig. 7 depicts the workflow of SrseFCM.

TABLE 1
Space and Time Complexity of Scalable Fuzzy Algorithms

Algorithm	Time Complexity	Space Complexity
SLFCM	$O(ncdt/w)$	$O(ncd)$
SrseFCM	$O(ncd/w)$	$O(ncd)$
SRSIO-FCM	$O(ncdt/w)$	$O(ncd/s)$

Algorithm 8. *SrseFCM Approximately Minimize* $J_m(U, V')$

```
Input : $X, V, c, m$
Output: $V', I'$
1: Sample the $n_s$ objects from X without replacement,
   denoted $X_s$.
2: $U_s, V = LFCM(X_s, V, c, m)$
3: Compute membership information for entire data.
       $I' = X.map(V).reduceByKey()$
4: Compute cluster centers $v'_j$ using $<j, <sum\_d_jx,$
   $sum\_d_j >>$ in $I'$ by Eq. (6), $\forall j \in [1, c]$
5: Return $V', I'$
```

5 Complexity

We analyze the space and time complexity of SLFCM, SrseFCM and SRSIO-FCM algorithm. All operations and storage space are counted as unit cost. We have not made any assumptions about the cost cutting using special programming tricks or properties of equations involved. For example, we do not assume space that might be saved by over-writing arrays or reusing variables. The space complexity is with respect to the amount of data held in RAM during computation. However, the asymptotic estimates that are shown in Table 1 for the growth in time and space where, n is the number of data-points in X, which are unaffected by the changes in counting procedures.

Table 1 shows the complexities of the scalable fuzzy clustering algorithms in terms of problem variables: X is a set that consists of n number of data-points in the d-dimensional space, $X \in R^d$; c is the number of clusters; w is the number of worker nodes in Spark Cluster; and X is partitioned, by random sampling without replacement, into s number of partitions represented as $X = \{X_1, X_2, ..X_s\}$. Each partition has (n/s) data-points. The number of iterations required for termination is taken as t. It may differ, but for the sake of simplicity and comparison, t is taken as the maximum of the number of iterations for one run of SLFCM, SrseFCM and SRSIO-FCM.

The complexity analysis is done as follows. First, we compute the cost of membership degree calculation during the map phase. Each map operation computes the membership degree of one data point with respect to c cluster centers. Since each data-point has d-dimensions, calculation of each membership degree takes $O(d)$ time and $O(d)$ space. Therefore, each map operation takes $O(cd)$ time and $O(cd)$ space. The reduceByKey operation linearly adds the values of all the map outputs corresponding to one cluster on each worker node and combines the resulting values on the master node. Assuming equal distribution of work to the worker nodes, each node takes $O(n'd)$ time and $O(cd)$ space, where n' is the number of data-points fed as input to one

TABLE 2
Description of the Data sets

Parameters	Data sets				
	MNIST8m	Replicated USPS	Monarch Skin	SUSY	Replicated dim32
Number of instances	8,100,000	14,582,000	1,470,342,000	5,000,000	104,448,000
Number of features	784	256	3	18	32
Number of classes	10	10	2	2	16
Data set Size	19 GB	34 GB	25.4 GB	2.4 GB	11.5 GB

worker node. Combining the results of reduce operations takes $O(cwd)$ time and $O(cd)$ space.

SLFCM performs map and reduceByKey operations on the entire data set. Each worker node process n/w data. All the worker nodes process data in parallel. Thus, the map phase takes a total of $O(ncd/w)$ time and $O(ncd)$ space for each iteration across all worker nodes. Assuming that SLFCM runs for t-iterations, the total time taken for map phase is $O(ncdt/w)$. Since, map outputs have been held in memory only for the duration of one iteration of SLFCM, thus the total space complexity for the map phase of SLFCM is $O(ncd)$. The reduceByKey function linearly adds n/w map outputs, corresponding to each cluster center in parallel, on each worker node. This takes $O(nd/w)$ time and $O(cd)$ space on a worker node. Since, each worker node performs tasks on n/w data in the same amount of time and in parallel, thus the total time required is $O(nd/w)$. Each worker node gives c outputs, each of dimension d, which are aggregated on the master node and added. This takes $O(cdw)$ time and $O(cdw)$ space. Thus, the total time complexity for reduceByKey phase is $O(ncd)$ and space complexity is $O(cdw)$. Therefore, the time complexity of SLFCM is $O(ncdt/w)$ and space complexity is $O(ncd)$, where $n \gg w, c$.

SrseFCM takes a very small subset of the entire data, denoted as $n^{'}$, by randomly selecting data points without replacement from the entire data and performs LFCM over it. So, its time complexity would be the same as that of LFCM for $(n^{'})$ data where $n^{'} \ll n$. But, while extending the cluster centers to the entire data, membership degree of all the points are calculated. The time complexity for this final step is $O(ncd/w)$ which is much higher than the time complexity of running LFCM over $(n^{'})$ data. Similarly, the space complexity of running LFCM over $(n^{'})$ data is very less as compared to the space complexity of the final extension step. In the final extension step, we compute the membership degree of n-sized data consist of d-dimensions for c different cluster centers. This results in a space complexity of $O(ncd)$. Thus, the time complexity is $O(ncd/w)$ and space complexity is $O(ncd)$ respectively.

SRSIO-FCM divides the entire data X into s equal subsets such that $X = \{X_1, X_2, \ldots, X_s\}$, where each subset is of size (n/s). It performs SLFCM over each of these subsets serially. The time and space complexity of performing SLFCM over each subset are $O(ncdt/sw)$ and $O(ncd/s)$ respectively. Since, all the subsets are processed one after another, the resulting time complexity is $O(ncdt/w)$ while the space complexity remains $O(ncd/s)$ because data corresponding to one subset is not retained in the memory while processing the next subset.

Table 1 shows that SLFCM and SRSIO-FCM share the same time complexity. This may lead one to think that both have the same run-time. However, this is not the case. Since, we divide the entire data into various subsets and perform clustering over each subset in SRSIO-FCM. So, clustering performed by SRSIO-FCM on each subset converges by taking the less number of iterations (t) for each subset. So, SRSIO-FCM has lesser run-time, since it performs clustering on a lesser amount of data as compared to SLFCM which performs clustering of the entire data. As we will see in Section 6.5, there is a significant difference in the run-time of the two algorithms. SrseFCM algorithm uses the cluster centers that are returned by LFCM on a subset of data to produce full data partitions. Although, we have not estimated complexity for the completion step, which is not used in our experiments.

The space complexity of SRSIO-FCM is less when compared with SrseFCM and SLFCM, and is proportional to the number of data-points that are present in each subset, i.e., n/s. The space complexity of SRSIO-FCM will be same as SrseFCM if the size of the subset in SRSIO-FCM is taken to be equal to the size of the subset in SrseFCM.

6 Experimental Results

In the experiments, we compare the performance of SRSIO-FCM with SLFCM and SrseFCM using various measures [37], [38], [39] etc. All the approaches are implemented on an Apache Spark Cluster.

6.1 Experimental Environment

The experiments were run on a Spark cluster with three nodes. Each node has the following configuration: Intel Xenon(r) CPU ES-1607 v3 @ 3.10 GHz x 4, 32 GB RAM and 2TB storage. The algorithms were implemented in Python and tested over Hadoop version 2.4 with Apache Spark version 1.4.0. We used HDFS for storing data across the cluster and YARN [26] for resource management.

6.2 Data Sets

We compare the performance of SRSIO-FCM, SLFCM and SrseFCM algorithm on following data sets. Table 2 presents the detail of the data sets used for the experimental study and description of these data sets is presented as follows:

1) Minst8m Dataset: It is a handwritten-digit recognition data set containing greyscale images (28 $\times$ 28 pixels). Each feature vector represents the intensity of the black color of a particular pixel. The data set contains 8,100,000 instances divided into 10 classes, each having 784 features.

TABLE 3
Parameters Specification for Various Data sets

Parameters	Data sets				
	MNIST8m USPS	Replicated Skin	Monarch	SUSY dim32	Replicated
ϵ	0.001	0.001	0.001	0.001	0.001
m	1.75	1.75	1.75	1.75	1.75
c	10	10	2	3	16

2) Replicated-USPS Dataset: It is numeric data obtained from the scanning of handwritten digits from envelopes by the U.S. Postal Service. The original scanned digits are binary and of different sizes and orientations; the images here have been deslanted and size normalized, resulting in 16×16 grayscale images. The original data set contains 7,291 instances and 2,007 test samples, each having 256 features and consists of 10 classes [33]. To test our algorithm on really huge datasets, we generated the Replicated-USPS data set by repeating the original USPS data set 2,000 times to make it larger. The resulting size of the data set is 34 GB. Thus, our final data set contains 145,82,000 instances.
3) Monarch-Skin Dataset: To test our algorithm on really huge data set, we created the Cartesian product of Skin segmentation data obtained from UCI [34] with monarch [35] data set. Monarch data have only one feature while Skin segmentation data consist of 245,057 instances having three features and consists of two classes. We used a 25.4 GB sized subset of the Cartesian product with 1,470,342,000 instances for our experiments. We referred it as Monarch-Skin data set.
4) SUSY Dataset: It has been produced using Monte Carlo simulations. It consists of two classes and has 18 features. The first eight features are kinematic properties measured by the particle detectors in the accelerator. The last ten features are functions of the first eight features; these are high-level features derived by physicists to help discriminate between the two classes. The size of this data set is 2.4 GB and contains 5,000,000 instances.
5) Replicated-dim32 Dataset: The Dim3 data set is a synthetic data with Gaussian clusters composed of 2026 3-D vectors, which is visually grouping into nine clusters. In order to get a huge data set, we replicated it 1,000 times. The resulting data set is 11.5 GB in size and contains 104,448,000 instances, referred as Replicated-dim32 data set.

6.3 Parameters Specification

We fix the value of fuzzification parameter, i.e., $m = 1.75$ and termination criteria $\epsilon = 0.001$ for all the datasets used in the experimental study. However, these values are proven to work well for most datasets [36]. Table 3 present the specification of parameters value used in experimental evaluation.

6.4 Evaluation Criteria

6.4.1 Normalized Mutual Information (NMI)

NMI [37] is used to compute the clustering results, which measure the agreement of the clustering results produced by an algorithm and the ground truth. If we refer to class as the ground truth and to cluster as the results of a clustering algorithm, the NMI is calculated as follows:

$$NMI = \frac{\sum_{c=1}^{k} \sum_{p=1}^{m} n_c^p \log\left(\frac{n.n_c^p}{n_c.n_p}\right)}{\sqrt{\left(\sum_{c=1}^{k} n_c \log\left(\frac{n_c}{n}\right)\right)\left(\sum_{p=1}^{m} n_p \log\left(\frac{n_p}{n}\right)\right)}}. \quad (7)$$

Where, n is the total number of data points, n_c and n_p are the numbers of data-points in the cth cluster and the pth class, respectively, and n_c^p is the number of common data-points in class p and cluster c.

6.4.2 F-Measure

F-measure [38] helps in determining the accuracy of a clustering solution. The F-measure for a cluster C'_j with respect to a certain class C_i indicates how good cluster C'_j describes class C_i by calculating the harmonic mean of precision and recall as follows:

$$p_{ij} = \frac{n_{ij}}{n_j}, r_{ij} = \frac{n_{ij}}{n_i} \quad (8)$$

$$f(C_i, C'_j) = \frac{2 * p_{ij} * r_{ij}}{p_{ij} + r_{ij}}. \quad (9)$$

Where, n_{ij} denotes number of samples in cluster C_i that are also present in class C_j, n_i denotes the number of samples belonging to cluster C_i and n_j denote the number of sample belongs to class C_j respectively, precision p_{ij} is the fraction of n_{ij} and n_j, recall r_{ij} is the fraction of n_{ij} and n_i. The overall f-measure is then defined as the weighted sum of the maximum f-measures for the clusters in C_j:

$$f\text{-}measure = \sum_i \frac{n_{ij}}{n} max_j f(C_i, C'_j). \quad (10)$$

Where, n is the total number of data points. The higher value of F-measure indicate better clustering results. F-measure equal to 1 indicates that the clustering result is same as the ground truth.

6.4.3 Adjusted Rand Index (ARI)

The adjusted Rand index [39] is the corrected-for-chance version of the Rand index, which is a measure of the similarity between two data clustering. To compute the ARI, we first harden the fuzzy partitions by setting the maximum element in each column of U to 1, and all else to 0. We use ARI to compare the clustering solutions with ground-truth labels (when available), as well as to compare other algorithms with the SRSIO -FCM solutions. The formulation of ARI is defined as follows:

$$ARI = \frac{\sum_{i,j}\binom{n_{ij}}{2} - \left[\sum_i \binom{n_{i.}}{2}\sum_j \binom{n_{.j}}{2}\right]/\binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{n_{i.}}{2} + \sum_j \binom{n_{.j}}{2}\right] - \left[\sum_i \binom{n_{i.}}{2}\sum_j \binom{n_{.j}}{2}\right]/\binom{n}{2}}. \quad (11)$$

6.4.4 Objective Function Value

The performance of SRSIO-FCM is evaluated on the basis of its ability to minimize the objective function stated in Eq. (1).

TABLE 4
NMI for SLFCM, SrseFCM and SRSIO-FCM with Various Chunk Sizes on Different Data Sets

Algorithm	Chunk size	Data sets				
		MNIST8m	Replicated USPS	Monarch Skin	SUSY	Replicated dim32
SRSIO-FCM	1%	0.117094	0.365415	0.012703	0.028723	0.967859
	2.5%	0.125052	0.365966	0.012703	0.029032	1.000000
	5%	0.122007	0.365119	0.012703	0.028674	1.000000
	10%	0.118471	0.365557	0.012705	0.028667	1.000000
	20%	0.126027	0.365292	0.012703	0.028798	1.000000
SLFCM	100%	0.122540	0.396155	0.012703	0.025895	1.000000
SrseFCM	0.001%	0.118607	0.253252	0.013281	0.001194	0.883591

TABLE 5
F-Measure for SLFCM, SrseFCM and SRSIO-FCM with Various Chunk Sizes on Different Data Sets

Algorithm	Chunk size	Data sets				
		MNIST8m	Replicated USPS	Monarch Skin	SUSY	Replicated dim32
SRSIO-FCM	1%	0.362525	0.787940	0.648496	0.641601	1.000000
	2.5%	0.365356	0.788448	0.648497	0.642083	1.000000
	5%	0.363194	0.788653	0.648496	0.640909	1.000000
	10%	0.364882	0.788539	0.648508	0.641277	1.000000
	20%	0.379519	0.788608	0.648497	0.641660	1.000000
SLFCM	100%	0.381972	0.842952	0.648497	0.607977	1.000000
SrseFCM	0.001%	0.380723	0.565469	0.619401	0.453477	0.947917

TABLE 6
ARI for SLFCM, SrseFCM and SRSIO-FCM with Various Chunk Sizes on Different Data Sets

Algorithm	Chunk size	Data sets				
		MNIST8m	Replicated USPS	Monarch Skin	SUSY	Replicated dim32
SRSIO-FCM	1%	0.048867	0.215461	0.035881	0.042860	0.915891
	2.5%	0.052707	0.215343	0.035882	0.043153	1.000000
	5%	0.051061	0.215777	0.035881	0.042747	1.000000
	10%	0.049511	0.215533	0.035889	0.043010	1.000000
	20%	0.053534	0.215607	0.035883	0.042879	1.000000
SLFCM	100%	0.052189	0.268365	0.035881	0.038094	1.000000
SrseFCM	0.001%	0.049464	0.294295	0.022201	0.0314584	0.679988

TABLE 7
Objective Function Value for SLFCM, SrseFCM and SRSIO-FCM with Various Chunk Sizes on Different Data Sets

Algorithm	Chunk size	Data sets				
		MNIST8m	Replicated USPS	Monarch Skin	SUSY	Replicated dim32
SRSIO-FCM	1%	74556041.35	83800629	9.53E+016	24522339.61	1.61E+011
	2.5%	74556041.52	83800623	9.53E+016	24522336.73	2.36E+010
	5%	74556038.97	83800617	9.53E+016	24522337.77	2.36E+010
	10%	74556042.40	83800622	9.53E+016	24522346.13	2.36E+010
	20%	74556036.17	83800622	9.53E+016	24522338.22	2.36E+010
SLFCM	100%	74556031.67	83800498	9.53E+016	24522337.10	2.36E+010
SrseFCM	0.001%	74647034.95	173762000	1.52e+17	24556967.02	1.21E+010

6.4.5 Run-Time

This criterion represents the actual run-time of the algorithm. We do not include the time required for subset creation which is only 0.2 percent of the time required for execution of the algorithm.

We aim to achieve higher NMI, F-measure and ARI but lower objective function value for SRSIO-FCM as compared to SrseFCM and lower run-time as compared to SLFCM.

6.5 Results and Discussion

This section discusses the results of experiments conducted in order to demonstrate the effectiveness of SRSIO-FCM in comparison with SLFCM and SrseFCM on five data sets in terms of various measures such as the NMI, F-measure, ARI, Objective function value and run time.

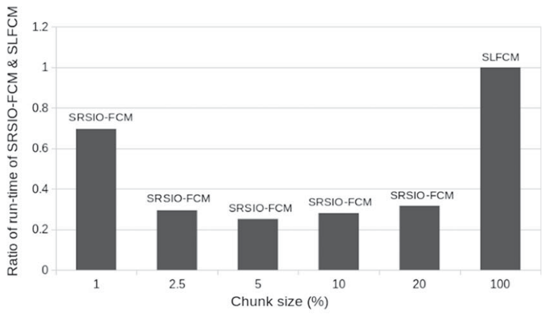
For each data set, we compare the performance of SRSIO-FCM with SLFCM and SrseFCM using the parameters as described in Table 3. Tables 4, 5, 6, and 7 compares the value of NMI, F-measure, ARI and Objective function for SLFCM, SrseFCM and various chunk sizes of SRSIO-FCM on five data sets. Fig. 8 displays the comparison of run-time of SLFCM with various chunk sizes of SRSIO-FCM.

Table 4 tabulates the value of NMI computed for the five data sets using SLFCM, SrseFCM and various chunk sizes of SRSIO-FCM. NMI indicates the quality of clustering. Thus, a better clustering would result in higher NMI. The table shows that the value of NMI for SLFCM and SRSIO-FCM are very close. While the NMI of SrseFCM is much lower as compared to that of SLFCM and SRSIO-FCM, in most cases. In addition to this, there is no significant variation in the value of NMI for various chunk sizes of SRSIO-FCMin case of some data sets. Thus, we can conclude that SRSIO-FCM with the given chunk sizes performs better than SrseFCM and equally well as SLFCM in terms of NMI.

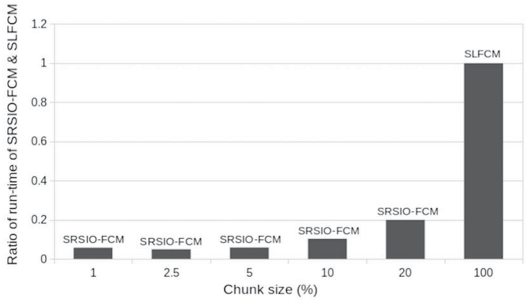
Table 5 tabulates the value of F-measure computed for the five data sets using SLFCM, SrseFCM and SRSIO-FCM with various chunk sizes. F-measure indicates the quality of clustering. Thus, higher value of F-measure indicates better clustering. The table indicates that the values of F-measure for SLFCM and SRSIO-FCM is very close. While the F-measure is significantly lower for SrseFCM when compared to SRSIO-FCM in case of most of the data sets. Also the values of F-measure for various chunk sizes of SRSIO-FCM is approximately equal. Thus, we can conclude that SRSIO-FCM performs as good as SLFCM and better than SrseFCM in terms of F-measure.

Table 6 tabulates the value of ARI computed for the five data sets using SLFCM, SrseFCM and SRSIO-FCM with various chunk sizes. The ARI value compares the clustering result with the ground truth labels. Higher value of ARI corresponds to better clustering. The table indicates that the values of ARI for SLFCM and SRSIO-FCM is very close and very low for SrseFCM. Also the values of ARI for various chunk sizes of SRSIO-FCM are approximately equal. Thus, we conclude that SRSIO-FCM performs as good as SLFCM and better than SrseFCM in terms of ARI.

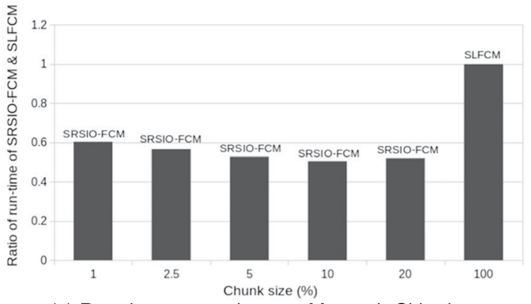
Table 7 displays the value of Objective function computed for the five data sets using SLFCM, SrseFCM and SRSIO-FCM for chunk sizes of 1, 2.5, 5, 10 and 20 percent. The Objective function values for SLFCM and SRSIO-FCM



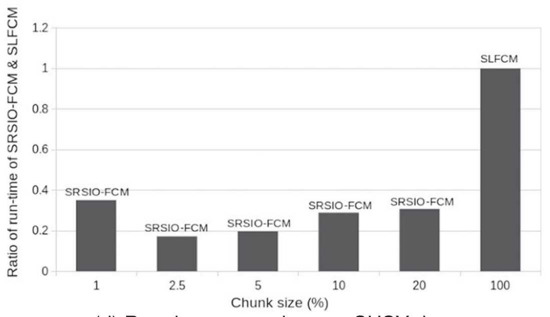
(a) Run-time comparison on MNIST8m data



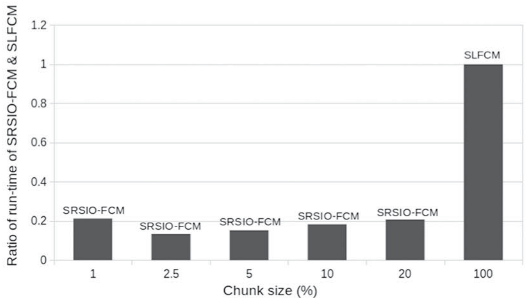
(b) Run-time comparison on Replicated-USPS data



(c) Run-time comparison on Monarch-Skin data



(d) Run-time comparison on SUSY data



(e) Run-time comparison on Replicated-dim32 data

Fig. 8. Comparison of run-time of SRSIO-FCM and SLFCM with varying chunk size on different data sets.

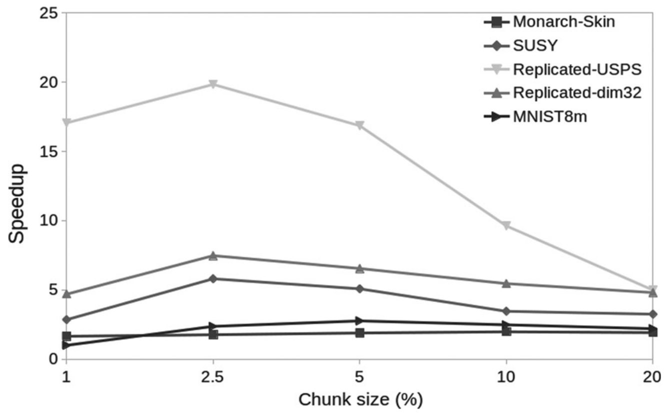


Fig. 9. Speedup of SRSIO-FCM with various chunk size on different data sets.

is found to be very close. While the Objective function values for SrseFCM is much higher than both SLFCM and SRSIO-FCM. Moreover, the values of an Objective function for various chunk sizes of SRSIO-FCM is approximately equal. Thus, we may conclude that SRSIO-FCM minimizes the Objective function equally well as SLFCM and better than SrseFCM in almost all the cases.

Fig. 8 shows the plot of the ratio of time taken for clustering using SLFCM and SRSIO-FCM for chunk sizes of 1, 2.5, 5, 10 and 20 percent on five different data sets. The plots indicate that SRSIO-FCM achieve significantly lower runtime as compared to SLFCM. SRSIO-FCM was found to be up-to 20 times faster than SLFCM in certain cases. However, when the chunk-size is too small then the run-time of SRSIO-FCM starts increasing. This can be attributed to the fact that the overhead associated with communication cost increases significantly when a large number of jobs are performed on a very small subset of data.

Fig. 9 shows the Speedup of SRSIO-FCM with the various chunk size of different data sets. Depending on the size of the data sets, the optimal chunk size may vary for different data sets. Speedup is defined as the ratio of time taken by SRSIO-FCM and that by SLFCM, i.e., $t_{SRSIO\text{-}FCM}/t_{SLFCM}$. However, the speedup would also depend on the number of iterations for which SRSIO-FCM runs on each chunk of data.

7 Conclusion and Future Work

In this paper, we proposed a Scalable Random Sampling with Iterative Optimization Fuzzy c-Means algorithm known as SRSIO-FCM. The proposed approach is implemented on Apache Spark to deal with the challenges associated with fuzzy clustering for handling Big Data. The SRSIO-FCM partitioned the big data into various chunks and process the data points present within the chunk in a parallel manner. One distinctive characteristic of SRSIO-FCM is that due to the parallel processing of a data points within the chunk, it achieves a significant reduction in runtime for the clustering of such huge amounts of data, without compromising the quality of clustering results. The other important characteristic is that, during the execution of the proposed algorithm, we eliminate the need of storing the membership matrix, which makes the execution of the proposed algorithm faster by reducing the run-time. This is a good optimization strategy for clustering of Big Data since, the membership matrix will be too huge to be stored. Empirical evaluations on the several large datasets demonstrated that SRSIO-FCM significantly outperformed over the scalable model of previously proposed iterative algorithms. Moreover, SRSIO-FCM achieves higher or comparable clustering results compared with the scalable iterative algorithms. The merits shown in the experiments indicate that SRSIO-FCM has great potential for use in big data clustering.

This works suggest some interesting directions for future work. As this work presented the clustering of Big Data. So, it is interesting to conduct the investigation in finding the number of clusters for specific data set A very important question that arises is the validity of the clustering. To measure the quality of clustering, cluster validity index for Big Data are needed. Many cluster validity measures for small sized data require full access to the data-points. Hence, we aim to extend some well-known cluster validity measures for use on Big Data by using similar extensions as presented here.

References

[1] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, "Big data: The next frontier for innovation, competition, and productivity," McKinsey Global Inst., Tech. Rep. 9341321, pp. 1–137, May 2011.

[2] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta, "A survey of kernel and spectral methods for clustering," *Pattern Recognit.*, vol. 41, no. 1, pp. 176–190, 2008.

[3] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognit. Lett.*, vol. 31, no. 8, pp. 651–666, 2010.

[4] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surveys*, vol. 31, no. 3, pp. 264–323, 1999.

[5] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, vol. 3. Hoboken, NJ, USA: Wiley, 1973.

[6] M. Steinbach, G. Karypis, and V. Kumar, "A comparison of document clustering techniques," in *Proc. KDD Workshop Text Mining*, 2000, vol. 400, no. 1, pp. 525–526.

[7] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. Berlin, Germany: Springer, 2013.

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

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

[10] P. Hore, L. O. Hall, D. B. Goldgof, Y. Gu, A. A. Maudsley, and A. Darkazanli, "A scalable framework for segmenting magnetic resonance images," *J. Signal Process. Systems*, vol. 54, no. 1–3, pp. 183–203, 2009.

[11] N. Labroche, "New incremental fuzzy c medoids clustering algorithms," in *Proc. Annu. Meeting North Amer. Fuzzy Inf. Process. Soc.*, 2010, pp. 1–6.

[12] R. Krishnapuram, A. Joshi, O. Nasraoui, and L. Yi, "Low-complexity fuzzy relational clustering algorithms for web mining," *IEEE Trans. Fuzzy Syst.*, vol. 9, no. 4, pp. 595–607, Aug. 2001.

[13] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, vol. 344. Hoboken, NJ, USA: Wiley, 2009.

[14] S. Har-Peled and S. Mazumdar, "On coresets for k-means and k-median clustering," in *Proc. 36th Annu. ACM Symp. Theory Comput.*, 2004, pp. 291–300.

[15] S. Guha, R. Rastogi, and K. Shim, "Cure: An efficient clustering algorithm for large databases," in *Proc. ACM SIGMOD Record*, 1998, vol. 27, no. 2, pp. 73–84.

[16] N. Bharill and A. Tiwari, "Handling big data with fuzzy based classification approach," in *Advance Trends in Soft Computing*. Berlin, Germany: Springer, 2014, pp. 219–227.

[17] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[18] P. Mika, "Flink: Semantic Web technology for the extraction and analysis of social networks," *Web Semantics: Sci. Services Agents World Wide Web*, vol. 3, no. 2, pp. 211–223, 2005.

[19] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "Haloop: Efficient iterative data processing on large clusters," *Proc. VLDB Endowment*, vol. 3, no. 1/2, pp. 285–296, 2010.

[20] A. M. Team, "Apache Mahout: Scalable machine-learning and data-mining library," (2011). [Online]. Available: http://mahout.apache.org/

[21] Y. Zhang, S. Chen, Q. Wang, and G Yu, "i2MapReduce: Incremental MapReduce for mining evolving big data," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 7, pp. 1906–1919, Jul. 2015.

[22] M. Zaharia, et al., "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation*, 2012, p. 2.

[23] J. C. Bezdek and R. J. Hathaway, "Convergence of alternating optimization," *Neural Parallel Sci. Comput.*, vol. 11, no. 4, pp. 351–368, 2003.

[24] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput.*, 2010, vol. 10, Art. no. 10.

[25] D. Borthakur, "HDFS architecture guide," *Hadoop Apache Project*, 2008. [Online]. Available: http://hadoop.apache.org/common/docs/current/hdfs design. pdf

[26] V. K. Vavilapalli, et al., "Apache Hadoop yarn: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput.*, 2013, Art. no. 5.

[27] B. Hindman, et al., "Mesos: A platform for fine-grained resource sharing in the data center," in *Proc. 8th USENIX Conf. Netw. Syst. Des. Implementation*, 2011, vol. 11, pp. 22–22.

[28] T. White, *Hadoop: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly Media, 2012.

[29] L. George, *HBase: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly Media, 2011.

[30] A. Thusoo, et al., "Hive: A warehousing solution over a map-reduce framework," *Proc. VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[31] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Tachyon: Reliable, memory speed storage for cluster computing frameworks," in *Proc. ACM Symp. Cloud Comput.*, 2014, pp. 1–15.

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

[33] J. J. Hull, "A database for handwritten text recognition research," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 5, pp. 550–554, May 1994.

[34] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[35] G. Frahling and C. Sohler, "Coresets in dynamic geometric data streams," in *Proc. 27th Annu. ACM Symp. Theory Comput.*, 2005, pp. 209–217.

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

[37] A. Strehl and J. Ghosh, "Cluster ensembles—a knowledge reuse framework for combining multiple partitions," *J. Mach. Learn. Res.*, vol. 3, pp. 583–617, 2003.

[38] Y. Wang, L. Chen, and J.-P. Mei, "Incremental fuzzy clustering with multiple medoids for large data," *IEEE Trans. Fuzzy Syst.*, vol. 22, no. 6, pp. 1557–1568, Dec. 2014.

[39] K. Y. Yeung and W. L. Ruzzo, "Details of the adjusted rand index and clustering algorithms, supplement to the paper an empirical study on principal component analysis for clustering gene expression data," *Bioinf.*, vol. 17, no. 9, pp. 763–774, 2001.

Neha Bharill received the BE degree in Department of Information Technology from Bansal Institute of Science and Technology, Bhopal, India, in 2008 and ME degree in Department of Computer Science and Engineering from Shri Govindaram Sakseria Institute of Technology and Science, Indore, India, in 2011. She is currently pursuing the PhD degree under supervision of Dr. Aruna Tiwari in Department of Computer Science and Engineering from Indian Institute of Technology Indore, India. Her current research interests include fuzzy sets and systems, Big Data, pattern recognition, data mining and machine learning. She is reviewer of *IEEE Transaction on Cybernetics*, *Swarm and Evolutionary Computation of Elsevier* and *Complex & Intelligent Systems of Springer*. She is a member of IEEE and IEEE Computational Intelligence Society.

Aruna Tiwari received the BE degree (Computer Engineering) in 1994 and ME degree (Computer Engineering) in 2000 from Shri Govindaram Sakseria Institute of Technology and Science, Indore and PhD degree from Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal, India. She joined the Indian Institute of Technology Indore, India, in 2012, where she is currently working as an Assistant Professor with the Department of Computer Science and Engineering. Her research interests include soft computing techniques with neural network learning algorithms, evolutionary approaches, fuzzy based approaches for handling Big Data and nonstationary data. She has many publications in peer reviewed journals, International conferences, and Book chapters. She is reviewer of many journals some of them are *IEEE Transaction on TKDE*, *Neurocomputing journal of Elsevier* etc. She is a member of IEEE Computational Intelligence Society and life member of Computer Society of India.

Aayushi Malviya received the BTech degree in Computer Science and Engineering from Indian Institute of Technology Indore, India, in 2016. She recently joined the Microsoft India Development Center, India, in 2016, where she is working as a Software Engineer. Her research interests include fuzzy sets and systems, Big Data, data mining and machine learning. She is a member of IEEE.