Design of Scalable Fuzzy Clustering Algorithms and its Application to Huge Genomics Data

Ph.D. Thesis

By

Preeti Jha

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

August 2021

Design of Scalable Fuzzy Clustering Algorithms and its Application to Huge Genomics Data

A THESIS

submitted to the

INDIAN INSTITUTE OF TECHNOLOGY INDORE

in partial fulfillment of the requirements for

the award of the degree

of

DOCTOR OF PHILOSOPHY

By

Preeti Jha

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

August 2021

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Design of Scalable Fuzzy Clustering Algorithms and its Application to Huge Genomics Data** in the partial fulfillment of the requirements for the award of the degree of **Doctor of Philosophy** and submitted in the **Department of Computer Science and Engineering, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the time period from December 2018 to June 2021 under the supervision of Dr. Aruna Tiwari, Associate Professor, Indian Institute of Technology Indore, India, Dr. Milind B. Ratnaparkhe, Senior Scientist (Biotechnology), ICAR-Indian Institute of Soybean Research (ICAR-IISR), Indore, India, and Dr. Neha Bharill, Assistant Professor, Department of Computer Science and Engineering, Mahindra University, Ecole Centrale School of Engineering (MEC), Hyderabad, India.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Preeti Jha 23.12.2021

Signature of the Student with Date

(Preeti Jha)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Signature of Thesis Supervisor with Date

(Dr. Aruna Tiwari)

Signature of Thesis Supervisor with Date

(Dr. Milind Ratnaparkhe)

Signature of Thesis Supervisor with Date

(Dr. Neha Bharill)

Preeti Jha has successfully given her Ph.D. Oral Examination held on 23-12-2021

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my heartfelt gratitude to a number of persons who in one or the other way contributed by making this time as learnable, enjoyable, and bearable. At first, I would like to thank my supervisors **Dr. Aruna Tiwari**, **Dr. Milind Ratnaparkhe**, and **Dr. Neha Bharill** who was a constant source of inspiration during my work. Without their constant guidance and research directions, this research work could not be completed. Their continuous support and encouragement has motivated me to remain streamlined in my research work.

I am thankful to **Dr. Kapil Ahuja** for all his extended help, support, and constructive feedback. I am thankful to **Dr. Abhishek Srivastava** and **Dr. Trapti Jain**, my research committee member for taking out some valuable time to evaluate my progress all these years. Their good comments and suggestions helped me to improve my work at various stages. I am also grateful to **Dr. Somnath Dey**, HOD of Computer Science and Engineering for his help and support.

My sincere acknowledgement and respect to **Prof. Neelesh Kumar Jain** , Director, Indian Institute of Technology Indore for providing me the opportunity to explore my research capabilities at Indian Institute of Technology Indore.

I extend my sincere thanks to the **Council of Scientific and Industrial Research (CSIR)** for funding the PhD research. This work was supported by the CSIR under Grant 22(0750)/17/EMR-II in collaboration with **ICAR-Indian Institute Of Soybean Research, Indore**.

I would like to appreciate the fine company of my dearest colleagues and friends especially, Chandan Gautam, Vikas Chauhan, Suchitra Agrawal, and Sonal Pandey. I am thankful to undergraduate students (Mounika Mukkamalla and Neha Nagendra) who have also supported me in my research work. I am also grateful to the institute staffs for their unfailing support and assistance.

I would like to express my heartfelt respect to my parents for their love, care and support they have provided to me throughout my life. Special thanks to my husband (Bhavesh), my sister (Priyanka), my brother (Piyush), and friends as this thesis would

not have been possible without the help of their support and encouragements. I also want to thank my in-laws for their support and blessings.

Finally, I am thankful to all who directly or indirectly contributed, helped and supported me. To sign off, I write a quote by Albert Einstein:

"In the middle of difficulty lies opportunity." –Albert Einstein

Preeti Jha

To my family and friends

Abstract

Clustering is one of the most popular methods used for exploratory data analysis. The need for clustering arises in many real-life problems, such as gene analysis, image processing, text organization, community detection, disease diagnosis, and protein categorization. In the bioinformatics domain, an enormous amount of new genome sequences are produced at a great pace. Hence, the clustering of genome sequencing gives rise to this new era of Big Data in bioinformatics. Clustering genome sequences in real life becomes a major challenge because sequences can belong to multiple clusters. So, there is a need to apply a clustering algorithm that assigns a data sample to more than one cluster. Fuzzy clustering is one of the most widely used methods to handle such real-life problems. The principle advantage of fuzzy clustering is that the membership degrees express how ambiguously a data sample should belong to a cluster. However, there are many aspects of the design of fuzzy clustering that need to be addressed for improving the overall performance of fuzzy clustering by preserving the quality of clustering to handle Big Data.

This thesis mainly investigates to design and develop the fuzzy based scalable clustering algorithms and feature extraction techniques for handling huge genome data using Apache Spark. To handle Big Data, novel scalable fuzzy clustering approaches are designed. First, we have proposed scalable kernelized fuzzy clustering algorithms for handling Big Data. These scalable kernelized fuzzy clustering algorithms are evolved to deal with the non-linear separable problems by applying a kernel Radial Basis Functions (RBF), which maps the input data space non-linearly into a high dimensional feature space. The proposed scalable kernelized fuzzy clustering algorithms are being implemented on Apache Spark cluster to perform the efficient clustering of Big Data due to its in-memory cluster computing technique. The proposed algorithms remove the problem of loading the entire data in memory all at once. This results in a significant reduction in run-time.

To further improve the cluster quality, we have proposed a novel scalable incremental fuzzy consensus clustering algorithm, which aims to find a single partition of data

that agrees as much as possible with existing basic partitions/segments. The scalable incremental fuzzy consensus clustering aims to identify a soft consensus partition with overlapping clusters from a set of fuzzy partitions. It has been implemented on Apache Spark cluster framework, a distributed data stream environment for handling big data by considering the data as a set of subsets of data that are processed incrementally. The scalable incremental fuzzy consensus clustering facilitates efficient Big Data clustering by improving the quality of clusters and thus performing storage space optimization and significantly reducing time complexity. The scalable kernelized fuzzy clustering and scalable fuzzy consensus clustering is applied to huge genome data. Before clustering raw genome sequences, there is a need to develop a method that can extract significant features from huge genome sequences.

To handle huge genome sequences, we have proposed novel scalable feature extraction techniques for preprocessing huge Single Nucleotide Polymorphism (SNP) and protein sequences that extract fixed-length numerical feature vectors. The extracted numerical feature vectors are then fed as an input to the developed scalable fuzzy clustering algorithms to cluster huge SNP and protein datasets. Finally, we have investigated massive protein data of the Severe Acute Respiratory Syndrome Coronavirus-2 (SARS-CoV-2) using our developed scalable feature extraction approach and scalable fuzzy clustering algorithms. Therefore, the scalable algorithms presented in this thesis are generalized to various genome datasets of any size (Big Data).

List of Publications

A. Published

A1. In Refereed Journals

1. **P. Jha**, A. Tiwari, N. Bharill, M. Ratnaparkhe, M. Mounika, and N. Nagendra. *A Novel Scalable Kernelized Fuzzy Clustering Algorithms Based on In-Memory Computation for Handling Big Data*, IEEE Transactions on Emerging Topics in Computational Intelligence, 2020, pp. 1-12, DOI= 10.1109/TETCI.2020.3016302.

2. **P. Jha**, A. Tiwari, N. Bharill, M. Ratnaparkhe, M. Mounika, and N. Nagendra. *Apache Spark based kernelized fuzzy clustering framework for single nucleotide polymorphism sequence analysis*, Computational Biology and Chemistry, vol. 92, pp. 107454, 2021 (Elsevier), DOI=https://doi.org/10.1016/j.compbiolchem.2021.107454. (**IF:2.877**)

3. **P. Jha**, A. Tiwari, N. Bharill, M. Ratnaparkhe, N. Nagendra, and M. Mounika. *Scalable Incremental Fuzzy Consensus Clustering Algorithm for Handling Big Data* Soft Computing, vol. 25, pp. 8703–8719, 2021 (Springer), DOI= https://doi.org/10.1007/s00500-021-05733-1. (**IF:3.050**)

A2. In Refereed Conferences

1. **P. Jha**, A. Tiwari, N. Bharill, M. Ratnaparkhe, N. Nagendra, and M. Mounika. *Fuzzy-Based Kernelized Clustering Algorithms for Handling Big Data Using Apache Spark*, In International Conference on Harmony Search Algorithm (Springer), Singapore, pp. 423-435, April 2020, DOI=https://doi.org/10.1007/978-981-15-8603-3_37.

B. Communicated

In Refereed Conferences

1. **P. Jha**, A. Tiwari, N. Bharill, M. Ratnaparkhe, O.P. Patel, N. Harshith, S. Solasa. *COVID-19 Protein Sequences study with a Novel Scalable Feature Extraction Approach and their Cluster Analysis with Kernelized Fuzzy Algorithm*, 2022 IEEE International Conference on Big Data and Smart Computing, January, 17-20, 2022. (Accepted in Nov 22, 2021)

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

FCM Fuzzy C-Means

LFCM Literal Fuzzy C-Means

SLFCM Scalable Literal Fuzzy C-Means

KSLFCM Kernelized Scalable Literal Fuzzy C-Means

RSIO-FCM Random Sampling Iterative Optimization Fuzzy C-Means

GB GigaByte

VL Very Large

ICAR Indian Council of Agricultural Research

IISR Indian Institute of Soybean Research

TBs TeraBytes

NMI Normalized Mutual Information

ARI Adjusted Rand Index

SRSIO Scalable Random Sampling with Iterative Optimization

KSRSIO Kernelized Scalable Random Sampling with Iterative Optimization

RAM Random Access Memory

GHZ GigaHertz

rseFCM random sampling plus extension Fuzzy C-Means

CPSF Co-occurrence based Probability Specific Feature

SCPSF Scalable Co-occurrence based Probability Specific Feature

HDFS Hadoop Distributed File System

RDD Resilient Distributed Datasets

RBF Radial Basis Function

DNA Deoxyribonucleic Acid

SNP Single Nucleotide Polymorphism

KFCM Kernel based Fuzzy C-Means

BSs Basic Segments

FKCM Fuzzy Kernel C-Means

COVID-19 Corona Virus Disease-19

SARS Severe Acute Respiratory Syndrome

IDC International Data Corporation

HPC High-Performance Computing

SI Silhouette Index

DBI Davies Bouldin Index

SEC Spectral Ensemble Clustering

KCC K-means Consensus Clustering

PSE Protein Sequence Encoding

GSM Global Similarity Measures

LSM Local Similarity Measures

DAG Directed Acyclic Graph

NCBI National Center for Biotechnology Information

PDB Protein Data Bank

SARS-CoV-2 Severe Acute Respiratory Syndrome Coronavirus-2

Chapter 1

Introduction

Clustering is a form of exploratory data analysis in which data are separated into groups or subsets such that the objects in each group share some similarity. Any field that uses or analyzes data can utilize clustering; the problem domains and clustering applications are innumerable. Clustering is used in various applications such as gene analysis, image processing, community detection, scientific data exploration, information retrieval, text mining, Web analysis, marketing, medical diagnostics, and many others [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. Clustering algorithms are mainly divided into hierarchical clustering and partitioning clustering [11, 12]. The hierarchical clustering algorithm creates a complete hierarchical structure by dividing the dataset into several levels of nested parts (clusters) represented by a dendrogram (tree). At the same time, the partitioning algorithm divides the dataset into a given number of groups or clusters, usually by optimizing the objective function. Partition based clustering algorithms are mainly divided into hard (or crisp) and fuzzy clustering [13]. The hard clustering algorithm divides data samples into multiple disjoint groups or clusters, and each data sample belongs to only one group. The main disadvantage of hard clustering is that it becomes unsuitable for real datasets with no defined boundaries between groups. Therefore, people realized the need to deal with this situation, which led to emerging algorithms based on the fuzzy set theory concept. According to Lotfi Zadeh [14], the fuzzy theory deals with the problem of uncertain boundaries between clusters. Therefore, a fuzzy clustering algorithm is evolved. Fuzzy clustering algorithm provide

Table 1.1: Huber's description of dataset sizes

Bytes	10^2	10^4	10^6	10^8	10^{10}	10^{12}	$10^{>12}$
Sizes	tiny	small	medium	large	huge	monster	VL

fuzzy partitions to measure the uncertainty of data samples belonging to two or more clusters with a varying degree of membership [15]. The main advantage of fuzzy clustering is that the degree of membership indicates the degree of belongingness with which the data sample can belong to multiple clusters. In addition, these membership degrees allow to handle ambiguous data properly [15]. Therefore, fuzzy clustering can deal with uncertainty and help to solve many complex problems in real-life [16].

Due to the increasing amount and ubiquity of Big Data in various fields such as banking, agriculture, chemistry, data mining, cloud computing, finance, marketing, stocks, and heath care, etc., many clustering algorithms are designed to handle such massive datasets [17, 18, 19, 20]. Also, bioinformatics has produced an abundance of exceedingly large genome datasets [21]. According to Huber statistics [22, 23], the dataset is classified into different categories based on distinctive sizes, as given in Table 1.1. Bezdek and Hathway [24] added the category of Very Large (VL) information to this table. Different types of fuzzy-based clustering algorithms have been proposed to cluster VL information [25, 26, 27, 28, 29, 30, 31, 32]. Literal strategies perform clustering on the whole dataset [33]. In contrast, modified fuzzy clustering approaches apply a clustering algorithm to test the complete dataset utilizing Literal Fuzzy C-Means (LFCM) [33, 32]. In most cases, FCM is suitable for clustering of data having linear data distribution in feature space [34]. For handling non-linear shape clusters [35], Chen and Kong [36] introduced the concept of the kernel function. Zhang and Chen [37] developed a Kernel Fuzzy C-Means (KFCM) algorithm by replacing the Euclidean distance metric used in FCM algorithms with a kernel metric. The Fuzzy Kernel C-Means (FKCM) algorithm was proposed by integrating FCM with a mercer kernel function to handle the issues that arise with fuzzy clustering [38]. The FKCM algorithm is suitable for clustering data that form a cluster with a linear and non-linear distribution of data in feature space. Above mentioned fuzzy clustering

algorithms efficiently perform clustering of small data but cannot efficiently cluster VL Data.

However, research along this line is still in progress with the fuzzy consensus clustering algorithmic studies in fuzzy systems [39]. Finding a fuzzy consensus partition from multiple fuzzy basic partitions efficiently and robustly is still an exciting open problem calling for further investigation. Consensus clustering is emerging as a promising solution for multi-source and heterogeneous data clustering. It aims to find a single partition that most agrees with multiple existing basic partitions [39]. Nevertheless, the relatively high time and space complexity preclude it from real-life large-scale data clustering. It has been widely recognized that consensus clustering can generate robust clustering results; find usual clusters, handle noise, outliers and sample variations, and integrate solutions from multiple distributed sources of data or attributes [40]. A recent study also gives rigorous proof of the robustness and generalizability of consensus clustering, which lays a theoretical foundation for the success of consensus clustering [39]. With the explosive growth of big online data in recent years, researchers and practitioners realized that a single clustering might fail with complex data, such as high dimensional genome data. Hence, the introduction of consensus clustering to fuzzy clustering becomes natural, and Fuzzy Consensus Clustering (FCC) thus emerges as a new research frontier. 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 [41, 42].

A likely reason for this is that these algorithms are not scalable to handle Big Data [41, 42]. Genomics is a Big Data science and will get much bigger very soon, but it is not known whether the needs of genomics will exceed other Big Data domains. Projecting to the year 2025, Stephens [43] compared genomics with three other significant generators of Big Data: astronomy, YouTube, and Twitter. With the increasing volume (in order of petabytes) of genome data originated from thousands of sources, the present Single Nucleotide Polymorphism (SNP), protein, and Deoxyribonucleic acid (DNA) sequence analysis tools have been found inadequate. Since the bioinformatics

field of genomics entered into the clustering of the high-dimensional information, this raises the prerequisite of creating scalable clustering algorithms to handle the high dimensional genome data [44, 45, 46]. To cluster genome data using clustering algorithms, there is a need to develop feature extraction methods for genome data to be applied to clustering algorithms. Earlier research works have used various traditional methods for feature extraction of SNP and protein data, but the efficiency is poor for massive data [47]. Many researchers used machine learning methods to extract relevant information from various genome datasets [48, 49]. Researchers also addressed this issue by developing scalable feature extraction methods for genome data [50, 51]. Due to the massive generation of genome data day by day, there is a need to innovate advancements in the present method/technology to handle such exponentially growing data. Motivated by the success of Big Data frameworks, this thesis investigated scalable feature extraction techniques for genome data and clustering of genome sequences by proposing various scalable fuzzy clustering models. However, the proposed scalable clustering models are general purpose which can be applied to any problem.

1.1 Motivation

This dissertation is a study of the design and analysis of scalable fuzzy based clustering algorithms for handling VL data and feature extraction techniques for huge genome Data.

Within the past few decades, cluster analysis has played a crucial part in machine learning, pattern recognition, data mining, and genome handling [21]. Recently, fuzzy clustering algorithms are broadly acknowledged for organizing unstructured information due to their capability to handle uncertainty. Clustering of genome sequences can help to identify unique and new genes and provide better suggestions to find cluster of disease. Despite of wide acceptance of fuzzy clustering, it suffers from scalability issues. Consequently, there is a need to plan scalable clustering approaches for dealing with VL and huge genomics problems. Big Data analytic frameworks are required to implement scalable fuzzy clustering algorithms and feature extraction techniques for

huge genome data. Apache Spark is one of the most broadly utilized Big Data analytic frameworks to design scalable clustering algorithms for handling huge genomics data and VL data.

This thesis investigates scalable kernelized clustering algorithms to solve Big Data problems. These kernelized clustering algorithms are evolved to deal with the non-linear separable issues by applying a kernel Radial Basis Function (RBF), which maps the input data space non-linearly into a high dimensional feature space. Also, the novel scalable kernelized clustering algorithms are tested on various benchmark datasets in terms of performance metrics for handling Big Data. The analytical study is also performed for scalable clustering algorithms in terms of space and time complexity. Additionally, we have investigated the consensus method in fuzzy clustering and developed a novel Scalable Incremental Fuzzy Consensus Clustering (SIFCC) algorithm for handling Big Data to improve quality of cluster. Furthermore, to investigate the performance of the scalable clustering algorithm on real-life genome data, massive protein and SNP datasets of complex plant genomes collected from Indian Council of Agricultural Research-Indian Institute of Soybean Research (ICAR-IISR), Indore, India for the feature extraction and clustering of genome sequences. This clustering helps ICAR-IISR scientists to efficiently characterize SNP and protein sequences of different plant species. To preprocess huge SNP and protein sequences, we propose novel scalable feature extraction approaches that extract fixed-length numeric feature vectors for SNP and protein sequences. The proposed scalable algorithms are designed to handle Big Data by utilizing the Apache Spark cluster computing framework. Additionally, the performance of the scalable clustering algorithms is investigated on a massive SNP and protein dataset in terms of validity measures for the quality of clustering results. We have also investigated the performance of the proposed clustering algorithms on the massive protein data of Coronavirus Disease-19 (COVID-19) caused by the SARS-CoV-2 virus. Before applying SARS-CoV-2 data on clustering algorithms, we have preprocessed the protein sequences of SARS-CoV-2 data using our developed feature extraction techniques. Our approaches accomplish a significant trade-off between clustering quality and the computational attempt required to lower

the run-time.

1.2 Objectives

In this thesis, we aim to achieve the following objectives:

(i) To develop scalable kernelized fuzzy-based iterative clustering algorithms for handling Big Data that speed up the Big Data clustering process by reducing the run-time and optimizing the storage space. Further, it is proposed to develop an analytical formulation for space and time complexity.

(ii) To develop a method that can combine scalable fuzzy clustering and fuzzy consensus clustering to improve the quality of clusters for Big Data using Apache Spark cluster.

(iii) To develop a novel scalable feature extraction algorithm for huge SNP sequences, which extracts 12-dimensional numeric feature vector. Further, it is used as input to the proposed scalable clustering algorithms to cluster massive SNP data.

(iv) To develop novel scalable feature extraction algorithms for massive protein sequences, which extracts fixed-length numeric feature vectors of 60-dimensions and 6-dimensions.

(v) To investigate massive SARS-CoV-2 protein datasets on developed scalable feature extraction and scalable kernelized clustering algorithms.

1.3 Thesis Contributions

The significant contributions of the work done in this field is to design and develop the clustering and feature extraction techniques for huge genome data. These contributions are divided into two broad categories. Firstly, the design of scalable fuzzy clustering algorithms for handling VL data. These algorithms are further applied to

the clustering of genome data. Secondly, the design of scalable feature extraction techniques for huge SNP and protein sequences using Apache Spark cluster. Additionally, investigation of SARS-CoV-2 genome data has been performed using the proposed scalable feature extraction technique and scalable clustering algorithms.

A brief overview of our research contributions is provided below, and more details are available in the later chapters.

Contribution I:

To overcome the limitations of Scalable Random Sampling with Iterative Optimization Fuzzy C-Means (SRSIO-FCM) [52], kernelized clustering algorithms are evolved that deal with the non-linear separable problems by applying a kernel RBF which maps the input data space non-linearly into a high dimensional feature space. The designed kernelized clustering algorithm; Kernelized Scalable Random Sampling with Iterative Optimization Fuzzy C-Means is named as KSRSIO-FCM. It processes the Big Data by partitioning it into various subsets and performs parallel processing of each subset using the Apache Spark Big Data processing framework. In the proposed approach, we eliminate the need for storing the membership matrix, which makes the execution of the proposed algorithm faster by reducing the run-time. In addition to this, we resolve the problem of using the highly deviated cluster centers to cluster the current subset. Thus, it provides faster convergence by taking fewer iterations during the clustering of each subset. It leads to the significant improvement in run-time due to the reduced space and time complexity. For a fair comparison of KSRSIO-FCM, we have also designed a kernelized model of the existing Scalable Literal Fuzzy C-Means (SLFCM) [52] named as KSLFCM algorithm. The performance of the proposed KSRSIO-FCM is evaluated in comparison with KSLFCM. The proposed scalable kernelized clustering algorithms have been tested on standard benchmark datasets and replicated big datasets. The results are compared with other scalable models. The performance on various benchmark datasets is evaluated in terms of NMI, ARI, and F-score.

Contribution II:

We have further explored Fuzzy Consensus Clustering (FCC) [53] for the development of a scalable model that can handle Big Data. The motivation of proposing another scalable algorithm based on fuzzy consensus clustering is to improve the cluster quality of Big Data. The proposed Scalable Incremental Fuzzy Consensus Clustering (SIFCC) algorithm aims to find a single partition of data that agrees as much as possible with existing basic segments. It has been implemented on an Apache Spark cluster framework, a distributed data stream environment for handling Big Data by considering the data as a set of incrementally processed subsets. For a fair comparison of SIFCC, we have also designed a scalable model of the existing FCC algorithm, i.e., SFCC [53]. Thus, the proposed SIFCC algorithm is compared with the SFCC algorithm by testing them on standard benchmark datasets and replicated big datasets. The performance is then evaluated on various benchmark datasets in terms of NMI, ARI, and F-score.

Contribution III:

Discovering clusters in the high-dimensional genomics data is exceptionally challenging for bioinformatics researchers for genome sequence analysis. We have developed a scalable preprocessing approach using Apache Spark cluster for generating numerical feature vectors for massive SNP sequences. The real-world SNP datasets consist of two different plant species, i.e., soybean [54], and rice [55, 56, 57]. These are preprocessed using the proposed scalable SNP preprocessing approach. Further, it is used as input to the proposed KSRSIO-FCM and SIFCC algorithms (discussed in contributions I and II) to cluster huge SNP datasets, and, thus, the performance is evaluated in terms of Silhouette Index (SI) and Davies Bouldin Index (DBI) measures.

Contribution IV:

We further develop scalable feature extraction techniques for preprocessing of huge protein sequences using Apache Spark cluster. A 60-dimensional Scalable Protein Feature extraction approach (60d-SPF) and a 6-dimensional Scalable Co-occurrence-based Probability-Specific Feature Extraction Approach (6d-SCPSF) has been developed. Both the proposed 60d-SPF and 6d-SCPSF approaches capture the statistical prop-

erties of amino acids to create a fixed-length numeric feature vector representing each protein sequence in terms of 60-dimensional and 6-dimensional features, respectively. Since the proposed scalable feature extraction methods are implemented on Apache Spark cluster, it performs efficient feature extraction of data due to its in-memory cluster computing capability. Executing it in the Apache Spark cluster makes the proposed feature extraction methods (60d-SPF and 6d-SCPSF) scalable, which are further applied to a real-world huge protein dataset. Further, it is used as input to the developed SRSIO-FCM algorithm to cluster huge protein datasets, and, thus, the performance is evaluated in terms of SI and DBI measures.

Contribution V:

The research contributions discussed in III and IV are for plant genome datasets. To investigate the SARS-CoV-2 protein data, we have applied the proposed scalable protein feature extraction 60d-SPF technique (as discussed in Contribution IV) to generate numerical feature vectors from huge SARS-CoV-2 genome datasets. Further, it is used as input to the proposed KSRSIO-FCM method (as discussed in Contribution I) to cluster huge SARS-CoV-2 data. Then, the performance is evaluated in terms of SI and DBI measures.

1.4 Organization of the Thesis

This thesis is organized into eight chapters. A summary of each chapter is provided below:

Chapter 1 (Introduction)

This chapter describes the background knowledge of clustering, the motivation of our work, and the contribution of this thesis.

Chapter 2 (Literature Survey and Research Methodology)

This chapter provides a detailed literature survey and a summary of various clustering techniques. After that, an overview of Big Data frameworks for scalable algorithms is presented. A study of scalable fuzzy clustering for Big Data and the feature extraction approaches designed for SNP and protein sequences of genome

sequences are also presented. It also provides evaluation metrics for performance analysis. Finally, a brief overview of genome datasets used in the experimental study is presented.

Chapter 3 (Scalable Kernelized Fuzzy Clustering Algorithms for Handling Big Data)

In this chapter, we proposed a novel scalable kernelized clustering algorithm for handling Big Data called the KSRSIO-FCM algorithm. The kernelized clustering algorithms deal with the non-linear separable problems by applying a kernel RBF which maps the input data space non-linearly into a high dimensional feature space. It is implemented and tested using the Apache Spark cluster for handling Big Data. This chapter presents how the proposed approach scales with the non-linear separable problems for Big Data compared to other scalable linear separable algorithms. The chapter finally reports the experimental evaluation that compares our proposed KSRSIO-FCM algorithm with other proposed scalable models, i.e., KSLFCM, SRSIO-FCM, and SLFCM, respectively.

Chapter 4 (Scalable Incremental Fuzzy Consensus Clustering Algorithms for Handling Big Data)

In this chapter, we present scalable incremental fuzzy consensus clustering algorithms using Apache Spark cluster. For developing a scalable incremental fuzzy consensus clustering, SRSIO-FCM [52] shown in Chapter 2 is utilized for finding concatenated basic segments, which is used as an input to the scalable incremental fuzzy consensus clustering. This chapter presents how the proposed approach performs better in comparison with the scalable fuzzy consensus clustering algorithm. The chapter finally reports the experimental evaluation that compares our proposed SIFCC algorithm with other proposed scalable models, i.e., the SFCC and SRSIO-FCM.

Chapter 5 (Design of Novel Scalable Feature Extraction Algorithm for Huge SNP Sequences with Application of Scalable Fuzzy Clustering Algorithms:)

The methods proposed in Chapters 3 and 4 do not consider raw genome sequences

for clustering massive SNP sequences. For addressing this issue, this chapter includes the proposal of scalable feature extraction approach for SNP sequences, which extracts 12-dimensional numerical feature vectors from massive SNP sequences. After that, extracted feature vectors from SNP sequences are applied to scalable clustering algorithms for clustering of massive SNP data is presented.

Chapter 6 (Design of Novel Scalable Feature Extraction Algorithms for Huge Protein Sequences with Application of Scalable Fuzzy Clustering Algorithm)

For clustering of massive protein sequences, novel scalable feature extraction techniques for protein sequences, a 60-dimensional Scalable Protein Feature (60d-SPF) extraction approach and a 6-dimensional Scalable Co-occurrence-based Probability-Specific Feature (6d-SCPSF) extraction approach are presented in this chapter. Thereafter, the protein sequences are applied to a scalable clustering algorithm for clustering.

Chapter 7 (Investigation of Massive SARS-CoV-2 Protein Datasets on Developed Scalable Feature Extraction and Scalable Fuzzy Clustering Algorithms)

Preprocessing and clustering of massive SARS-CoV-2 datasets are presented in this chapter. The scalable feature extraction techniques presented in Chapter 6 is used to preprocess soybean protein sequences. This chapter briefly describes preprocessing of raw SARS-CoV-2 protein datasets using the 60d-SPF method. Further, it is applied to the developed scalable kernelized clustering algorithms presented in Chapter 3 to find a cluster for massive SARS-CoV-2 protein sequences.

Chapter 8 (Conclusions and Future Work)

This chapter briefly describes the contribution of this thesis and the possible future directions of our work.

Chapter 2

Literature Survey and Research Methodology

This chapter discusses various clustering algorithms and issues related to those clustering algorithms in Section 2.1. The study related to Big Data processing frameworks is presented along with the parallel processing algorithms for handling Big Data in Section 2.2. Further, a survey of scalable fuzzy clustering algorithms for processing VL datasets is carried out in Section 2.3. Then, the focus of the discussion shifts to the feature extraction algorithms for genome data and matters about genome clustering in Section 2.4. Further, Section 2.5 discusses performance measures. The last section (Section 2.6) presents the details of the real-life genome datasets used in this study.

2.1 Clustering and its type

Clustering [58] is a data mining strategy that is extensively taken into consideration for mining significant information underlining unlabeled data. It is an unsupervised learning approach applied on data to form groups such that data samples within a group share a similarity, this results in finding patterns of datasets [59]. Clustering of objects is required for different purposes in various fields of engineering [60, 61, 62], science and technology [63], humanities [64, 65], bioinformatics [66], medical science [67, 68], gene analysis [69, 70], indexing and compression [71], image analysis [72], signal processing [73], text classification [74, 75], cyber security, and data mining [76].

Over the last five decades, numerous clustering algorithms [77, 78, 79, 80] have been designed based on different hypotheses and applications. Scientific classification of clustering algorithms was depicted by Fahad et al. [80]. An outline of the taxonomy of clustering algorithms is displayed in Figure 2.1. Within the taxonomy of clustering algorithms, numerous refinements of clustering algorithms are displayed, but our center is on the refinement of partitional clustering algorithm [1, 11, 12]. A partitional clustering strategy classifies the data into different groups based on the characteristics and closeness of the information. It is the data investigators that decide the number of clusters required to be created for the clustering strategies [59]. The partitional clustering algorithms minimize a given clustering model by iteratively relocating data samples between clusters until a (locally) ideal partition is accomplished [81]. Partitional clustering is broadly divided into hard (or crisp) and soft. The subtle elements of both these categories of algorithms are explained below. In hard clustering algorithms, each data sample must belong to absolutely one cluster for the particular dataset. Hard clustering strategies are based on the factor that the membership vector is represented in binary because either an item belongs to a cluster or it does not. Consequently, the groups in a hard clustering are disjoint. Due to disjoint clusters

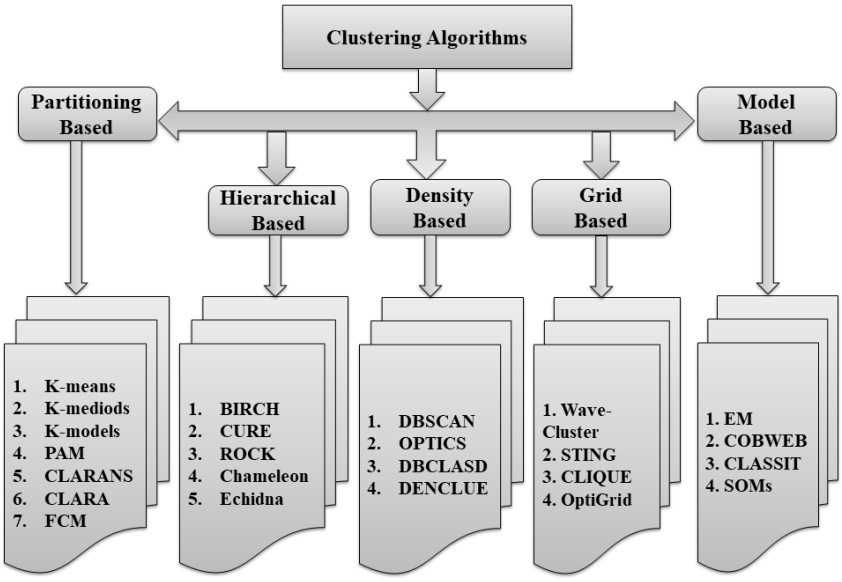


Figure 2.1: Taxonomy of clustering algorithms

formed with the hard clustering, it may lose relevant information and be unable to capture the structure of real datasets. There are no precise boundaries between the clusters. Due to the loss of crucial details, hard clustering leads to meaningless clustering [82]. One of the most commonly used hard clustering algorithms is K-means; a distance-based partitioning algorithm [1]. The limitations of the K-means clustering algorithm are as follows: 1) The K-means algorithm requires the number of clusters to be known in advance, which implies it should initialize a set of cluster centers at the beginning [1]. 2) K-means are unable to ideally cluster the data where clusters are of varying sizes and density. 3) Although K-means clustering is quite efficient in terms of computational time, it is susceptible to outliers [83].

Soft clustering is a grouping of the data samples such that a data sample can belong to multiple clusters. For soft clustering algorithms, we need to compute a fuzziness coefficient that controls the degree of fuzziness. Fuzzy C-Means (FCM) is a famous soft clustering algorithm. This method (developed by Dunn in 1973 [84] and improved by Bezdek in 1981 [85]) is frequently used in many applications. The FCM working is such that data samples are assigned probabilities which are essentially expressing the strength of belongingness of data samples to a cluster [86]. A membership value is created during the FCM process which expresses the probability of the membership, ranging from 0 to 1 that indicates how similar an item is to the mean of the cluster. Fuzzy set theory is used in the clustering algorithm so that a data sample can belong to multiple clusters [87]. A fuzzy clustering determines fuzzy partitions expressed by the membership matrix, M of size $n \times c$. The degree of membership of a data sample i in cluster j is represented by m_{ij}. This is subjected to the following constraints [85, 88]:

$$m_{ij} \in [0,1], 1 \leq i \leq n, 1 \leq j \leq c \tag{2.1}$$

$$\sum_{j=1}^{c} m_{ij} = 1, 1 \leq i \leq n \tag{2.2}$$

$$\sum_{i=1}^{n} m_{ij} > 0, 1 \leq j \leq c \tag{2.3}$$

The concept of fuzzy clustering analysis has been effectively incorporated into nu-

merous clustering algorithms [89, 90, 91, 92]. FCM is a method of clustering which allows one piece of data to belong to two or more clusters. It is also known as Literal Fuzzy C-Means (LFCM) clustering [32]. LFCM partitions the n data samples represented by x_i, $i = 1, ..., n$ into c clusters. LFCM executes iteratively until the difference in the previous and current iteration cluster centers are less than the defined termination criteria (ϵ) [88]. ***Algorithm 2.1***, presents the steps of the LFCM algorithm.

Algorithm 2.1 Literal Fuzzy C-Means (LFCM) to Iteratively Minimize $J_p(M, V^{'})$

Input: X, V, c, p, ϵ

Output: $M, V^{'}$

1: **Randomly** initialize cluster centers $V = \{v_1, v_2.., v_c\}$.

2: **Compute** the membership matrix M using Eq. (2.4) such that $\sum_{j=1}^{c} m_{ij} = 1$.

$$m_{ij} = \frac{\parallel x_i - v_j \parallel^{\frac{-2}{p-1}}}{\sum_{k=1}^{c} \parallel x_i - v_k \parallel^{\frac{-2}{p-1}}}, \forall i, j \tag{2.4}$$

3: **Compute** the cluster centers $v_j^{'}$ for $j = 1, 2, .., c$.

$$v_j^{'} = \frac{\sum_{i=1}^{n} [m_{ij}]^p x_i}{\sum_{i=1}^{n} [m_{ij}]^p}, \forall j \in [1, c] \tag{2.5}$$

4: **If** $\parallel V^{'} - V \parallel < \epsilon$, then stop;

5: **Else**: $V = V'$ and go to Line 2.

6: **Return** $M, V^{'}$

The purpose is to minimize the objective function (J_p). In LFCM, the membership degree m_{ij} can take any values between 0 and 1. The objective function of LFCM can be formulated as follows:

$$J_p(M, V^{'}) = \sum_{i=1}^{n} \sum_{j=1}^{c} (m_{ij})^p \|x_i - v_j^{'}\|^2 \tag{2.6}$$

where $V^{'}$ denote the set of final cluster centers such that $V^{'} = \{v_1^{'}, v_2^{'}, .., v_c^{'}\}$. The parameter p represents the fuzziness parameter for each data sample. The LFCM

algorithm is based on the iterative optimization of the objective function given in Eq. (2.6) by updating the membership matrix M and cluster centers to get a set $V^{'}$ of final cluster centers.

The run-time complexity of LFCM algorithm is $O(ntdc^2)$ [29]. Despite of the wide acknowledgment of the LFCM algorithm, it endures few drawbacks. There are numerous parameters in LFCM, which essentially influence the execution and outcomes of fuzzy clustering. One of the major problems with the LFCM clustering approach is the scalability, i.e., it's very time-consuming when applied on large datasets. On huge datasets, characterized as 10^{10} bytes concurring to Huber's description [22], the time taken by conventional clustering algorithms is too long. Subsequently, researchers have disallowed the use of traditional algorithms and have developed new clustering algorithms [27, 29, 30, 32, 93] that perform clustering by processing the data in subsets [93]. Recently, Bharill [52] proposed a scalable model of the LFCM named SLFCM algorithm for clustering of Big Data. The SLFCM performs clustering by processing the data in subsets. The detailed description of SLFCM is presented in Section 2.3.1.

The limitations of the FCM clustering algorithm is as follows:

(i) The FCM algorithm requires long computational time [32].

(ii) FCM performs clustering on entire dataset and it does not work well for Big Data [52].

(iii) Sensitivity to noise and low (or even no) membership degree for outliers (noisy data samples) [94].

The FCM clustering algorithm adopts iterative optimization to minimize an objective function using a similarity measure on feature space. In most cases, FCM is suitable for clustering of the data having linear data distribution in feature space [34]. For handling non-linear shape clusters, the concept of kernel function is introduced [35]. A detailed description of kernel based fuzzy clustering method is presented next.

Kernel based Fuzzy C-Means algorithm:

The main idea of the Kernel Fuzzy C-Means (KFCM) algorithm is described as follows. The kernel method maps the input data space nonlinearly into a high dimensional feature space. In analyzing past research work [95, 96, 97, 98, 99, 100], the researcher demonstrated that the adaptation of kernel functions could improve Euclidean distance measure over customary clustering algorithms. Many researchers have worked on the clustering of non-linear data, which are discussed in detail as follows: Huang [95] discussed the limitation of FCM by mapping non-linear data into a feature space using the kernel tricks that extend the FCM method with multiple learning. Baili [96] proposed the FCM with multiple kernels, which allows a soft linear partitioning of the feature space. Timothy [97] compares the efficacy of FCM algorithms and their kernelized version for the clustering of very large data. In addition to this, Liu and Xu [98] built up a kernelized fuzzy attribute c-means clustering algorithm that updates the distance by utilizing kernel-induced distance. Graves and Pedrycz [99] set up a far-reaching investigation of kernel-based fuzzy clustering. Tsai and Lin [100] proposed KFCM-σ, an extended version of KFCM using a distance metric, which allows the grouping of non-linear separable data.

The main concept behind the kernel based algorithms is the kernel trick [97]. The kernel trick is an implicit non-linear map (ϕ) from the input space X to a high dimensional feature space R [101].

$$\phi : x \rightarrow \phi(x) \in R^d \tag{2.7}$$

where the data samples $\{x_1, x_2, ..., x_n\} \subseteq X$. In this, an input data space with a lower dimension is mapped to potentially much higher dimensional feature space (R) or inner product [102]. The inner product operation in the kernel space can be expressed by a mercer kernel [102] represented as a function K below:

$$K(x_i, v_j) = \phi(x_i)^\top \phi(v_j) \tag{2.8}$$

where $x_i, v_j \in R^d$ such that $i = 1, .., n$ and $j = 1, ..., c$. Thus, using this mapping ϕ,

the kernelized version of FCM [97] is represented as follows:

$$J_p(M,\ V') = \sum_{i=1}^{n}\sum_{j=1}^{c} m_{ij}^p \|\phi(x_i) - \phi(v_j')\|^2, \qquad p > 1 \tag{2.9}$$

Like FCM, each data sample x_i fulfills the constraint $\sum_{j=1}^{c} m_{ij} = 1$. By the kernel substitution, we have the following equation.

$$\begin{aligned} &\|\phi(x_i) - \phi(v_j)\|^2 = (\phi(x_i) - \phi(v_j))^\top(\phi(x_i) - \phi(v_j)) \\ &= \phi(x_i)^\top\phi(x_i) - \phi(v_j)^\top\phi(x_i) - \phi(x_i)^\top\phi(v_j) + \phi(v_j)^\top\phi(v_j) \\ &= K(x_i, x_i) + K(v_j, v_j) - 2K(x_i, v_j) \end{aligned} \tag{2.10}$$

In this manner, a new class of non-Euclidean distance measures in the original input space (additionally with a Euclidean distance in feature space) is achieved. Thus, different kernels will generate different measures for the original space. In this work, $K(x_i, v_j)$ is the RBF kernel [103], which is a well-known kernel function represented as follows:

$$K(x_i, v_j) = \exp(-\|x_i - v_j\|^2/\sigma^2) \tag{2.11}$$

Where, σ is denoted as the kernel parameter. The choice of kernel parameter is the most critical task [103]. In this work, the kernel parameter is selected in the following way:

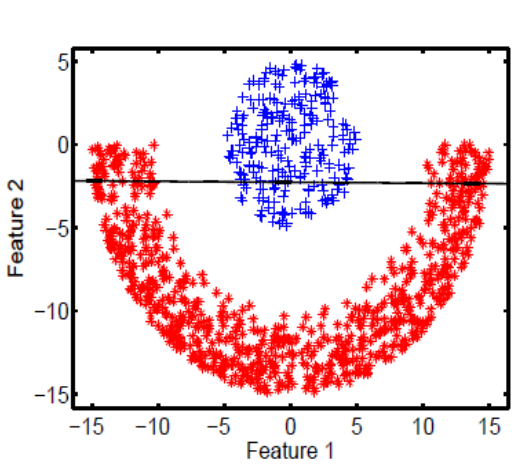
$$\sigma = \sqrt{\frac{\sum_{i=1}^{s}(z - \bar{z})}{n-1}} \tag{2.12}$$

Where, $z_i = \|x_i - \bar{x}\|$ and $\bar{z}$ is the average of all distances z_i. In case of RBF kernel (also called Gaussian Kernel), $K(x_i, x_i) = 1$ and $K(v_j, v_j) = 1$ [103]. Thus Eq. (2.10) is simplified to

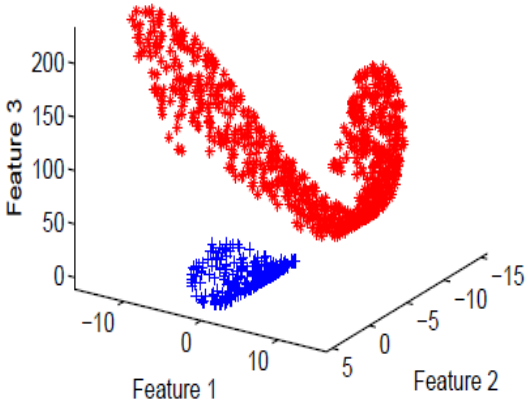
$$\|\phi(x_i) - \phi(v_j)\|^2 = 2(1 - K(x_i - v_j)) \tag{2.13}$$

so, the Eq. (2.9) can be edited as:

$$J_p(M,\ V') = 2\sum_{i=1}^{s}\sum_{j=1}^{c} m_{ij}^p (1 - K(x_i, v_j')) \tag{2.14}$$



(a) Lower Dimension.



(b) Data is projected into higher dimensions by creating a new dimension by taking the square of Feature 1 as the third dimension.

Figure 2.2: Visualization of data in lower and higher dimensions.

where, $V' = \{v'_1, v'_2, ...v'_c\}$. To minimize the objective function, the membership matrix m_{ij} and the updated cluster center v'_j need to be computed.

$$m_{ij} = \frac{(1 - K(x_i, v_j))^{1/(p-1)}}{\sum_{j=1}^{c}(1 - K(x_i, v_j))^{1/(p-1)}} \tag{2.15}$$

$$v'_j = \frac{\sum_{i=1}^{s} m_{ij}^p K(x_i, v_j) x_i}{\sum_{i=1}^{s} m_{ij}^p K(x_i, v_j)} \tag{2.16}$$

We need to understand, what the kernel trick essentially does? It isn't generally conceivable to isolate the two classes from one another in the lower dimensional space utilizing a hyperplane. Thus, information is projected into the higher dimensional space so a hyperplane can easily separate it [104]. This can be visualized in Figure 2.2 where Figure 2.2(a) represents data in low dimension space, and it can be seen that data is not linearly separable. After adding one more dimension, i.e., Feature 3= (Feature 1)^2, data can be linearly separated by a hyperplane as shown in Figure 2.2(b).

The kernel based fuzzy clustering algorithm is explained in ***Algorithm 2.2***. Dur-

ing the execution of ***Algorithm 2.2***, the Eq. (2.16) is updated till there is no significant change in the values of cluster centers, and then the algorithm terminates.

Algorithm 2.2 Kernel based Fuzzy Clustering Algorithm

Input: X, c, p, ϵ; X is an array of data samples such that $X = \{x_1, x_2, ...x_n\}$.

Output: $M, V^{'}$

1: **Randomly** initialize $V = \{v_1, v_2, ...v_c\}$.

2: **Compute** the membership matrix by using Eq. (2.15).

3: **Compute** the set of final cluster centers $V' = \{v'_1, v'_2, ...v'_c\}$ by using Eq. (2.16).

4: If $\parallel V' - V \parallel < \epsilon$ then stop, otherwise continue with step 2.

5: **Return** $M, V^{'}$

There are a large number of methods available to perform clustering, but it is often unclear which method is best suited to the data and how to quantify the quality of the clustering produced. To develop a method for executing multi-algorithmic and multi-condition clustering for improving the quality of clusters, we explored the consensus clustering approach [105]. A consensus clustering plans to consolidate multiple clustering to get better, gradually more robust clustering results, which has shown interest in discovering unusual clusters, handling noise, and incorporating clustering arrangements from various distributed sources [106, 107, 108]. The detailed description of consensus clustering is explained in a subsequent section.

2.1.1 Consensus Clustering

Consensus clustering means finding different segments of the data sample in different existing Basic Segments (BSs). An ongoing report likewise establishes a hypothetical framework for the achievement of consensus clustering, which gives thorough verification of the stability and generalization of consensus clustering [39]. Many researchers have worked on consensus clustering, which is discussed in detail as follows: Liu et al. [39] developed spectral ensemble clustering (SEC) based on the co-association matrix, which is an effective contender to some best in class consen-

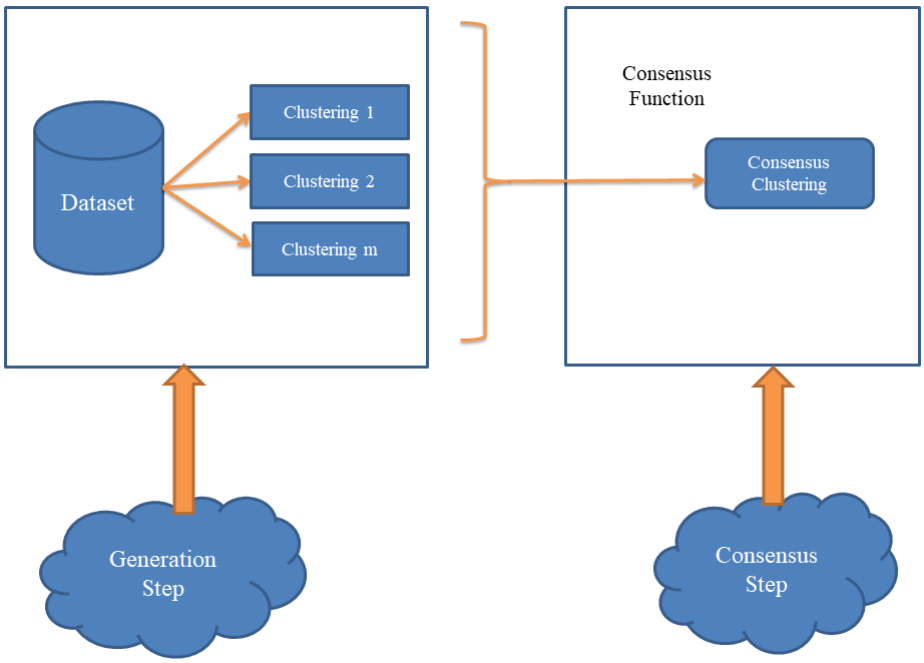


Figure 2.3: Architecture of Consensus Clustering

sus clustering techniques and is additionally appropriate for Big Data clustering. Wu et al. [105] introduced the K-means consensus clustering (KCC) algorithm for handling incomplete basic partitioning. Li and Liu [99] extended the work of Wu et al. [105], where KCC suffers from initialization sensitivity. While greedy optimization of KCC termed as GKCC intends to illuminate the affectability of K-means initialization, GKCC [109] consistently merges greedy K-means and KCC to accomplishes the benefits acquired by GKCC. Liu et al. [110], extended the work of Liu et al. [39], to address the difficulty emerging from incomplete basic partitions, in light of which they proposed a row-segmentation scheme for Big Data clustering. Figure 2.3 shows the architecture of consensus clustering.

Consensus clustering [106] is broadly divided into two parts, i.e., implicit or explicit objective functions. The process of using implicit objective functions follows heuristic functions to find output rather than directly establishing objective functions. Representative processes include an association matrix based procedure [111], graph-based algorithms [112], relabeling, and voting methods [113], kernel-based methods [114],

and genetic algorithm based methods [115]. The methods in explicit objective functions have explicit global objective functions for consensus clustering. Representative procedures consist of a K-means-like algorithm [116], the Expectation-Maximization (EM) algorithm to extend consensus clustering [117], simulated annealing [112], and the combination regularization [118]. Consensus clustering algorithms can be found in the analysis presented by Liu et al. [119] and Huang et al. [120]. Considering the huge number of consensus clustering strategies, the good survey on consensus clustering was presented by Liu et al. [119]. Even though they plan to exhibit a complete prologue to existing techniques, they may disregard a few key strategies and also disregard to reveal the profound association among these strategies. Liu et al. [119] provided a survey on consensus clustering from an embedded point of view. The embedded representation is taken from the co-association matrix, followed by the classical clustering algorithm to complete the consensus segment [119].

In these methodologies and throughout the discussion, we use the following convention. For a d-dimensional real vector $x \in R^d$, $\|x\|_\jmath$ indicates the $l_\jmath - norm$ of x, i.e., $\|x\|_\jmath = (\sum_{i=1}^{d} |x_i|^\jmath)^{1/\jmath}, \jmath \geqslant 1$. $x^\top$ means the transposition of x. The gradient of the multivariate function f is expressed as $\bigtriangledown f$. The logarithm of base 2 is indicated as a log. Let $X = \{x_1, x_2, \ldots, x_n\}$ indicate a set of data samples. X is partitioned into c clusters which are denoted as an accumulation of c clusters of data samples in $\mathbb{C} = \{\mathbb{C}_k | k = 1, \ldots, c\}$, with $\mathbb{C}_k \bigcap \mathbb{C}_{k'} = \emptyset; \forall k \neq k'$, and $\bigcup_{k=1}^{c} C_k = X$ or as a feature vector $\prod = (L_\pi(x_1), \ldots, L_\pi(x_n))^\top$, where L_π maps x_l to some feature in $\{1, 2, \ldots, c\}$, $1 \leq l \leq n$. The issue of consensus clustering is commonly defined as pursues. $\prod = \{\pi_1, \pi_2, \ldots \pi_r\}$ of X is a set of r BSs.

$$\max_{\pi} \sum_{i=1}^{r} \omega_i U(\pi, \pi_i) \tag{2.17}$$

The aim is to find the consensus segment π so Eq. (2.17) is maximized where π denoted as consensus segment, $U : Z_{++}^n \times Z_{++}^n \mapsto \mathbb{R}$ is a utility function. The similarity between π and π_i, is evaluated by utility function and $\omega_i \in [0, 1]$ is a user-defined value for BS π_i, $\sum_{i=1}^{r} \omega_i = 1$ [53]. The success of consensus clustering depends on the utility

function. Assume π and π_i are two segments containing c and c_i clusters, respectively. $n_{kj}^{(i)}$ means the quantity of data sample shared by a group $\mathbb{C}_j^{(i)}$ (in π_i) and group $\mathbb{C}_k$ (in π). Accordingly, $n_{k+} = \sum_{j=1}^{c_i} n_{kj}^{(i)}$ denote the quantities of data sample in $\mathbb{C}_k$ and $\mathbb{C}_j^{(i)}$, respectively. Nearly all regularly utilized utility functions can be characterized by this matrix. For example, let $\mathbb{P}_k^{(i)} = (n_{k1}^{(i)}/n_{k+}, \ldots, n_{kc_i}^{(i)}/n_{k+})^\top$, then noticeable category utility function [121] can be formulated as:

$$U(\pi, \pi_i) \propto \sum_{k=1}^{c} n_{k+} \|\mathbb{P}_k^{(i)}\|_2^2. \tag{2.18}$$

Utility functions [53] have different mathematical properties and also have different degrees of computational difficulty for consensus clustering. The current strategy for crisp consensus clustering is typically designed for one or more utility functions that are less adaptable when applied with real Big Data from different application areas. However, with the explosive growth of online Big Data in recent years, researchers and practitioners have realized that a single fuzzy clustering might fail with complex data, such as high dimensional genome data. Hence, the introduction of consensus clustering to fuzzy clustering becomes natural. Thus Fuzzy Consensus Clustering (FCC) emerges as a new research frontier. A detailed discussion of FCC is present in the subsequent section.

2.1.2 Fuzzy Consensus Clustering

FCC focuses on fusing various existing clustering results and has a global objective function that guides the consensus clustering, which can be solved naturally via a simple FCM-like iterative process. The FCC is explained with a couple of partitioning algorithmic designs. For example, Mojarad et al. [122] proposed a robust clustering ensemble based on sampling and cluster clustering (RCESCC) algorithm, which initially creates a consensus of fuzzy clustering produced by the Fuzzy C-Means (FCM) algorithm on sampled data. Then, a hierarchical clustering algorithm is used for partitioning the clusters and finally assigning the data samples to combine clusters. Zoghlami et al. [123] developed the merging based FCC algorithm in which

partitioned data are logically converged into groups through the worldwide collection. Pedrycz and Hirota [124] proposed a consensus-driven fuzzy clustering based on the FCM algorithm. In this, fuzzy sets are used for the allocation of patterns to clusters, which is used to find good quality of clusters. Hidri et al. [125] extended the work of Pedrycz and Hirota [124] by using consensus clustering to handle Big Data. Like the KCC [105] system, Wu et al. [53] developed FCC utility functions by updating FCC to a weighted piece-wise FCM clustering and thus proposed a new fuzzy contingency matrix. A set of FCC utility functions can turn the FCC into a weighted piece-wise FCM clustering problem. Finally, build the algorithmic framework for the FCC with the custom decision of the utility function. To solve the problems associated with analyzing large amounts of data, vertical and horizontal segmentation schemes are used to parallelize FCC on the Apache Spark framework.

As the name suggests, FCC is the fuzzy concept used in consensus clustering [53]. Now defining FCC mathematically: The objective function, in this case, would be the same as Eq. (2.17), with the incorporated fuzzy concept that uses fuzzification parameter. As mentioned before, the final BS π was an integer vector in consensus clustering. Still, in fuzzy case, a new variable μ is defined as $(n \times c)$ membership matrix without changing the meaning. Similarly, π_i would replace by $\mu^{(i)}$, where $\mu^{(i)}$ is $(n \times c_i)$ membership matrix; c_i denotes the number of clusters in i^{th} BS. The aim is to find the fuzzy consensus segment μ for optimization of the following goal;

$$\max_{\boldsymbol{\mu}} \sum_{i=1}^{r} \omega_i U_f(\boldsymbol{\mu}, \boldsymbol{\mu}^{(i)}; p) \tag{2.19}$$

Where, U_f is the fuzzy utility function, $p \in (1, +\infty)$ is a user-defined fuzzification parameter, $\omega = (\omega_1, \omega_2, ..., \omega_r)^\top$ is the vector of user-defined weights for BSs, with $\sum_{i=1}^{r} \omega_i$=1. The success of FCC depends on the utility function [53]. A fuzzy utility function for the FCC is defined as follows:

$$U_f(\boldsymbol{\mu}, \boldsymbol{\mu}^{(i)}; p) \;=\; \sum_{k=1}^{c} \|\mu_{\cdot k}^{p}\|_1 \Phi(v_k^{(i)}) - \sum_{l=1}^{n} \|\mu_{l\cdot}^{p}\|_1 \Phi(\mu_{l\cdot}^{(i)}) \tag{2.20}$$

Where, Φ is a convex function (the second norm in our case), for example, Φ for the utility function is the squared $l_2 - norm$, which is an obvious convex function for each component of a discrete distribution. Φ is a function defined on discrete distributions. c is the number of clusters in FCC, $\mu^p_{.k} = \{\mu^p_{1k}, \mu^p_{2k}, ..., \mu^p_{nk}\}$ represents a k^{th} column vector in μ, $\mu^p_{l.} = \{\mu^p_{l1}, \mu^p_{l2}, ..., \mu^p_{lc}\}$, represents a l^{th} row vector in μ, $\mu^{(i)}_{l.} = \{\mu^{(i)}_{l1}, \mu^{(i)}_{l2}, .., \mu^{(i)}_{lc_i}\}$, and $\mu^{(i)}_{l.}$ denotes l^{th} row of i^{th} BS. Let $Y = \{y_1, y_2, y_3....y_n\}$ be the concatenated basic segments, which is of dimension $(n \times \sum_{i=1}^{r} c_i)$, and it is derived from the set of BSs; $\prod = \{\mu^{(1)}, \mu^{(2)}, ..., \mu^{(r)}\}$. y_l corresponds to l^{th} data sample in Y, which shows the membership degrees of x_l in a cluster of each BS. Thus, y_l is a $\sum_{i=1}^{r} c_i$ dimensional vector with $\|y_l\|_1 = r$; $\forall l$.

$$y_l = (\mu^{(1)}_{l.}, \ldots, \mu^{(i)}_{l.}, \ldots, \mu^{(r)}_{l.})^\top; \quad \forall l \tag{2.21}$$

Next, the cluster similarity matrix, $V = \{v_1, v_2, v_3...v_c\}$, is defined as using the below mentioned equations.

$$v_k = (v^{(1)}_k, \ldots, v^{(i)}_k, \ldots, v^{(r)}_k)^\top \tag{2.22}$$

The v_k is also a $(\sum_{i=1}^{r} c_i)$- dimensional vector. Let μ_{lk} and $\mu^{(i)}_{lj}$ express the membership degrees of y_l to $\mathbb{C}_k$ and $\mathbb{C}^{(i)}_j$, respectively. After that, the membership degree matrices are defined as follows: $\boldsymbol{\mu} = [\mu_{lk}]_{n \times c}$ and $\boldsymbol{\mu}^{(i)} = [\mu^{(i)}_{lj}]_{n \times c_i}$. Let

$$\mu_{.k} = (\mu_{1k}, \ldots, \mu_{lk}, \ldots, \mu_{nk})^\top \tag{2.23}$$

$$\mu^{(i)}_{.j} = \left(\mu^{(i)}_{1j}, \ldots, \mu^{(i)}_{lj}, \ldots, \mu^{(i)}_{nj}\right)^\top \tag{2.24}$$

where, $\mu_{.k}$ represents k^{th} column vector in μ, and $\mu^{(i)}_{.j}$ represents a vector of membership degree of data samples (1 to n) in j^{th} cluster of i^{th} BS. Now, to transform FCC into FCM [53], following equations are defined as follows:

$$f(\mu^{(i)}_{l.}, v^{(i)}_k) = \Phi(\mu^{(i)}_{l.}) - \Phi(v^{(i)}_k) - (\mu^{(i)}_{l.} - v^{(i)}_k)^\top \nabla \Phi(v^{(i)}_k) \tag{2.25}$$

$$J_{pf}(\boldsymbol{\mu}, V; \prod) = \sum_{k=1}^{K} \sum_{l=1}^{n} (\mu_{lk})^p \left(\sum_{i=1}^{r} \omega_i f(\mu_{l\cdot}^{(i)}, v_k^{(i)}) \right) \tag{2.26}$$

Where, μ_{lk} is the membership degree of l^{th} data sample in k^{th} cluster in final BS. The task of FCC and FCC utility function is equivalent to the task of minimizing J_{pf} [53].

$$\max_{\boldsymbol{\mu}} \sum_{i=1}^{r} \omega_i U_f(\boldsymbol{\mu}, \boldsymbol{\mu}^{(i)}; p) \Longleftrightarrow \min_{\boldsymbol{\mu}, \boldsymbol{v}} J_{pf}(\boldsymbol{\mu}, V; \prod). \tag{2.27}$$

With the concatenated basic segments and cluster similarity matrix, to minimize J_{pf} is to take a FCM clustering on concatenated basic segments Y, with the k^{th} cluster similarity matrix v_k; $\forall k$, and the distance from y_l to v_k is defined as follows:

$$d(y_l, v_k) = \sum_{i=1}^{r} \omega_i f(\mu_{l\cdot}^{(i)}, v_k^{(i)}); \quad \forall\, l, k \tag{2.28}$$

Based on the above analysis, a FCM can formulate to minimize J_{pf}. Without loss of generality, the μ_{lk} and $v_k^{(i)}$ is defined as follows:

$$\mu_{lk} = \frac{d(y_l, v_k)^{-1/(p-1)}}{\sum_{k'=1}^{K} d(y_l, v_{k'})^{-1/(p-1)}}; \quad \forall\, l, k \tag{2.29}$$

and

$$v_k^{(i)} = \sum_{l=1}^{n} \frac{(\mu_{lk})^p}{\|\mu_{\cdot k}^p\|_1} \mu_{l\cdot}^{(i)}; \quad \forall\, k, i \tag{2.30}$$

Algorithm 2.3 FCC

1: Create the concatenated basic segments Y from $\prod = \{\mu_{(1)}, ..., \mu_{(r)}\}$;.
2: Set the weights $\{\omega_i\}_{i=1}^{r}$ and derive f using Eq. (2.25).
3: Call weighted piece-wise FCM algorithm.

Algorithm 2.3 and ***Algorithm 2.4*** summarizes the steps of FCC and weighted piece-wise FCM [53], where $\mu = [\mu_{lk}]_{n \times c} = F(V)$ and $V = \{v_k\}_{k=1}^{c} = H(\mu)$, the F and H are two mapping function defined by Eq. (2.29) and Eq. (2.30).

Algorithm 2.4 weighted piece-wise FCM

1: $l \leftarrow 0$;
2: **Initialize** $\mu_{[l]}$, and let $v_{[l]} = H(\mu_{[l]})$;
3: repeat
4: $l \leftarrow l + 1$.
5: Let $\mu_{[l]} = F(v_{[l-1]})$ using Eq. (2.29).
6: Let $v_{[l]} = H(\mu_{[l]})$ using Eq. (2.30).
7: Until some stopping criterion met.

To meet Big Data challenge, we further parallelize FCC on the Apache Spark platform. For FCC, it is difficult to obtain BSs on large high-dimensional data, which also brings great trouble to the subsequent consensus clustering. As mentioned in the objectives, Section 1.2, it is proposed to design scalable clustering algorithms for handling Big Data. A detailed description of Big Data frameworks is presented in the next section.

2.2 Big Data Frameworks

As we enter into the information age, data are being generated by a variety of sources other than people and servers, such as sensors embedded into phones and wearable devices, video surveillance cameras, MRI scanners, and set-top boxes. IDC[1] has released a report on the ever-growing datasphere and reported that it will grow 175 zettabytes (ZB) by 2025. Nearly 30% of the world's data will need real-time processing. The amount of data in the world was estimated to be 44 ZB at the dawn of 2020 [126]. Google, Facebook, Microsoft, and Amazon store at least 1,200 petabytes of information. The world spends almost $1 million per minute on commodities on the Internet. Electronic Arts process roughly 50 terabytes of data every day. By 2025, there would be 75 billion Internet-of-Things (IoT) devices in the world. By 2030, nine out of every ten people aged six and above would be digitally active [127]. Nowadays, endless advanced information is gathered at an increasing rate in various

[1] `https://www.idc.com/getdoc.jsp?containerId=prUS47560321`

fields [60, 67, 64, 69]. Due to the rising volume of Big Data from different sources, there is a need of genuine research with a proper investigation in Big Data analytic to pick up experiences from the valuable information in Big Data.

In the bioinformatics domain also the data is generated at a great pace. Big data sources are no longer limited to particle physics experiments or search-engine logs and indexes. With the digitization of all processes and high throughput devices at lower costs, data volume is rising everywhere, including the bioinformatics domain. For instance, the size of a single sequenced human genome is approximately 200 gigabytes. Biologists no longer use traditional laboratories to discover a novel biomarker for disease, rather they rely on huge and continuously growing genomic data made available by various research groups. Technologies for capturing biological data are becoming cheaper and more effective, such as automated genome sequencers, clustering of genome sequencing gives rise to this new era of Big Data in bioinformatics. Clustering is one of the most widely used data mining methods for bioinformatics genome data investigation. In genome data investigation, the surging volume of genome data has put colossal weight on clustering algorithms to scale beyond a single machine due to both space and time bottlenecks. To scale the clustering algorithms for huge genome data, there is a requirement for Big Data handling systems. Recently, incalculable handling frameworks have been designed precisely for the utilization of Big Data [17, 18, 19, 20]. Table 2.1 shows the summarization of some features of Big Data

Table 2.1: Summarization of some features of Big Data frameworks.

Framework	**Programming Model**	**Data Storage**	**Data Type**	**Processing Mode**	**Programming Language**
Hadoop	Map-Reduce	HDFS	Key-Value Pair	Batch	C/C++/Java/ Python/Perl/Ruby
Spark	Transformation and Action	HDFS/ DBMS	Key-Value Pair/RDD	Batch/ Real-Time	Java/Python /Scala/R
Flink	Topology	HDFS/ Hbase	Key-Value Pair	Batch/ Real-Time	Java/Python /Scala/R
Storm	Transformation	File/ Stream	Key-Value Pair	Real-Time	Java/Clojure
Samza	Map-Reduce	File	Events	Real-Time	Java/Scala

frameworks. There are a wide variety of processing frameworks available for Big Data such as Apache Hadoop [128], Map Reduce [129], Apache Flink [130], Apache Mahout [131], Apache Haloop [132], Apache Storm [133], Apache Samza [134], and Apache Spark [135, 136, 137].

Apache Spark is an outstanding, universally useful, distributed Big Data framework that keeps the advantages of MapReduce scalable and making it more adaptable. Veiga [138] directed an exploratory examination on Spark, Hadoop, and Flink. They built up the asset usage for both MapReduce [139] and Spark. Also, they detailed the fragment of execution of jobs. Even though it is adaptable to the non-critical failure of MapReduce, it has a multistage in-memory programming model. With such a propelled model, Apache Spark is a lot quicker and simpler to utilize. Spark is built on Hadoop's data volume model, i.e., Hadoop Distributed File System (HDFS). Apache Spark is perfect for deploying an application with a MapReduce technique. It was initially developed at the University of California in 2009. Spark performs up to 100 times faster than Hadoop MapReduce and significantly faster than other frameworks.

As mentioned in the objectives, Section 1.2, it is proposed to design the scalable clustering algorithms and scalable feature extraction techniques for preprocessing of SNP and protein sequences, which aim to utilize the Apache Spark framework for the processing of Big Data. Therefore, a detailed description of the Apache Spark framework is presented next.

Working of Apache Spark

Apache Spark is a scalable in-memory computation framework for Big Data processing. It allows subsets of the dataset to be processed in parallel across a cluster. Apache Spark is a high-speed cluster computing system with efficient and straightforward development APIs which allow worker nodes to access a dataset iteratively and execute efficiently. The in-memory cluster computing technique of spark increases the processing speed of an application by implementing a spark job on the Hadoop framework to share a cluster and dataset while satisfying consistent levels of service and response. Apache Spark works with Hadoop to access data from spark engines

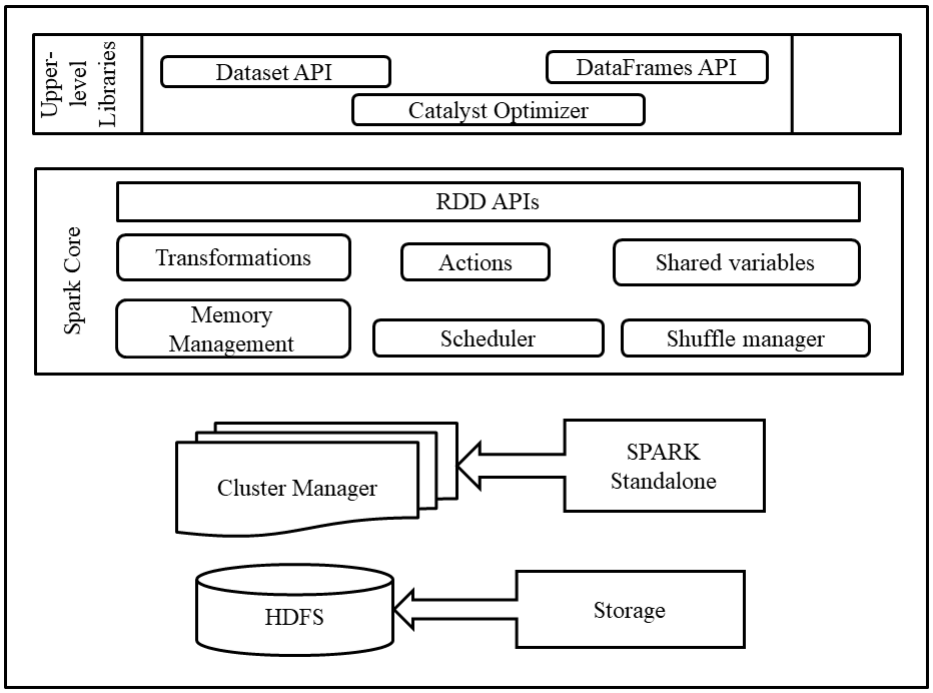


Figure 2.4: Apache Spark cluster stack.

[140]. Spark builds with a stack of libraries, including SQL and Data Frames, MLlib, GraphX, and Spark Streaming. MLlib is a library that provides a machine learning algorithm for data science techniques. Figure 2.4 shows the Apache Spark cluster stack used in our experiment and the details for the same are explained below.

- Spark core: The Spark core runs on different cluster managers and can access data on any Hadoop data source. It provides a simple programming interface for large-scale processing datasets, Resilient Distributed Dataset (RDD). Spark core is embedded in Scala, but it comes with APIs in Scala, Java, Python, and R (Python in our case). Besides, Spark core provides a key function for in-memory cluster computing, including memory management, job scheduling, data shuffling, and error recovery [141].

- Upper-level libraries: Spark SQL has been created to manage various workloads on the Spark core.

- Cluster managers and data source: Cluster manager is used to obtain cluster

resources to run a job. Spark Engine works with the built-in Spark cluster manager (i.e. standalone). Cluster manager manages resource sharing between Spark applications. On the other hand, Spark can access data from the HDFS [140].

- Resilient Distributed Datasets: Spark core is built on Resilient Distributed Dataset (RDD) abstraction. Spark revolves around the concept of RDD, which is a fault-tolerant collection of elements that can be operated in parallel. There are two ways to create RDDs: parallelizing an existing collection in the driver program or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat [135]. Parallelized collections are created by calling SparkContext's parallelize method on an existing iterable or collection in the driver program. The elements of the collection are copied to form distributed dataset that can be operated in parallel. PySpark can create distributed datasets from any storage source supported by Hadoop, including local file system, HDFS, Cassandra, HBase, Amazon S3, etc. Spark supports text files, SequenceFiles, and any other Hadoop InputFormat. Text file RDDs can be created using SparkContext's textFile method. This method takes a URI for the file (either a local path on the machine or a hdfs://, s3a://, etc URI) and reads it as a collection of lines.

  Two types of operations that can be performed on an RDD:

  1 Transformations: Transformations are operations on an RDD that result into another RDD.

    - Map: A map is a transformation operation in Apache Spark. It applies to each element of RDD and returns the result as a new RDD. Spark Map function takes one element as input and processes it according to custom code (specified by the developer) and returns one element at a time. Map transforms an RDD of length into another RDD of length. The input and output RDDs will typically have the same number of records.

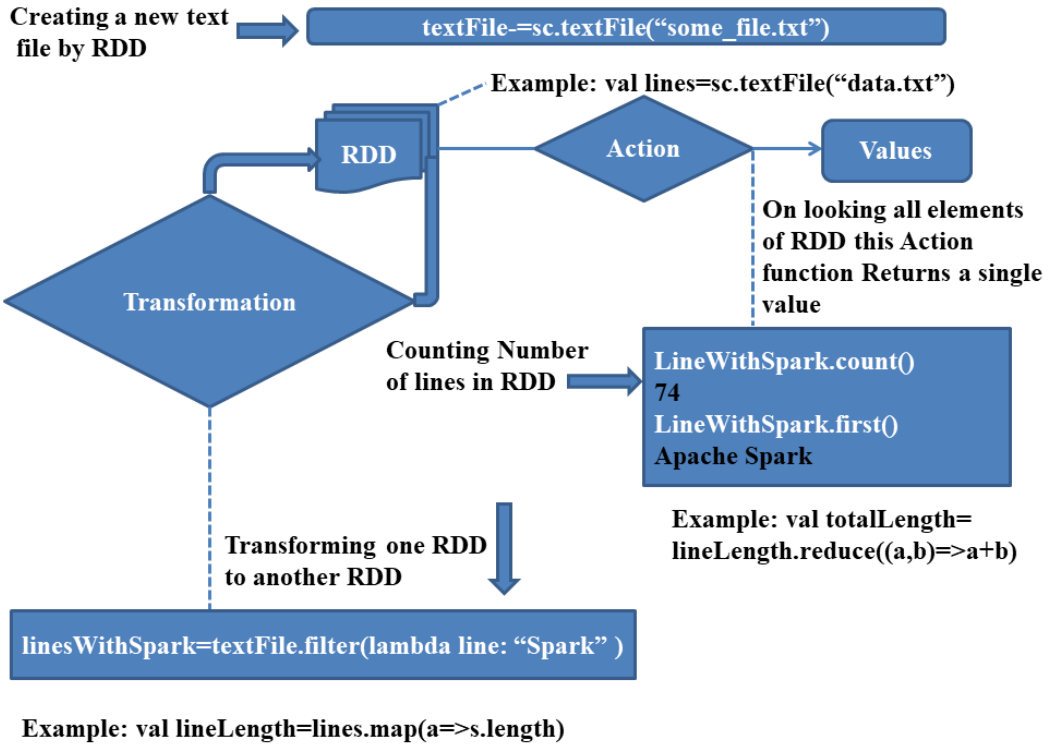


Figure 2.5: Workflow of Apache Spark operation.

– Aggregations: A dataset with key-value sets needs to aggregate statistics across all the elements with the same key. Spark encompasses a set of operations that combines values with the same key by employing a reduceByKey function. This work combines the values for each key utilizing an affiliated decrease work. It works only for RDDs that contain key and value pairs of elements (i.e. RDDs having tuple or Map as a data element). One associative function is passed as a parameter, which can be connected to the source RDD and will make a new RDD with resulting values (i.e. key-value sets). This function produces the same result when drearily utilized on the same set of RDD information multiple partitions irrespective of elements order. Figure 2.5 shows workflow of Apache Spark operation.

2 Actions: Returns a value to the driver program after running a computation on the dataset. Reduce is an action that aggregates all the elements of the

RDD using some function and returns the final result to the driver program (although there is also a parallel reduceByKey that returns a distributed dataset).

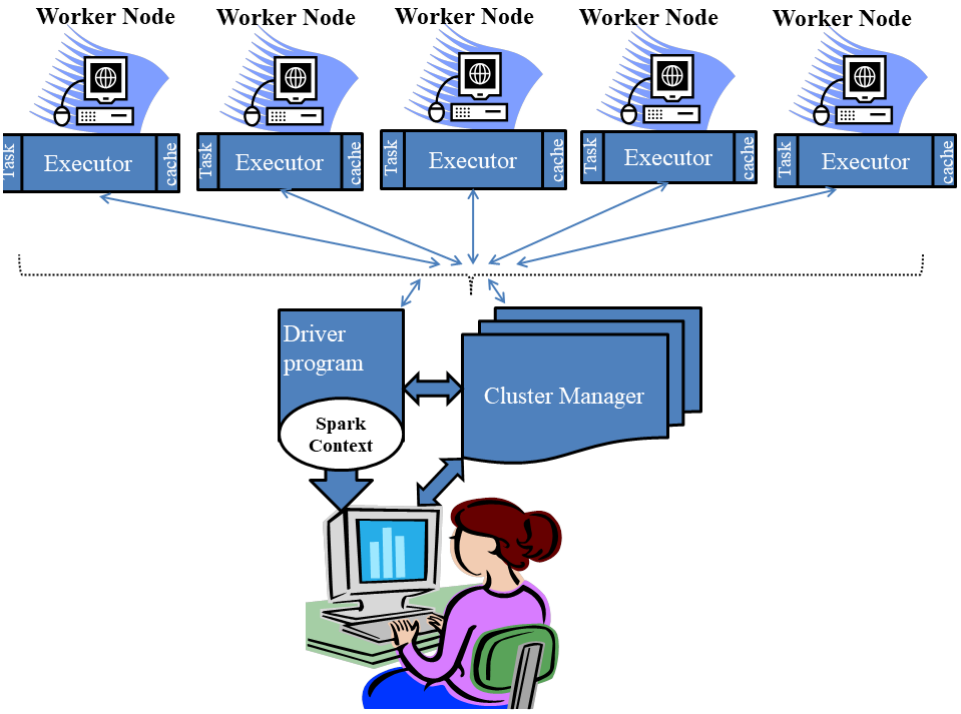


Figure 2.6: Apache Spark cluster Application.

- Spark-application: The Spark application execution consists of five main units, a controller program, a cluster administrator, workers, executors, and tasks. The Apache Spark cluster application is shown in Figure 2.6. A driver program is an application that uses Spark as a library and defines the high-level flow of control for the target calculation. While a worker provides CPU, memory, and storage resources to a Spark application, an executor is a JVM process that Spark creates on each worker for that application. A job is a set of calculations that the Spark controller performs on a cluster to get results in the program. A Spark application can start multiple jobs. Spark divides the work into steps of a Directed Acyclic Graph (DAG), where each phase is a collection of tasks. A task is the smallest unit of work that a spark sends to an executor. The main entry

point for spark functions is a spark context through which the driver program uses Spark. A Spark context represents a connection to a computing cluster.

With the new advancement of the Big Data environment, there is a need to scale clustering algorithms on extensive data computing frameworks to accomplish high performance without influencing clustering. Hence, we present the scalable fuzzy clustering algorithms executed in the Apache Spark framework to handle Big Data. A detailed description of scalable fuzzy clustering algorithms are presented in the next section.

2.3 Scalable Fuzzy Clustering Algorithms for Handling Big Data

Recently, a wide variety of algorithms [142, 143, 144, 145, 145] has been proposed by researchers for processing Big Data using various frameworks. This is because the enlarging volume of information emerging by the progress of technology makes the clustering of Big Data a challenging task. Kwok et al. [146], proposed a parallel version of the FCM algorithm for clustering. Beringer et al. [147], developed a scalable online version of the Fuzzy C-Means algorithm. Zhao et al. [148], proposed a parallel K-means clustering algorithm based on MapReduce, which is a simple yet powerful parallel programming technique. Zhang et al. [142], proposed i^2MapReduce, a novel MapReduce-based framework for incremental Big Data processing. Due to the rapid advancement of clustering algorithms, these techniques have received a lesser acknowledgment in Big Data issues, and this is because these algorithms are not scalable [149]. This confines many clustering algorithms from being used at enormous information scales. The Big Data frameworks [150] are needed to design scalable algorithms for handling Big Data generated from various sources.

In this section, we are presenting a Scalable Random Sampling with Iterative Optimization Fuzzy C-Means (SRSIO-FCM) algorithm. It is designed to deal with the challenges associated with fuzzy clustering for handling Big Data. The SLFCM

is a scalable version of the LFCM implemented on the Apache Spark cluster. The SLFCM is an integral part of the SRSIO-FCM algorithm. The details of the SLFCM and SRSIO-FCM algorithms are presented next.

2.3.1 Scalable Version of LFCM Algorithm

The SLFCM clustering algorithm is executed on Apache Spark to handle Big Data. In SLFCM [151], the computation of membership knowledge is evaluated in parallel on slave nodes. Thus it overcomes time complexity as compared to linear execution of an algorithm on a standalone machine. This process is continued until the difference observed is not useful for the benefit of cluster centers. In this, each data sample is reserved in the form of an array of features in RDDs [152], which is a data structure to store objects precisely in memory. The run-time complexity of the SLFCM algorithm is $O(ncdt/w)$, where w is the number of slave nodes. ***Algorithm 2.5*** summarizes the steps of SLFCM.

Algorithm 2.5 SLFCM to Iteratively Minimize $J_p(M, V')$

Input: X, c, p, ϵ, (initial V); X is an array of data samples such that $X = \{x_1, x_2, ...x_n\}$.

Output: I', V'

1: If V is not initialized, randomly initialize $V = \{v_1, v_2, ...v_c\}$.

2: **Compute** membership knowledge.

3: $I' = X.Map(V).ReduceByKey()$

4: **Compute** cluster centers.

5: $v'_j = \frac{\sum_{i=1}^{s} m_{ij}^p x_i}{\sum_{i=1}^{s} m_{ij}^p}, \forall j.$

6: If $\| V' - V \| < \epsilon$ then stop, otherwise go to step 2.

7: **Return** I', V'

In ***Algorithm 2.5***, the membership degree of all the data samples is combined at the master node, which is used in Line 3 of ***Algorithm 2.5*** to update the cluster center v_j to evaluate the value of parameters present in the numerator and denominator of v_j. The SLFCM is an integral part of the SRSIO-FCM algorithm which is discussed

in the next section.

2.3.2 Scalable Version of Random Sampling with Iterative Optimization Fuzzy C-Means Algorithm

The extension of FCM is a random sampling plus extension Fuzzy C-Means (rseFCM), that is used for handling Big Data [153], but the overlapping of the cluster is the main issue in rseFCM. The overlapping is removed by Random Sampling Iterative Optimization Fuzzy C-Means (RSIO-FCM) [154]. However, RSIO-FCM suffers from a sudden rise in several iterations during clustering. To overcome the issues of RSIO-FCM, a Scalable RSIO-FCM termed as SRSIO-FCM [151] has been developed, which is an incremental fuzzy clustering approach. The SRSIO-FCM algorithm [151], is executed on Apache Spark. It divides the huge data into various subsets (or chunks). Thus, SRSIO-FCM partitions the set X into s subsets such that $X = \{X_1, X_2, ..., X_s\}$ where X_1 represents the first subset consists of random n/s samples. ***Algorithm 2.6*** summarizes the steps of SRSIO-FCM.

Algorithm 2.6 SRSIO-FCM to Iteratively Minimize $J_p(M, V^{'})$

Input: X, c, p, ϵ; X is an array of data samples such that $X = \{x_1, x_2, ...x_n\}$.

Output: $I^{'}, V^{'}$.

1: **Partition** set X into s subsets such that $X = \{X_1, X_2, ..., X_s\}$.
2: **Randomly** select X_1 from X without replacement where X_1 represents the first subset consist of random n/s samples.
3: $I^{'}, V^{'} = SLFCM(X_1, c, p, \epsilon)$
4: **for** $t = 2$ to n **do**
5: $I, V' = SLFCM(X_t, c, p, \epsilon, V')$
6: **Merge** the partition of all processed subsets
7: $I' = I' \cup I$
8: **Compute** updated cluster center v'_j using I'
9: **end for**
10: **Compute** the objective function using Eq. (2.6).
11: **Return** I', V'

The cluster centers and membership matrix of the first subset are obtained through the usage of SLFCM. The cluster centers obtained from the first subset is fed to the second subset for clustering. The membership matrices obtained from both subsets are combined, and then the cluster centers are fed as input to the third subset. This procedure is repeated for the clustering of all the subsequent subsets. The SRSIO-FCM helps in reducing the run-time of a grouping of huge data without negotiating the clustering quality. The SRSIO-FCM eliminates the need for storing the membership matrix, which makes the execution of the SRSIO-FCM algorithm much faster by decreasing the run-time. Since each subset is handled steadily, the time complexity is $O(ncdt/w)$ where w represents the number of slave nodes in a Spark cluster and d represents the dimensions of data samples. While the space complexity remains $O(ncd/s)$ because data corresponding to one subset is not held in the memory while processing the next subset. The 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.

The SRSIO-FCM improves the quality of the formed clusters over the RSIO-FCM algorithm, but it is not used to cluster the data having non-linear separable data distribution in the feature space. This means that it is not suitable for the clustering of non-linear data. The reason is that the SRSIO-FCM algorithm was developed from the RSIO-FCM algorithm, which is used to cluster linear separable data. Therefore, the drawback still lies in the SRSIO-FCM algorithm. The focal point of the SRSIO-FCM and SLFCM algorithm is promoting the clustering or centroid calculation of data having linear separable data distribution. Thus, this algorithm does not focus on the clustering of data with non-linear separable data distribution. Therefore, the kernel function is applied to achieve better mapping for non-linear separable datasets. The kernelized version of SLFCM and SRSIO-FCM is presented in Chapter 3.

Big Data is evolving in various domains in the current scenario, so it is essential to investigate the applicability and performance of scalable Big Data algorithms on a real-life problem. As discussed in the objectives Section 1.2, it is proposed to study the applicability of scalable clustering algorithms on real-life genome data, i.e., the massive SNP and protein sequences of soybean, rice, and SARS-CoV-2. Therefore, there is a need to propose a scalable feature extraction approach for SNP and protein sequence for handling huge genome data such that the features extracted from genome data can be fed to scalable clustering algorithms for the analysis of huge genome data. Thus, now a survey related to feature extraction approaches is presented next.

2.4 Survey on Genome Sequences

Bioinformatics has been an active area of research for the last three decades, and it is continuously gaining careful attention from computer scientists and the biologist's research community. The objective of bioinformatics is to store and manage biological data and to develop sophisticated computational tools that are helpful in analysis and modeling [155]. Since the field of bioinformatics, i.e., genomics stepped into the clustering of the high-dimensional data, this raises the requirement of developing the machine learning algorithms to handle the genomics data [44, 45, 46]. With ongoing advances in genomics, the clustering of high-dimensional genome sequences can be noticed all over the place [156]. Yet, the developing requirement for storage and handling massive genome data impose complex challenges to biologists [157]. Different clustering algorithms have been designed to cluster similar genes more accurately, as indicated by identical gene sequences [158]. Recently, genome sequences and SNP datasets have become incredible both in volume and complexity [159]. Subsequently, scalable clustering algorithms are expected to deal with them [160, 151]. The analytic breakthroughs brought an impressive and remarkable generation of biological data, which was a dream some years ago, which includes sequencing of Protein, DNA, and SNP [161]. To analyze huge biological data, there is a need for efficient data storage, searching, analysis, and feature extraction methods [150, 138]. Genome sequencing

projects currently produce an enormous amount of new sequences and cause the rapid increase of genome sequences. This leads to several problems to cluster SNP and protein sequences using machine learning approaches. Accordingly, there is a need for an efficient feature extraction approach that extracts significant features. However, feature extraction of hundreds of millions of protein sequences is impractical using current algorithms because they are not scalable. There are so many valuable processing frameworks that have been designed for the use of huge data [138, 139, 162]. Apache Spark is a unique, generally helpful, distributed enormous information system that maintains the benefits of MapReduce versatile and making it more scalable [163]. Spark is built on top of the HDFS. Apache Spark is ideal for conveying an application with a MapReduce method. With the fast improvement of Next-Generation Sequencing (NGS) innovation, many genomic datasets have been created, representing a significant challenge to customary bioinformatics tools [164].

An important issue in applying any algorithm to genome sequence clustering is to represent the genome sequences in terms of feature vectors. The high dimensionality of genome data creates several crucial problems for researchers during the implementation of machine learning algorithms [165]. A suitable input representation (extraction of features) is necessary to categorize SNP and protein sequences correctly. This section discusses a few methodologies that forms the basis for our proposed algorithms and various steps involved in preprocessing SNP and protein sequences. The feature extraction approaches for SNP and protein sequences are discussed in the subsequent section.

2.4.1 Methods for Single Nucleotide Polymorphisms (SNPs) Sequences

SNP represents change at a single position in a DNA sequence in the population [166]. A DNA sequence is formed from A (adenine), T (thymine), G (guanine), and C (cytosine) called a nucleotide [167]. For instance, an SNP may change the nucleotide C with the nucleotide T in a DNA sequence. SNPs typically occur throughout a

person's DNA. They appear almost once in every 1,000 nucleotides on average, which means there are roughly 4 to 5 million SNPs in a person's genome. These variations may be unique or occur in many individuals; scientists have found more than 100 million SNPs in populations worldwide. Most commonly, these variations are found in the DNA between genes. They can act as biological markers, helping scientists locate genes that are associated with the disease. When SNPs occur within a gene or in a regulatory region near a gene, they may play a more direct role in disease by affecting its function. Each SNP sequence is a lengthy collection of nucleotides, which can be challenging to work around [168, 169]. Hence, the aim is to reduce the length of sequences and extract useful features in float values.

Liu et al. [170] developed an approach to extract a 12-dimensional numerical feature vector from a DNA sequence. We have applied similar approach on SNP data to extract a 12-dimensional numeric feature vector for each SNP sequence. In this work, we have built an Apache Spark cluster over which the SNP feature extraction approach is executed in parallel and gives 12 numeric features. The steps used for preprocessing of SNP data is stated as follows:

(i) The nucleotide A, T, G, and C content from the SNP sequence are selected as the first parameter in the feature vector extraction. The total length of A, T, G, and C defined as ℓ_A, ℓ_T, ℓ_G, and ℓ_C. These four integers stand for the numbers of nucleotide A, T, G, and C in the SNP sequence, respectively.

(ii) The second numerical parameter in the feature vector is the sum of distances of each nucleotide base to the first nucleotide. The total distance $\mathcal{T}_i$ is defined as follows:

$$\mathcal{T}_i = \sum_{j=1}^{\ell_i} t_j \tag{2.31}$$

where, $i = A, T, G, C$; t_j is the distance from the first nucleotide to the j^{th} nucleotide of i in the SNP sequence. The other four feature vectors of the total distances are denoted as $\mathcal{T}_A, \mathcal{T}_T, \mathcal{T}_G$, and $\mathcal{T}_C$.

(iii) The third parameter chosen for the feature vector extraction is the distribution

of each nucleotide along the SNP sequence. The variance of distance for each nucleotide utilized to characterize the distribution is defined as follows:

$$D_i = \sum_{j=1}^{\ell_i} \frac{(t_j - d_i)^2}{\ell_i} \tag{2.32}$$

where, $i = A, T, G, C$; t_j is the distance from the first nucleotide to the j^{th} nucleotide of i in the SNP sequence and $d_i = \frac{\mathcal{T}_i}{\ell_i}$. So, the feature vector, which contains 12-dimensional data, is given as follows:

$$\langle \ell_A, \mathcal{T}_A, D_A, \ell_T, \mathcal{T}_T, D_T, \ell_G, \mathcal{T}_G, D_G, \ell_C, \mathcal{T}_C, D_C \rangle$$

As mentioned in the objectives, Section 1.2, it is proposed to design the scalable SNP preprocessing for handling Big Data. The 12-dimensional numeric feature vector of the SNP sequence can be made scalable using the Apache Spark framework (as discussed in Section 2.2) for handling massive SNP data. The preprocessed 12-dimensional feature vectors can be used as input to the scalable fuzzy clustering algorithms to cluster huge SNP data.

Discovering clusters in the high-dimensional genomics data is extremely challenging for bioinformatics researchers for genome analysis. The Micro-array datasets are unpredictable and have inherent outliers or missing values [171]. The clustering of soybean and rice genome data and their analyzed results will help in setting a strong foundation for handling and analysis of subsequent large scale genome re-sequencing efforts in the future. Large scale sequencing of genome data has generated huge genomic, omics, and protein data [172, 173, 174]. Similarly, SNP datasets are also growing exponentially [159]. SNPs are becoming the most popular type of marker in linkage and association studies to discover genes associated with various traits such as diseases and drought resistance [166]. Therefore, faster methods for SNP clustering are required so that accurate data analysis can be done for the identification of genes associated with various traits.

Recently, fuzzy clustering approaches have been taken in consideration because of their capability to assign one gene to more than one cluster, which may allow capturing

genes involved in multiple transcriptional programs and biological processes [175, 176]. It is often in the SNP sequences that the number of dimensions of feature is a lot higher than the number of sequences [177, 178]. The fuzzy clustering algorithm divides the data into subsets such that each subset deal with a small size of data and it takes less number of iterations to converge faster and achieve good clustering quality results while dealing with huge data sizes [179].

2.4.2 Methods for Protein Sequences

The building blocks of proteins are amino acids, which are small organic molecules that consist of an alpha (central) carbon atom linked to an amino group, a carboxyl group, a hydrogen atom, and a variable component called a side chain. Proteins are built from a set of only twenty amino acids, each of which has a unique side chain [180]. The most important task while feeding a protein sequence in any machine learning algorithm is to encode the protein sequence into a feature vector and then apply any machine learning algorithm on the protein sequence [174]. A protein sequence contains characters from the 20-letter amino acid alphabets $\sum$={A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}. The protein sequences can be of any length, and the amino acids present in a sequence are arranged in any order. There is a substantial need for a machine-learning algorithm to analyze and model huge protein sequences [181]. An important issue in applying any algorithm to protein sequences is representing the protein sequences in feature vectors. The high dimensionality of protein data creates several crucial problems for researchers during the implementation of machine learning algorithms [165].

Wang [182], proposed a new technique for feature extraction, which tries to capture both the global similarity and local similarity of sequences. The global similarity refers to the overall similarity among multiple sequences, whereas the local similarity refers to frequently occurring sub-strings in the sequences. It considered that the 2-gram method computed the global similarity of sequences and adopted a 6-letter exchange group to represent a sequence [174, 183]. Then it uses a sequence mining tool to compute the local similarity. Another feature extraction approach is proposed by

Bandyopadhyay [184] that uses a 1-gram technique for feature encoding. The feature size comprises 20 amino acids. The extracted features are such that they take into consideration the probabilities of occurrences of the amino acids in the different positions of the sequences. Mansoori [185], proposed a new technique for feature extraction. To extract the relevant features from protein sequences, the features are counted as the occurrences of six exchange groups in each sequence [183]. Another feature extraction approach for protein sequence is proposed by Mansoori [181]. This method uses 2-grams and a 2-gram exchange group from the training and test data. In this approach, the distance-based feature ranking method was utilized to select the best and most appropriate features. Bharill [186] proposed a novel Co-occurrence based Probability Specific Feature (CPSF) approach, which represents each variable-length protein sequence with a fixed-length numeric vector. It considers all possible position-specific variations of amino acids in a protein sequence. Chou [187] suggested two types of models commonly used to present protein models: sequential model and descriptive model, but both the models have drawbacks. The sequential model fails to function when the query protein has no significant sequence similarity with any characteristic-known protein. The main drawback of the independent model is the missing effect of sequential order. Therefore, a separate model is proposed by Chou [188], known as the 'pseudo-amino acid composition' (PSeAAC) model. A suitable input representation (extraction of features) is necessary for the proper clustering of protein sequences. A detailed description of two methods for protein data preprocessing is discussed in this section. First, we provide the PSeAAC, a 60-dimensional feature vector, and then the CPSF, a 6-dimensional feature vector for protein sequences, is discussed.

Pseudo-Amino Acid Composition (PseAAC)

This section describes the detailed description of the 60-dimensional feature vector for protein sequences via the general form of pseudo amino acid composition (PseAAC model) [187, 189]. The general form of PseAAC for the P protein can be formulated as follows:

$$[P = P_1, P_2, ...P_\psi]^\top \tag{2.33}$$

The transpose operator is $\top$, and the subscript ψ is an integer, and its value and components $P_1, P_2...$ depends on how the desired information is extracted from the amino acid sequence. The details of the collection of characteristics of the protein sequence, a linear polymer of 20 amino acids, are described below.

The three parameters of each of these 20 amino acids contribute to the 60-dimensional vector. The vector's numerical construction is based on the following points:

(i) The content of each of the 20 amino acids.

(ii) The distance of each amino acid in sequence from the first amino acid sequence.

(iii) The distribution of each amino acid with the protein sequence.

The mathematical properties of protein sequencing have been analyzed separately.

(i) Amino acid count: The twenty amino acids for protein sequences are $\sum$={A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}. The total length of 20 amino acids defined as $\ell_A, \ell_C, \ell_D, \ell_E, \ell_F, \ell_G, \ell_H, \ell_I, \ell_K, \ell_L, \ell_M, \ell_N, \ell_P, \ell_Q, \ell_R, \ell_S, \ell_T, \ell_V, \ell_W, \ell_Y$.
These twenty integers stand for the numbers of amino acids $A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y$ in the protein sequence, respectively. The length of the protein sequence is the count of all the amino acids in the sequence. This parameter alone is not sufficient as the sequences containing the same number of amino acids in different locations are significantly different. Therefore, it contributes to the first 20 values in a 60-dimensional vector [189].

(ii) Total distance: The total distance is the summation of distances of each amino acid from the first amino acid in the protein sequence. These 20 values contribute to the other 20 values in the 60-dimensional vector. But this parameter will sometimes look the same for similar protein sequences. The total distance of $\mathcal{T}_i$ is defined as follows:

$$\mathcal{T}_i = \sum_{j=1}^{\ell_i} t_j \tag{2.34}$$

where, $i = A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y$; t_j is the distance from the first amino acid to the j^{th} amino acid of i in the protein sequence. The other twenty feature vectors of the total distances are denoted as $\mathcal{T}_A, \mathcal{T}_C, \mathcal{T}_D, \mathcal{T}_E, \mathcal{T}_F, \mathcal{T}_G, \mathcal{T}_H, \mathcal{T}_I, \mathcal{T}_K, \mathcal{T}_L, \mathcal{T}_M, \mathcal{T}_N, \mathcal{T}_P, \mathcal{T}_Q, \mathcal{T}_R, \mathcal{T}_S, \mathcal{T}_T, \mathcal{T}_V, \mathcal{T}_W, \mathcal{T}_Y$.

(iii) Distribution: We have seen two parameters above that cannot distinguish the similarity/dissimilarity of the two sequences correctly. Therefore, we have taken into account the third parameter D_i which represents the distribution of 20 amino acids throughout the protein sequence. The distribution of amino acids is different for two protein sequences, even if they have the same content and total spacing of 20 amino acids. Therefore, the 20 amino acid distribution contributes to one-third of the 60-dimensional vector. The distribution of each amino acid is calculated as follows:

$$D_i = \sum_{j=1}^{\ell_i} \frac{(t_j - d_i)^2}{\ell_i} \tag{2.35}$$

where, $i = A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y$; t_j is the distance from the first amino acid to the j^{th} amino acid of i in the protein sequence and $d_i = \frac{\mathcal{T}_i}{\ell_i}$. We can then say that all these three characteristics make up the 60-dimensional vector that characterizes the protein sequence. We have explicitly correlated the extracted features with the general form of Eq. (2.33) of PseAAC as mentioned above. The 60-dimensional equal-length vectors were constructed for unequal-length protein sequences. So, the feature vector, which contains 60-dimensional data is given as follows:

$$\langle \ell_A, \ell_C, \ell_D, \ell_E, \ell_F, \ell_G, \ell_H, \ell_I, \ell_K, ell_L, \ell_M, \ell_N, \ell_P, \ell_Q, \ell_R, \ell_S, \ell_T, \ell_V, \ell_W, \ell_Y, \mathcal{T}_A, \mathcal{T}_C,$$
$$\mathcal{T}_D, \mathcal{T}_E, \mathcal{T}_F, \mathcal{T}_G, \mathcal{T}_H, \mathcal{T}_I, \mathcal{T}_K, \mathcal{T}_L, \mathcal{T}_M, \mathcal{T}_N, \mathcal{T}_P, \mathcal{T}_Q, \mathcal{T}_R, \mathcal{T}_S, \mathcal{T}_T, \mathcal{T}_V, \mathcal{T}_W, \mathcal{T}_Y, D_A, D_C, D_D,$$
$$D_E, D_F, D_G, D_H, D_I, D_K, D_L, D_M, D_N, D_P, D_Q, D_R, D_S, D_T, D_V, D_W, D_Y \rangle$$

A detailed description of preprocessing of protein sequences using the CPSF approach is presented next.

Co-occurrence based Probability Specific Feature (CPSF) approach

This section contains a detailed description of the CPSF [186] approach. The

CPSF approach captures the protein sequence's statistical properties and the amino acid position information to generate a vector of fixed-length numerical features for each protein sequence. The CPSF approach extracts features from the protein dataset in three steps: Protein Sequence Encoding (PSE), Global Similarity Measures (GSM), and Local Similarity Measures (LSM). The first step encodes protein sequences, representing each protein sequence as exchange groups [183]. In the second stage, the overall GSM is calculated, considering the probability that each amino acid appears at a particular position concerning the total number of protein sequences present in a specific species. In the third stage, the LSM calculates each amino acid's weight concerning each protein sequence, taking into account the GSM. In this way, the CPSF approach represents each protein sequence by a fixed-length numeric feature vector consisting of only six dimensions.

Table 2.2: Structure of five protein sequences.

$sequence_1$	M	D	G	N	P	L	G	N
$sequence_2$	M	E	G	N	D	L	Q	N
$sequence_3$	T	N	Q	D	F	V	R	L
$sequence_4$	V	D	Q	N	P	V	E	L
$sequence_5$	T	E	G	N	P	I	R	N

Stage One: PSE

In the first step of the CPSF approach [186], each protein sequence is encoded and presented in terms of six exchange groups. According to Dayhoff and Schwartz [183], the amino acids in the protein sequence belongs to six exchange groups. This is because, within each exchange group, there is a high evolutionary similarity between these amino acids. Exchange groups are efficient amino acid equivalent classes, formally represented by $\{e_1, e_2, e_3, e_4, e_5, e_6\}$, where e_1=$\{H, R, K\}$, e_2=$\{D, E, N, Q\}$, e_3=$\{C\}$, e_4=$\{S, T, P, A, G\}$, e_5=$\{M, I, L, V\}$ and e_6=$\{F, Y, W\}$ [182]. The protein sequences belong to different species, and within each species, the protein sequences share some structural similarities. An example of five protein sequences is presented in Table 2.2.

The protein sequences are encoded into six exchange groups as follows: $sequence_1$,

Table 2.3: Encoded positional representation of amino acids.

Sequence	**Positions**							
	1	2	3	4	5	6	7	8
1	e_5	e_2	e_4	e_2	e_4	e_5	e_4	e_2
2	e_5	e_2	e_4	e_2	e_2	e_5	e_2	e_2
3	e_4	e_2	e_2	e_2	e_6	e_5	e_1	e_5
4	e_5	e_2	e_2	e_2	e_4	e_5	e_2	e_5
5	e_4	e_2	e_4	e_2	e_4	e_5	e_1	e_2

i.e. MDGNPLGN encoded as $\{M, L\} \in e_5$, $\{D, N\} \in e_2$, $\{P, G\} \in e_4$. Similarly, $sequence_2$, i.e. MEGNDLQN is encoded as $\{M, L\} \in e_5$, $\{D, E, N, Q\} \in e_2$, $\{G\} \in e_4$. In the same way $sequence_3$, i.e. TNQDFVRL is encoded as $\{R\} \in e_1$, $\{N, Q, D\} \in e_2$, $\{T\} \in e_4$, $\{V, L\} \in e_5$, $\{F\} \in e_6$. Similarly, $sequence_4$, i.e, VDQNPVEL is encoded as $\{D, Q, E, N\} \in e_2$, $\{P\} \in e_4$, $\{V, L\} \in e_5$. In the same manner $sequence_5$, i.e. TEGNPIRN is encoded as $\{R\} \in e_1$, $\{E, N\} \in e_2$, $\{P, T, G\} \in e_4$, $\{I\} \in e_5$. Table 2.3 displays the encoded position information of all the protein sequences are given in the structure data as shown in Table 2.2. Once the sequence of proteins is encoded by the exchange groups (as shown in Table 2.3), the total similarity between the encoded exchange groups is calculated. A detailed description of this subject is given next.

Stage Two: GSM

In the second step of the CPSF method, we calculate the GSM by estimating the instance probability of all exchange groups at each position relative to the total number of protein sequences in the species. The GSM is calculated as follows:

$$(Probability)_{ij} = (Instance)_{ij}/\rho \tag{2.36}$$

Where $(Probability)_{ij}$ denotes the probability of instance of the i^{th} exchange group at j^{th} position, $(instance)_{ij}$ represents the frequency at which the i^{th} exchange group appears at j^{th} position and ρ represents the total number of sequences in a particular species. The GSM is then calculated according to the exchange groups are shown in Table 2.3. In this table, exchange group e_5 appears in the first place three times out of five sequences, so the $(Probability)_{51}=\frac{3}{5}$. Similarly, the probability of other

exchange groups present in this table is also calculated. Thus, in Table 2.4, the values obtained after estimating the overall GSM corresponding to all the encoded sequences are presented in Table 2.3. After that, LSM is calculated, which determines the specific weight at each exchange group's position. A detailed description of LSM is given next.

Stage Three: LSM

In the third stage of the CPSF approach [186], the LSM is calculated, which determines the location-specific weight of each exchange group within the sequence considering the weight factors. These weight factors ultimately represent the numeric feature vectors for each protein sequence. The weight of each exchange group is calculated as follows:

$$(Weight)_i^{sequence_k} = \sum_{j=1}^{j'} (Probability)_{ij} \times (PW)_{ij}^{sequence_k} \tag{2.37}$$

where $(Weight)_i^{sequence_k}$ represents the weight of i^{th} exchange group corresponding to the k^{th} protein sequence, $(Probability)_{ij}$ denotes the probability of occurrence of the i^{th} exchange group at j^{th} position and $(PW)_{ij}^{sequence_k}$ is the positional weight assign to the i^{th} exchange group based on the presence of k^{th} protein sequence at j^{th} position. The weight of exchange groups, i.e., the first encoded protein sequence ($e_5e_2e_4e_2e_4e_5e_4e_2$) present in Table 2.3 is calculated as follows:

$$(Weight)_{e_2}^{sequence_1} = 1 \times 1 + 1 \times 1 + 0.6 \times 1 = 2.6$$
$$(Weight)_{e_4}^{sequence_1} = 0.6 \times 1 + 0.6 \times 1 + 0.2 \times 1 = 1.4$$
$$(Weight)_{e_5}^{sequence_1} = 0.6 \times 1 + 1 \times 1 = 1.6$$

The first encoded protein sequence present in Table 2.3 is composed of only three exchange groups e_2, e_4, and e_5, and the remaining exchange groups, i.e., e_1, e_3, and e_6 are absent in the first sequence ($sequence_1$). Due to the absence of exchange groups e_1, e_3, and e_6 in $sequence_1$, the positional weight ($(PW)_{ab}^{sequence_o}$) corresponding to these exchange groups is zero. On the contrary, the exchange group e_2 occurs three times in $sequence_1$ at positions 2, 4, and 8, respectively. Therefore, the positional weight assigned to the exchange group e_2 for each position is 1 which is multiplied by

Table 2.4: Global Similarity Measure of encoded protein sequences.

Exchange groups	**Positions**							
	1	2	3	4	5	6	7	8
e_1	0	0	0	0	0	0	0.4	0
e_2	0	1	0.4	1	0.2	0	0.4	0.6
e_3	0	0	0	0	0	0	0	0
e_4	0.4	0	0.6	0	0.6	0	0.2	0
e_5	0.6	0	0	0	0	1	0	0.4
e_6	0	0	0	0	0.2	0	0	0

Table 2.5: Representation of feature vector.

Sequence	e_1	e_2	e_3	e_4	e_5	e_6
1	0	2.6	0	1.4	1.6	0
2	0	3.2	0	0.6	1.6	0
3	0.4	2.4	0	0.4	1.4	0.2
4	0	2.8	0	0.6	2.0	0
5	0.4	2.6	0	1.6	1	0

the probability of occurrence of exchange group e_2 on these positions, i.e., 1, 1, and 0.6 reported in Table 2.4. Thus, the final weight of exchange group e_2 is determined by adding the product of positional weight and probability of occurrence of exchange group e_2 based on the presence in $sequence_1$. Similarly, the weight of exchange groups e_4 and e_5 are computed. Finally, the feature vector for $sequence_1$ is obtained as $\{(e_1, 0), (e_2, 2.6), (e_3, 0), (e_4, 1.4), (e_5, 1.6), (e_6, 0)\}$. Similarly, by using three stages of the CPSF approach, the feature vectors of all the protein sequences present in Table 2.2 are determined and reported in Table 2.5. The CPSF approach finally represent each protein sequence with a feature vector consists of only six dimensions numeric features.

The PseAAC and CPSF extraction methods discussed in this section can not handle huge protein sequences. Hence, the two approaches PseAAC and CPSF can be made scalable using the Apache Spark framework (As mentioned in the objectives, Section 1.2) for handling massive protein data. The obtained 60-dimensional and 6-dimensional numerical feature vectors are used as the input to the scalable fuzzy clustering algorithms for clustering of huge protein data.

The clustering of protein sequences aims to provide meaningful partitioning from

a huge protein dataset [46]. The protein sequences are arranged into clusters based on their similarity in protein sequences. During the most recent decade, expenses and throughput of protein sequencing have dropped two-fold every year, twice quicker than computational costs [190]. This tremendous advancement has brought about countless protein sequences and several billions of putative genes. Clustering protein sequences predicted from sequencing reads can impressively lessen the excess of sequence sets and expenses of downstream analysis and storage [190]. Han and Baker [191] have utilized the K-means clustering algorithm to investigate the protein sequence-to-structure relationship. The high-quality sequence clusters have been created using the K-means algorithm [191]. The K-means was utilized to understand how protein sequences correspond to local 3D protein structures [192]. However, the K-means algorithm calculates the distance between the data sample with exact precision. When this distance function is not well characterized, the K-means algorithm may not effectively reveal the sequence-to-structure relationship [193, 194]. As a result, some of the clusters provide a poor match between protein sequences.

Many researchers have worked with fuzzy clustering methods for protein sequence clustering [195, 196, 197]. The fuzzy clustering method applies to the items that cannot be completely divided into two different groups, and thus it is profoundly applicable for the biological things that have a slow development relationship and can not be partitioned into two particular clusters [198]. Zhang [195], developed Fuzzy C-Means (FCM) based method for predicting the structural class of a protein from its amino acid composition. Lu [196], applied the fuzzy clustering method to all plant cysteine-rich polycomb-like protein transcription factors. The feature vector of each protein sequence for the fuzzy clustering method is encoded by the short length peptides and the combination of functional domain models. Farhangi [197], developed a protein motif sequence clustering using the FCM algorithm based on the Hadoop framework. Because of the quick progression of clustering algorithms, these approaches received a lesser acknowledgment for handling Big Data issues. The reason behind this is that these algorithms are not scalable [149]. The Big Data environments [150] are needed to implement scalable algorithms for dealing with large information produced from

different sources.

The huge SNP and protein data are applied to the proposed scalable clustering algorithms (as mentioned in the objectives Section 1.2) for their clustering. Further, validation measures are used to evaluate the performance of scalable clustering algorithms without any class information [199]. The validation measures include external measures such as Normalized Mutual Information (NMI) [200], Adjusted Rand Index (ARI) [201], F-score [193], and internal measures such as Silhouette Index [202], and Davies-Bouldin Index (DBI) [203]. We have used SI and DBI for assessing the performance of scalable clustering algorithms for SNP and protein sequences. The detail of these measures is presented in the subsequent section.

2.5 Performance Measures

This section presents the different performance measures to evaluate the performance of scalable clustering algorithms. There are various measures available for the validation of clustering algorithms [199, 204, 205]. One option is to use external validation measures for which a priori knowledge of dataset information is required, but it is hard to say if they can be used in real problems (usually, real problems do not have prior information of the dataset in question). Another option is to use internal validity measures, which do not require a priori information from the dataset. Hence, we can use internal measures for the validation of genome data. The details of external and internal measures used in our experimentation are presented next.

2.5.1 External Measures

Here, we discuss the measures used to evaluate the performance of the proposed scalable fuzzy clustering algorithms on Big Data which are discussed as follows:

Normalized Mutual Information (NMI)

The NMI [200], is utilized for the assessment of clustering quality, which estimates the proportion of the common data for the clustering, ground truth, and their harmonic

mean. The NMI is characterized as pursues:

$$NMI = \frac{\sum_{c=1}^{i}\sum_{q=1}^{j} n_c^q \log\left(\frac{n.n_c^q}{n_c.n_q}\right)}{\sqrt{\left(\sum_{c=1}^{i} n_c \log\left(\frac{n_c}{n}\right)\right)\left(\sum_{q=1}^{j} n_q \log\left(\frac{n_q}{n}\right)\right)}} \tag{2.38}$$

Where, n denotes the total number of data samples, n_c and n_q are the data samples in the c^{th} cluster and the q^{th} class, respectively, and n_c^q is the number of common data samples in class q and cluster c.

Adjusted Rand Index (ARI)

The ARI [201], is utilized to discover the likeness between the clustering of two datasets. It is the corrected-for-chance version of the Rand index, which evaluates the similarity between two partitions. The ARI measure assumes that the clustering is discrete, i.e., hard [206]. To compute the ARI, in the case of fuzzy clustering we first harden the fuzzy partitions by setting the highest membership value of each data sample to the cluster equal to 1, and all else to 0 [32]. We use ARI to compare the clustering solutions with ground truth labels (when available), as well as it examines the partition of an accelerated algorithm to that of the reference algorithm. The formulation of ARI is defined as follows:

$$ARI = \frac{\sum_{q,c}\binom{n_{qc}}{2} - \left[\sum_q \binom{n_{q.}}{2}\sum_c \binom{n_{.c}}{2}\right]/\binom{n}{2}}{\frac{1}{2}\left[\sum_q \binom{n_{q.}}{2} + \sum_c \binom{n_{.c}}{2}\right] - \left[\sum_q \binom{n_{q.}}{2}\sum_c \binom{n_{.c}}{2}\right]/\binom{n}{2}} \tag{2.39}$$

F-score

F-score is used to calculate the accuracy of a clustering output [193]. The precision and recall of the cluster for each given class are computed as follows:

$$\mathcal{P}_{qc} = \frac{n_{qc}}{n_c}, \mathcal{R}_{qc} = \frac{n_{qc}}{n_q} \tag{2.40}$$

Where, n_{qc} denotes the number of samples of class q that are also present in cluster c, n_q denotes the number of samples belonging to class q, and n_c denotes the number of samples belongs to cluster c. The F-score of cluster c and class q is represented as

follows:

$$\varphi(q,c) = \frac{2 * \mathcal{P}_{qc} * \mathcal{R}_{qc}}{\mathcal{P}_{qc} + \mathcal{R}_{qc}} \tag{2.41}$$

The overall F-score is then defined as the weighted sum of the maximum F-scores for each class and is given by the following:

$$F - score = \sum_{q} \frac{n_q}{n} max\{\varphi(q,c)\} \tag{2.42}$$

where, n is the total number of data samples. The higher value of the F-score indicates better clustering results. The F-score value approaching 1 reflects that the attained clustering results are similar to the ground truth value.

2.5.2 Internal Measures

In this section, we are discussing the measures most widely used in genomics. We have used these measures to evaluate the performance of the proposed scalable clustering algorithms on genome data.

Silhouette Index (SI)

This measure is useful for the validation of consistency within clusters of genome data. SI [202] is a measure of how similar a data sample is to its cluster in comparison with other clusters. Thus SI is characterized as:

$$SI = \frac{a_2(i) - a_1(i)}{max[a_1(i), a_2(i)]} \tag{2.43}$$

Where $a_1(i)$ is the average distance between i^{th} sample from all other data samples within the same cluster, $a_2(i)$ is the lowest average distance of i^{th} sample to all the data samples in any other cluster, of which i is not a member. The Silhouette value is bounded in a range of -1 to 1. A negative value indicates low clustering, and a positive value indicates good clustering quality.

Davies Bouldin Index (DBI)

The DBI [203] is used for evaluating the performance of clustering. It consolidates a single record in two measures, one identified with the scattering of individual clusters

and the other to the partition between various clusters.

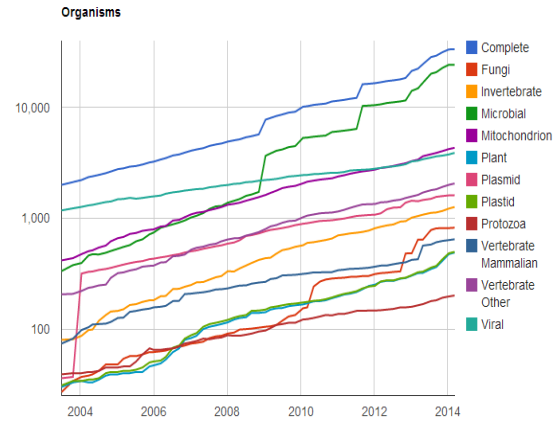
$$DBI = \frac{1}{c}\sum_{i=1}^{c} \max_{j \neq i} \left[\frac{\text{diam}(c_i) + \text{diam}(c_j)}{\text{dist}(c_i, c_j)} \right] \tag{2.44}$$

Where $\text{dist}(c_i, c_j)$ correlates to the distance between the center of clusters c_i and c_j, $\text{diam}(c_i)$ is the maximum distance between all the data samples of cluster c_i, and c is the number of clusters. The DBI is not limited inside a given range, and thus the lower DBI indicates good clustering quality.

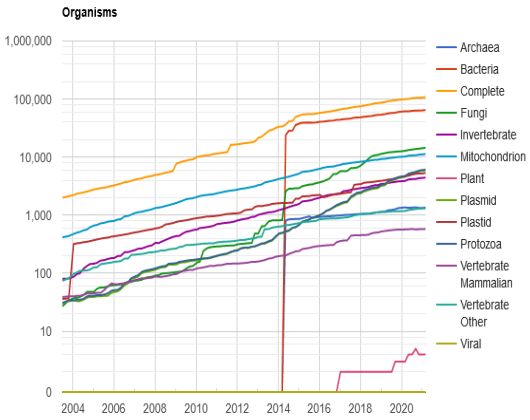
The SI and DBI measures evaluate the performance of scalable fuzzy clustering algorithms on huge genome data. The details of genome datasets used in our experimentation are presented in the next section.

2.6 Real-life Genome Data

Genomics is an interdisciplinary field of biology focusing on the structure, function, evolution, mapping, and editing of genomes [207, 208, 209]. A genome is an organism's complete set of genetic instructions. Each genome contains all of the information needed to build that organism and allow it to grow and develop. The role of technology in genomics focuses on the massive growth in genome sequencing, which has a development rate quicker than expected by Moore's law [210]. Scientists are expecting as many as 1 billion people to have their genomes sequenced by 2025. The amount of data being produced in genomics is doubling every seven months, so within the next decade, genomics is looking at generating somewhere between 2 and 40 exabytes a year [43]. Computer databases are becoming popular for organizing the vast amounts of biological data currently available and to make it easier for researchers to locate relevant information [211]. The numerous existing databases [21, 212, 56, 166, 54] such as National Center for Biotechnology Information (NCBI) [213], UniProtKB [214], and Protein Data Bank (PDB) [215] plays a vital role in a research environment and medical purposes. Genomic and proteomic databases such as NCBI and PDB are beneficial to know research history about the genome of any organism, protein func-



(a) Exponential Growth of NCBI Genomes [219].



(b) Growth statistics of organism [219]

Figure 2.7: The phenomenal growth of genome data in NCBI is challenging to manage, and continues unabated.

tion, proteome nature, etc. The sheer volume of the raw sequence data present in these repositories has led to an attempt to reorganize this information into various kinds of smaller and specialized databases. Such databases include various genome browsers, model organism databases, molecule or process-specific databases, and others. To understand the growth of these resources, one needs to look at the annual database issue of the journal Nucleic Acids Research. GenBank is described as one of the first database issues, in which only a few dozen genomics databases are listed [216]. In contrast, the latest database issue describes over 1,000 genomics databases [217] and tools [218]. However, even this list of resources is only a part of the overall picture. Today, it appears that there are upwards of 3,000 distinct genomic resources, tools, and databases publicly available on the Internet. Figure 2.7(a) represents the number of database resources that organize and displays the data are also increasing rapidly. The rise is exponential in nature. The organism growth is represented using Figure 2.7(b). There are >800 databases of human genetic variation, but only a few central databases that are most widely used [220]. The largest database of common genetic variation is the NCBI's dbSNP[2], created after the Human Genome Project

[2]https://www.ncbi.nlm.nih.gov/snp/

discovered a significant number of common variants. Moreover, Chen [221] discussed many publicly available data repositories and resources for protein data.

We have performed comparative studies using whole genome re-sequence of soybean genotype $EC241780$ to identify genomic variations between the rust-resistant line $EC241780$ and susceptible cultivar $JS335$ and to develop suitable breeding strategies to impart rust resistance in soybean. Recent rapid developments of high-throughput sequencing technologies bring genome researchers to the age of Big Data, where the research paradigm has shifted from hypothesis-driven to data-driven. Big Data opens new avenues to study genomics and brings new challenges for bioinformatics to explore ways to efficiently manage and analyze data and eventually turn data into usable and actionable knowledge. The detailed description of the SNP and protein dataset used in our experimental analysis are presented in the subsequent section.

2.6.1 Soybean and Rice SNP Dataset Description

Each SNP represents a difference in a single DNA building block called a nucleotide. The detailed description of SNP is presented in Section 2.4.1. The SNP datasets used for our experimental analysis are discussed next.

SoySNP50K Wm82.a1: The SoySNP50K[3] iSelect BeadChip consists of 50,000 SNPs from soybean [222]. The subsequent size of the SoySNP50K Wm82.a1 data set is 1.7 GB. The complete dataset contains 20,081 SNP sequences.

SNP-seek rice: The SNP-seek rice data contains rice chromosomes (ch1-12)[4]; we have merged all the rice chromosomes from ch1-12 into a single file to perform clustering on a huge SNP dataset. Mansueto et al. [57] discussed the SNP-seek rice data in detail. The subsequent size of the dataset is 16.3 MB.

MAGIC-rice: The MAGIC-rice dataset[5] consists of SNP sequences, 1,411 Samples are divided into 12 files (for each chromosome). To perform clustering on a huge SNP

[3] `https://www.soybase.org/snps/`

[4] `https://s3-ap-southeast-1.amazonaws.com/oryzasnp-atcg-irri-org/osnp_legacy/diversity_rice31.oryzasnp.hapmap.tar.gz`

[5] `https://s3-ap-southeast-1.amazonaws.com/oryzasnp-atcg-irri-org/pub-data/MAGIC-Raw-genotype-data-Raghavan-2017.zip`

dataset, we have merged all the chromosomes from ch1-12 and formed the MAGIC-rice dataset. Bandillo et al. [223] discussed the detailed descriptions of the population. The subsequent size of the dataset is 1.05 GB.

248Entries rice: The 248Entries rice data[6] contains a total of 248 samples composed of indica and aus genotypes. Dilla-Ermita et al. [55] discussed the details of 248Entries rice data. The subsequent size of the dataset is 30.8 MB. The complete dataset contains 40,840 SNPs of rice data.

2.6.2 Soybean Protein Dataset Description

Proteins are the end products of the decoding process that starts with the information in cellular DNA. Each gene in cellular DNA contains the code for a unique protein structure. The detailed description of protein is presented in Section 2.4.2. The protein datasets used for our experimental analysis are discussed next.

Lee: The Lee strain, which crosses between the Chinese lines CNS and S-100, is widely used as a parent in many breeding projects in the southern United States and Brazil. Diversity is characterized by resistance to bacteria from Phytophthora rot, Peanut Mottle Virus, and bacterial pustule [224].

Williams82: Williams 82, a soybean cultivar used to generate the reference genomic sequence. This was obtained from reverse mating of the Phytophthora root rot resistance locus from the donor parent Kingwa to the recurrent Williams parent [225].

PI483463: *Glycine Soja* is the closest wild soybean of *Glycine max*. Species remain interfertile, and specimens of *G. soja* are used in breeding projects to introduce traits such as resistance to certain diseases or environmental stress. *Glycine Soja* accession PI483463 have salt tolerance [226].

W05: *Glycine Soja* accession W05 is a salt tolerant wild soybean whose genome sequenced and to serve as a reference genome assembly. W05 affiliation has been used for genetic studies of various traits including uncertainty, seed size, number of pods per plant, and seed color [227].

[6] `https://s3-ap-southeast-1.amazonaws.com/oryzasnp-atcg-irri-org/pub-data/248Entries_40840SNPs_inorder_21May2015_v2.zip`

The genome datasets discussed in this section are used for feature extraction and clustering. Clustering plant genome sequences can help to identify unique and new genes to improve crop production with higher yields, drought resistance, improved crop quality, and provide better suggestions to find a cluster of diseases. Moreover, due to the rapid spread of Coronavirus Disease-19 (COVID-19) caused by the Severe Acute Respiratory Syndrome Coronavirus-2 (SARS-CoV-2), the genome sequences of SARS-CoV-2 genome data are generated on a large scale, it is difficult for health professionals to keep up with new information on the virus. Therefore, we have investigated the SARS-CoV-2 data on the proposed scalable feature extraction technique, and then clustering of SARS-CoV-2 data is performed using proposed scalable clustering methods. The detailed description of SARS-CoV-2 data used in our experimentation is presented next.

2.6.3 SARS-CoV-2 Protein Dataset Description

COVID-19 presentation, which began with the reporting of unknown causes of pneumonia in Wuhan, Hubei province of China on December 31, 2019, has rapidly become a pandemic [228, 229, 230, 231]. The disease is named COVID-19, and the virus is termed SARS-CoV-2. Severe acute respiratory syndrome (SARS) coronavirus has caused severe respiratory disease and death in humans [232]. Nowadays, researchers working on different disciplines in many countries deal with the COVİD-19 virus intensely. Coronaviruses belong to the subfamily Orthocoronavirinae in the family Coronaviridae, Order Nidovirales. There are four genera within the subfamily Orthocoronavirinae, namely alphacoronavirus, betacoronavirus, gammacoronavirus, and deltacoronavirus. This section presents the description of SARS-CoV-2 protein datasets used in our experimental analysis to cluster massive SARS-CoV-2 protein sequences.

SARS: The SARS[7] dataset is a severe acute respiratory syndrome-related coronavirus obtained from NCBI. Severe Acute Respiratory Syndrome Coronavirus-2 (SARS-CoV-2) is a single-stranded, enveloped RNA virus and the etiological agent of the current

coronavirus disease 2019 pandemic. Efficient replication of the virus relies on the activity of nonstructural protein 1 (Nsp1), a major virulence factor shown to facilitate suppression of host gene expression through the promotion of host mRNA degradation and interaction with the 40S ribosomal subunit.

Coronaviridae: The Coronaviridae[7] dataset is obtained from NCBI. The NCBI Datasets Project has developed virus-specific genome and protein datasets, initially limited to the Coronaviridae (NCBI Taxonomy ID: 11118) family of viruses, in response to the COVID-19 pandemic. FASTA amino acid sequence file containing RefSeq and GenBank protein sequences, including polyproteins, mature peptides processed from polyproteins and other proteins.

P0DTD1: The P0DTD1[8] dataset is a multifunctional protein involved in the transcription and replication of viral RNAs. It contains the proteinases responsible for the cleavages of the polyprotein. Biological function: Methyltransferase that mediates mRNA cap 2'-O-ribose methylation to the 5'-cap structure of viral mRNAs. N7-methyl guanosine cap is a prerequisite for binding of nsp16. Therefore plays an essential role in viral mRNAs cap methylation which is essential to evade the immune system.

[7] `https://www.ncbi.nlm.nih.gov/datasetscoronavirus/genomes/`

[8] `https://www.ebi.ac.uk/pdbe/covid-19`

Chapter 3

Scalable Kernelized Fuzzy Clustering Algorithms for Handling Big Data

In this chapter, we present an Apache Spark cluster-based kernelized clustering algorithm named Kernelized Scalable Random Sampling with Iterative Optimization Fuzzy C-Means (KSRSIO-FCM). This is based on the Kernelized Scalable Literal Fuzzy C-Means (KSLFCM) clustering algorithm, in which kernel function is used. This proposed work is inspired by a Scalable Random Sampling with Iterative Optimization Fuzzy C-Means (SRSIO-FCM) algorithm [52]. The SRSIO-FCM improves the quality of the formed clusters over the RSIO-FCM algorithm, but it is unable to cluster the data having non-linear separable data distribution in the feature space. To tackle this issue, a novel algorithm KSRSIO-FCM is proposed. A kernel function is applied to achieve better mapping for non-linear separable datasets. Data is projected into the higher dimensional space so a hyperplane can easily separate it. Our proposed KSRSIO-FCM algorithm can handle non-linear separable data using the Radial Basis Function (RBF), which helps in improving clustering results. The details of the proposed methods are discussed next.

3.1 Introduction

Data has become an integral part of our life. There are 2.5 quintillion bytes of data at the current pace generated every day. Many types of software are being designed

to handle the voluminous data. Such an immense quantity of information containing valuable data is called Big Data. It is becoming necessary to mine such Big Data to gain insights into the valuable information that can be of great use in scientific and business applications. Presently, there are several fuzzy clustering algorithms such as the extension of FCM known as random sampling plus extension Fuzzy C-Means (rseFCM) that are used for handling Big Data [153], but the overlapping of the cluster is the main issue of rseFCM. The overlapping is removed by Random Sampling with Iterative Optimization Fuzzy C-Means (RSIO-FCM) algorithm [154]. However, RSIO-FCM suffers from a sudden rise in several iterations during clustering. To overcome the issues of RSIO-FCM, a Scalable RSIO-FCM termed as SRSIO-FCM [151] has been developed, which is an incremental fuzzy clustering approach. The SRSIO-FCM uses Scalable Literal Fuzzy C-Means (SLFCM) [151] algorithm for the computation of membership knowledge and cluster centers for all subsets. The SRSIO-FCM improves the quality of the formed clusters over the RSIO-FCM algorithm, but it is not used to cluster the data having non-linear separable data distribution in the feature space. The reason is that the SRSIO-FCM algorithm was developed from the RSIO-FCM algorithm, which is used to cluster linear separable data. Therefore, the drawback still lies in the SRSIO-FCM algorithm. The focal point of all these algorithms is on promoting the clustering or centroid calculation of data having linear separable data distribution. Thus, this algorithm does not focus on the clustering of data with non-linear separable data distribution. Therefore, the kernel function is applied to achieve better mapping for non-linear separable datasets. Our developed KSRSIO-FCM algorithm aims to overcome this drawback by applying the RBF. The RBF kernel is one of the most widely used kernels due to its similarity to the Gaussian distribution [38]. Thus, it helps in enhancing the KSRSIO-FCM clustering results. For this reason, we present the design of scalable kernelized fuzzy clustering algorithms implemented in the Apache Spark framework for handling Big Data.

3.2 Proposed Kernelized Scalable Fuzzy Clustering Algorithms for Handling Big Data

The proposed novel KSRSIO-FCM algorithm is designed to deal with the challenges associated with fuzzy clustering for handling Big Data and overcomes the drawbacks of the SRSIO-FCM algorithm as mentioned in Section 2.3.2. The proposed KSRSIO-FCM approach starts by randomly partitioning the data into various subsets. The KSRSIO-FCM initialized the cluster centers randomly for clustering of the first subset. After clustering the first subset, the cluster centers and membership knowledge corresponding to the first subset is obtained. After clustering the first subset, the final cluster centers are used as an input for clustering the second subset. After clustering of the second subsets, the cluster centers, and membership knowledge is obtained. Then, it combines the membership knowledge of the first and second subset to compute the new cluster centers. These cluster centers are then fed as an input for clustering of the third subset. Thereafter, clustering of this subset, the cluster centers, and membership knowledge corresponding to it is found. The same procedure is repeated for the clustering of the rest of the subsets. Unlike SRSIO-FCM [151], the proposed KSRSIO-FCM tackles linear and non-linear separable data by applying RBF kernel function, which map the input data space non-linearly into a high dimensional feature space. This is because, in KSRSIO-FCM, vector norm is used, which is defined in terms of RBF function instead of Euclidean distance. For performance comparison of this proposed algorithm, we have designed and implemented the kernelized version of the existing clustering algorithm, i.e., SLFCM on the Apache Spark cluster. The KSRSIO-FCM clustering algorithm is implemented with the help of an integrally proposed Kernelized version of Scalable Literal Fuzzy C-Means [151] termed as KSLFCM, to make the KSRSIO-FCM more efficient. The KSLFCM algorithm uses RBF kernel function to optimize the objective function. We executed these algorithms on Apache Spark to utilize its in-memory computation capability to overcome the difficulties that arise in fuzzy clustering while dealing with Big Data. The KSRSIO-FCM divides the data into subsets, and KSLFCM is performed on each subset sequentially, where the

Table 3.1: Main Math Symbols

Notation	Description
X	set of data samples
x_i	i^{th} data sample
M	membership matrix
V	set of initial cluster centers
V'	set of final cluster centers
v'_j	updated j^{th} cluster center
n	total number of data samples
s	number of subsets
c	number of clusters
R^d	d dimensions of R feature space
v_j	j^{th} initial cluster center
m_{ij}	membership degree of a data sample x_i to cluster v_j
p	fuzzification parameter
I	membership knowledge
I'	updated membership knowledge

input to each subset is a combination of the output of previous subsets. Before presenting the details of the proposed KSRSIO-FCM algorithm, we present the working of the KSLFCM algorithm. The details of the KSLFCM algorithm are presented in the subsequent section and Table 3.1 presents the description of the symbols that we are using throughout the discussion in this chapter.

3.2.1 Kernelized Version of SLFCM Algorithm to Handle Big Data

To design the kernelized version of the SLFCM algorithm implemented on the Apache Spark cluster, we first need to identify the computations in the SLFCM algorithm, which could be executed in a parallel manner, and the computations that could be executed only in a serial manner. As mentioned earlier, SLFCM can handle linear relations. To handle non-linear relations, the concept of the kernel method is introduced to extend SLFCM. The KSLFCM is implemented by applying the RBF kernel function. The kernel RBF is characterized in Eq. (2.11), which maps the input

data space non-linearly into a high dimensional feature space. The membership degree is calculated using data samples and cluster center values, as stated in Eq. (2.15). Therefore, the calculation of the membership degree of particular data samples can be performed in parallel on various slave nodes.

Algorithm 3.1 Algorithm for Kernelized Scalable Literal Fuzzy C-Means (KSLFCM) to Iteratively Minimize $J_p(M, V')$

Input: X, c, p, ϵ, (initial V); X is an array of data samples such that $X = \{x_1, x_2, ...x_n\}$, V is the set of initial cluster centers represented as $V = \{v_1, v_2, ...v_c\}$, c denotes the number of clusters, and ϵ represents the termination criteria.

Output: $V^{'}, I^{'}; V^{'}$ represents set of final cluster centers and $I^{'}$ represents the membership knowledge of all the data samples.

1: **If** V is not initialized, randomly initialize $V = \{v_1, v_2, ...v_c\}$.

2: **Compute** membership knowledge by using Eq. (2.15).

$$I' = X.Map(V).ReduceByKey() \tag{3.1}$$

3: **Compute** the set of final cluster centers $V' = \{v'_1, v'_2, ...v'_c\}$ by using Eq. (3.2).

$$v_j^{'} = \frac{sum\_d_j x}{sum\_d_j}, \forall j \in [1, c] \tag{3.2}$$

4: **If** $\parallel V' - V \parallel < \epsilon$ then stop, otherwise go to step 2.

5: **Return** I', V'.

In ***Algorithm 3.1***, the membership degree is calculated separately for each data sample. In Line 2 of ***Algorithm 3.1***, we have used the Map and ReduceByKey functions to obtain the parallel computation of the membership knowledge of all the data samples. Furthermore, Line 3 of ***Algorithm 3.1***, is used to update the cluster center values from membership degrees of all data samples. Thus, Line 3 of ***Algorithm 3.1*** is executed after membership degrees of all locations have been computed. At the master node, the membership degree of all the data samples is merged and saved as a membership knowledge I', which is required in Eq. (2.16) to update the cluster center

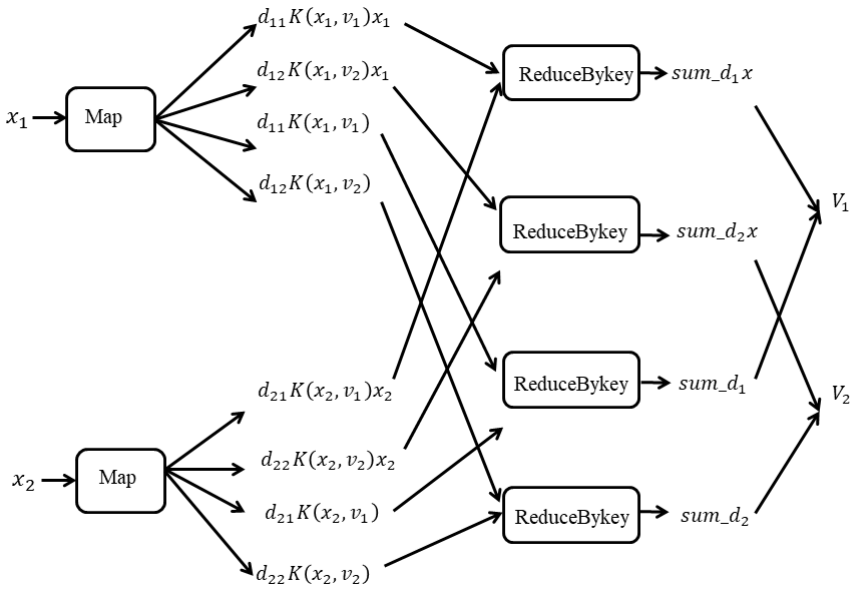


Figure 3.1: The figure describes repository space improvement by avoiding the storage of membership matrix of subsets.

v_j. After that, we have calculated the difference between the old values of the initialized cluster center and newly calculated cluster center values, and this is given in Line 4 of ***Algorithm 3.1***. Repeat this procedure until no change in the values of cluster centers is recognized. After that, all the iterations are executed sequentially since the updated cluster centers are required as input for the next iteration.

In order to reduce the space requirements in the proposed KSLFCM algorithm, we avoid storing the membership matrix of data samples. Figure 3.1 exhibits the whole methodology of repository space improvement, demonstrates the entire procedure of storage space optimization. For example: In ***Algorithm 3.1***, the membership matrix M is required to calculate the cluster centers (V') using Eq. (2.16). Rather than saving the huge membership matrix, calculation of the estimation of the parameter present in the numerator and denominator of Eq. (2.16), i.e., $m_{ij}^p K(x_i, v_j)x_i$, and $m_{ij}^p K(x_i, v_j)$, where m_{ij}^p is denoted as d_{ij} and $K(x_i, v_j)$ denoted as k_{ij}. The membership matrix M is expected to find cluster centers V' using Eq. (2.16) and for each sample x_i and cluster center v_j, which is represented as $d_{ij}k_{ij}x_i$ and as $d_{ij}k_{ij}$, respectively. This avoids the need of storing the huge membership matrix M. Then, we do the summation of all the

$d_{ij}k_{ij}x_i$ values and all the $d_{ij}k_{ij}$ values of the data samples comparable to the cluster center v_j to calculate the numerator and denominator of Eq. (2.16) which represent as sum_d_jx and sum_d_j and reserved as membership knowledge in variable I'. After that, we access these values in Line 3 of ***Algorithm 3.1*** to process the cluster centers by utilizing Eq. (2.16), which is used for calculating the updated cluster centers. Due to this, we save a significantly huge amount of space and computational time.

Map function

The map function works in the following manner. First, the entire data is split into several chunks at the master node. The master node logically separates the data and the slave nodes deal with chunks of the separated data. The membership degree of a point is determined by utilizing the data sample itself and the cluster center values. Thus, the computation of membership degrees of two data samples is independent of each other. Hence, we have implemented this operation independently for every data sample on a slave node and join the subsequent values on the master node. This obtained result is the same as that of the result obtained by implementing this operation for all data samples on a solitary machine. According to Line 2 of ***Algorithm 3.1***, the Map function is executed in parallel on Apache Spark and returns the results to RDD in terms of key/value pairs [135]. RDD is nothing but a data structure used to store the samples efficiently in memory as already discussed in Section 2.2. Here, a map function is created corresponding to each data sample, and it gives as many outputs equal to the number of clusters as results in RDD in terms of key-value pairs where each key represents a cluster number and a value contains the result of an operation performed in a Map function. Thus, this makes the execution of the KSLFCM algorithm much faster by saving computational time [233]. The Map function, characterized in ***Algorithm 3.2***, computes the membership degree of a data sample for each cluster center and returns every one of them exclusively. Consequently, each Map function gives the same number of outcomes as the number of clusters. ***Algorithm 3.2***, depicts the arrangement of tasks performed during a Map function to acquire $m_{ij}^p K(x_i, v_j)x_i$, and $m_{ij}^p K(x_i, v_j)$ for each point x_i and cluster center v_j as discussed in Line 3 of ***Algorithm 3.1***. In this algorithm, $m_{ij}^p K(x_i, v_j)x_i$,

Algorithm 3.2 Algorithm for Map Function (Map(x,V))

Input: x_i, V.

Output: $< j, < d_{ij}k_{ij}x_i, d_{ij}k_{ij} >>$

1: **for each** v_j **in** V **do**

2: $d_{ij}k_{ij}= d_{ij}$ (membership degree of x_i concerning v_j, k_{ij} (kernel value for an i^{th} data sample in the j^{th} cluster).

3: $d_{ij}k_{ij}x_i = d_{ij}k_{ij} * x_i$.

4: **yield** $< j, < d_{ij}k_{ij}x_i, d_{ij}k_{ij} >>$.

5: **end for**

and $m_{ij}^p K(x_i, v_j)$ is denoted as $d_{ij}k_{ij}x_i$ and $d_{ij}k_{ij}$, respectively. The parameter x_i represents the i^{th} data sample, yield is used when a function has multiple return values, and $< j, < d_{ij}k_{ij}x_i, d_{ij}k_{ij} >>$; $\forall j \in [1, c]$ represents key-value pairs as outputs of a Map function.

ReduceByKey function

The Map function results in many key-value sets having a similar key value. ReduceByKey performs tasks on the key-value matches that have a similar key. To rearrange things, here we depict the tasks that are performed on two such key-value sets. Spark deals with similar tasks on all the key-value pairs and Map outputs. To update the cluster center characteristics, we need to find the numerator and denominator in Line 3 of ***Algorithm 3.1***. Since we have determined, $m_{ij}^p K(x_i, v_j)x_i$, and $m_{ij}^p K(x_i, v_j)$ as $d_{ij}k_{ij}x_i$ and $d_{ij}k_{ij}$, respectively, for each point x_i and cluster center v_j. During the Map stage, we have to add every one of these values. This is done using the ReduceByKey function which is described in ***Algorithm 3.3***. This gives us the numerator and denominator of Eq. (2.16) for each cluster center v_j as sum_d_jx and sum_d_j, respectively. In ***Algorithm 3.3***, a and b are the output of two Map functions, concerning cluster center v_j, $(d_{ij}k_{ij}x_i)_a$ and $(d_{ij}k_{ij}x_i)_b$ denotes the value of $d_{ij}k_{ij}x_i$ corresponding to Map function outputs a and b respectively, $(d_{ij}k_{ij})_a$ and $(d_{ij}k_{ij})_b$ denotes the value of $d_{ij}k_{ij}$ corresponding to Map function outputs a and b respectively. The output of the ReduceByKey function is used to calculate the updated cluster center values using Eq. (2.16) on the master node.

Algorithm 3.3 Algorithm for ReduceByKey Function (ReduceByKey(a,b))

Input: a, b such that $a =< j, < (d_{ij}k_{ij}x_i)_a, (d_{ij}k_{ij})_a >> b =< j, < (d_{ij}k_{ij}x_i)_b, (d_{ij}k_{ij})_b >>$.

Output: $< j, < sum\_d_jx, sum\_d_j >>$

1: $sum\_d_jx = (d_{ij}k_{ij}x_i)_a + (d_{ij}k_{ij}x_i)_b$

2: $sum\_d_j = (d_{ij}k_{ij})_a + (d_{ij}k_{ij})_b$

3: **return**:$< j, < sum\_d_jx, sum\_d_j >>$

3.2.2 Proposed Design of a Novel KSRSIO-FCM Algorithm to Handle Big Data

The KSRSIO-FCM algorithm starts by partitioning the entire data into various subsets (chunks). These subsets are created by randomly selecting data samples from the entire dataset by selecting 100% of the data without replacement for Big Data. The data samples present in a subset are distinct from the data samples present in other subsets. The clustering of each subset is done in parallel on the Apache Spark cluster. ***Algorithm 3.4***, summarizes the steps of KSRSIO-FCM. In this algorithm, we compute cluster centers and membership knowledge for the first subset X_1, denoted as $V^{'}$ and $I^{'}$, respectively in parallel by using Eqs. (3.2) and (3.1). Data samples in one subset are different from other subsets. For the clustering of the first subset, the cluster centers are initialized randomly. Then KSRSIO-FCM calculates the cluster centers and membership knowledge for initial subset X_1, represented by V' and I', respectively, by applying KSLFCM. Then V' is fed as an input cluster center for clustering of second subset X_2 . The KSRSIO-FCM performs clustering of X_2 by applying KSLFCM and calculates the cluster centers and membership knowledge represented by I and V'. However, KSRSIO-FCM does not feed V' as an input for the clustering of the third subset. This is because KSRSIO-FCM considers the reality that arbitrary partitioning may bring about the two continuous subsets containing data samples 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 KSRSIO-FCM avoids utilizing cluster centers of the previous

Algorithm 3.4 Algorithm for Scalable Random Sampling with Iterative Optimization Fuzzy C-Means to Iteratively Minimize $J_p(M, V^{'})$

Input: X, c, p, ϵ; X is an array of data samples such that $X = \{x_1, x_2, ...x_n\}$

Output: $I^{'}, V^{'}$

1: **Begin**
2: **Partition** set X into s subsets such that $X = \{X_1, X_2, ...X_s\}$.
3: **Randomly** select X_1 from X without replacement where X_1 represents the first subset consist of random n/s samples.
4: $I^{'}, V^{'} = KSLFCM(X_1, c, p, \epsilon)$
5: **for** $t = 2$ to s do
6: $I, V' = KSLFCM(X_t, c, p, \epsilon, V')$
7: Merge the partition of all blocks of processed subsets
8: **for** $j = 1$ to c **do**
9: $I_j^{'} =< j, < (sum\_d_j x)_{I_j}, +(sum\_d_j x)_{I_j^{'}}, (sum\_d_j)_{I_j}, + (sum\_d_j)_{I_j^{'}} >>$
10: **end for**
11: Compute updated cluster center v_j' using:
12: $< j, < sum\_d_j x, sum\_d_j >>$ in I' by Eq. (2.16) $\forall \in [1, c]$
13: **end for**
14: **Return** I', V'
15: **End**

iteration as the initial cluster centers for the clustering of the current subset. Rather it joins the membership knowledge of all the processed subsets. Thus, it merges the membership knowledge of all the processed subsets, i.e., it combines I' and I, and the updated cluster centers are evaluated using Eq. (2.16). These cluster centers are the more prominent way of estimation of actual cluster centers since they are computed with the combined membership knowledge of a larger number of data samples that cover a wider sample space.

In ***Algorithm 3.1***, updated membership knowledge I' gives the numerator and denominator of V. Since membership values of one data sample are independent of other data samples, combining membership matrices equivalently means union of the first subset (I_1) and the second subset (I_2). Thus, instead of allotting a huge amount

of space in storing I, we can combine I_1 and I_2 without loss of any information. This helps in optimizing space, this optimization analogy works for the remaining subsets, i.e., for all $s \in [3, s]$, where s is the number of subsets. Since the operations on one subset are done serially, effectively only $(\frac{1}{s})^{th}$ times of the space will be used for KSLFCM. Due to this, we save a significantly huge amount of space and computational time.

Figure 3.2 demonstrates the workflow of KSRSIO-FCM, which makes use of KSLFCM for computing membership knowledge and cluster centers of all subsets. It demonstrates how the data sample is randomly distributed into various subsets and how the underlying cluster centers are randomly chosen for the clustering of the initial subset.

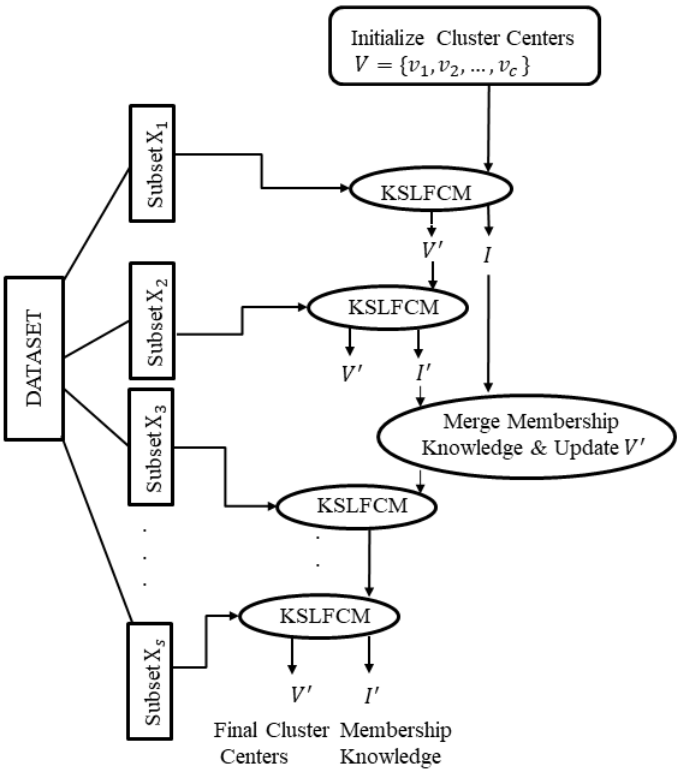


Figure 3.2: Workflow of KSRSIO-FCM algorithm.

3.3 Complexity Analysis

Many researchers used kernel based clustering algorithms with huge data where the complexity of the distance equations is quadratic [97]. The complexity analysis of our proposed KSLFCM and KSRSIO-FCM algorithm leads to the linear complexity in terms of the input data sample. The space complexity is regarding the amount of information held in RAM throughout the calculation. Table 3.2 demonstrates the complexity analysis of the kernelized algorithms in terms of variables. Here, X represents the dataset that comprises of n number of data samples in the d high dimensional space such that $X = \{x_1, ..., x_n\}$, $x_i \in R^d$ where c is the number of clusters, w is the number of the slave nodes in Spark cluster, and X is randomly distributed into s number of partitions delineated as $X = \{X_1, X_2, \ldots . X_s\}$ such that each subset consists of random n/s samples. The number of iterations required for the termination is denoted as t. However, this is not constant and may vary at different iterations. Thus, for simplicity, we consider t as the maximum number of iterations for the execution of KSLFCM and KSRSIO-FCM. In these algorithms, we compute the expense of membership degrees during the Map stage. Each Map task calculates the membership degree of one data sample as to c cluster centers. Since each data sample has d-dimensions, the computation of each membership degree takes $O(d)$ time and $O(d)$ space. Each Map task takes $O(cd)$ time and $O(cd)$ space. The ReduceByKey task linearly adds the estimations of all the Map operations, corresponds to one cluster on each slave node, and joins the subsequent values on the master node. We have assumed an equal distribution of jobs to the w slave nodes. Joining the outputs of Reduce action takes $O(cwd)$ time and $O(cd)$ space.

KSLFCM executes Map and ReduceByKey tasks on the whole dataset. Every slave node processes (n/w) data samples in parallel. In this way, the Map stage takes an aggregate of $O(nd/w)$ time and $O(ncd)$ space for each iteration across all slave nodes. Expecting that KSLFCM keeps running for t iterations, the total time taken for a Map stage is $O(ncdt/w)$. Since Map results have been held in memory just for the term of one iteration of KSLFCM, hence the total space complexity for

Table 3.2: Complexity Analysis of Kernelized Algorithm.

Algorithm	**Time Complexity**	**Space Complexity**
KSLFCM	$O(ncdt/w)$	$O(ncd)$
KSRSIO-FCM	$O(ncdt/w)$	$O(ncd/s)$

the Map stage of KSLFCM is $O(ncd)$. The ReduceByKey action linearly adds s/w Map results, comparing to each cluster centers in parallel, on every slave node. This takes $O(nd/w)$ time and $O(cd)$ space on a slave node. Since each slave node performs tasks on n/w data in parallel on the same amount of time, the total required time is $O(nd/w)$. Every slave nodes give c outputs, which are accumulated on the master node and added. This takes $O(cd/w)$ time and $O(cd/w)$ space. Along these lines, the total time complexity for the ReduceByKey stage is $O(ncdt/w)$ and space complexity is $O(cd/w)$. Accordingly, the time complexity of KSLFCM is $O(ncdt/w)$ and space complexity is $O(ncd))$ where $n >> w$, c.

KSRSIO-FCM partitions the whole dataset X into s equivalent subsets with the end goal that $X = \{X_1, X_2, \ldots . X_s\}$, where every subset is of size (n/s). It executes KSLFCM over each one of these subsets sequentially. The time and space complexity of performing KSLFCM over every subset is $O(ncdt/sw)$ and $O(ncd/s)$, respectively. Since every one of the subsets is handled in a steady progression, the time complexity is $O(ncdt/w)$ while the space complexity remains $O(ncd/s)$ because data corresponding to one subset is not held in the memory while processing the next subset.

In Table 3.2, we have described that KSLFCM and KSRSIO-FCM share a similar time complexity. Although, it appears that both have a similar run-time, but this is not the case. Since we partitioned the whole dataset into various subsets and performed clustering over each subset in KSRSIO-FCM. Due to this, clustering performed by KSRSIO-FCM on each subset converges by taking the less number of iterations (t) for each subset. Hence, KSRSIO-FCM has lesser run-time as it performs clustering on a small chunk of data in each subset in comparison with KSLFCM that performs clustering of the whole data. The performance analysis in terms of the ratio of time taken by KSRSIO-FCM with different chunk sizes of each dataset versus the time taken

by KSLFCM on the whole dataset is shown in Figure 3.3 in a subsequent section.

3.4 Experimental Evaluation

In this section, we present the experiments of our proposed scalable kernelized clustering algorithms on various Big Datasets. We analyze the performance of KSRSIO-FCM in comparison with KSLFCM, SRSIO-FCM, and SLFCM by using different measures such as NMI [200], ARI [201], and F-score [193], respectively. These three measures (NMI, ARI, F-score) are briefly discussed in Section 2.5.1. Also, we have done the performance analysis in terms of the ratio of time taken by KSRSIO-FCM with different chunk sizes of each dataset versus the time taken by KSLFCM on the whole dataset. All these approaches discussed above are executed on the Apache Spark cluster.

3.4.1 Datasets and Experimental Settings

To demonstrate the efficacy of the proposed KSRSIO-FCM over the proposed KSLFCM, we have created Big Datasets by taking data from various sources [234, 235, 236, 237] and evaluate the performance of these approaches on various Big Datasets, i.e., SUSY, Monarch-Skin, Mnist8m, and Reproduced-Dim32. The details of these datasets are presented in the subsequent section. To evaluate the performance of all the approaches, we fix the value of fuzzifier $p = 1.75$ and termination criteria $\epsilon = 0.001$ for these datasets used in the experimental study. Also, we have fixed the value of number of clusters (c) for SUSY ($c = 2$), Monarch-Skin ($c = 2$), Mnist8m ($c = 10$), and Reproduced-Dim32 ($c = 16$), respectively. After exhaustive experimentation, we found that these values are more suitable for the datasets because on these values the datasets achieve better performance. Also, these values are proven to work well for most of the datasets [32, 238].

Table 3.3: Description of Datasets.

Datasets	**#instances**	**#features**	**Classes**	**Dataset Size**
SUSY	5,000,000	18	2	2.4 GB
Monarch-Skin	1,470,342,000	4	2	25.4 GB
MNIST8m	14,986,500	784	10	31 GB
Reproduced-Dim32	1,00,000	32	16	12.4 GB

3.4.2 Experimental Environment

The Apache Spark cluster is used to perform the experimental evaluation. The spark cluster consists of five slave nodes and one master node. Each slave node has the following configuration: Intel(R) Core(TM) i7-77000 CPU @ 3.60 $GHz \times 8$, 16 GB RAM, 1TB storage, and the master node has 32 GB RAM, Intel(R) CPU E5-1607 v3 @ 3.10 $GHz \times 4$, 1TB storage. HDFS is used across the cluster for data storage with spark standalone mode. The algorithms were implemented in Python version 3.6.7, Apache Spark cluster 2.4.0 setup on Ubuntu 18.04 with Hadoop version 2.7.3. As discussed earlier in Section 2.2, Apache Spark requires a cluster manager and a distributed storage system so here we used spark standalone [239] for cluster management and HDFS [140] for storing data across the Spark Cluster.

3.4.3 Datasets Description

We utilize four real-world datasets for our experiments that are openly accessible. We look at the execution of KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM algorithms on these datasets. Table 3.3 demonstrates a few characteristics of these datasets used for the exploratory experimentation, and the depiction of these datasets is displayed as follows:

SUSY Dataset: We have obtained this dataset from the UCI machine learning repository [234]. The dataset size is 2.5 GB with 5,000,000 instances.

Monarch-Skin Dataset: We made the Cartesian result of Skin division data acquired from UCI [234] with the monarch dataset [235] to examine our algorithm on

the massive size. Skin data comprises of the measurement 245057×4, while monarch data consist of one element. We utilized a 25.4 GB estimated subset of the Cartesian product, with 1,470,342,000 occurrences for our tests.

MNIST8m Dataset: It contains dark scale pictures of hand-drawn digits, from zero through nine [236]. Each picture is 28 pixels in height and 28 pixels in width for 784 pixels altogether. The following dataset is 31 GB in size and contains 14,986,500 occurrences separated into ten classes.

Reproduced-Dim32 Dataset: The Dim32 data collection consists of 1024 high-dimensional informational indexes and 16 Gaussian clusters. All the clusters are isolated even in the higher dimensions. To get a huge dataset, we replicated it multiple times. The resulting dataset is 12.4 GB in size and contains 1,00,000 cases, referred to as Reproduced-Dim32 dataset [237].

3.4.4 Experimental Results and Discussion

This section presents the discussion of the effectiveness of KSRSIO-FCM in comparison with KSLFCM, SRSIO-FCM, and SLFCM evaluated on four datasets as per the various estimates, such as NMI, ARI, and F-score, respectively. For every dataset, we have compared the performance of KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM by utilizing the parameters as portrayed previously. The estimations of NMI, ARI, and F-score for KSRSIO-FCM and SRSIO-FCM on various chunk sizes of four datasets in comparison with KSLFCM and SLFCM are shown in tables reported subsequently. Additionally, it calculates the performance analysis in terms of the ratio of time taken by KSRSIO-FCM with different chunk sizes of each dataset versus the time taken by KSLFCM on the whole dataset.

Table 3.4 organizes the estimations of NMI, ARI, and F-score for the SUSY dataset. NMI demonstrates the quality of clustering. The higher NMI value indicates good clustering. While speaking about the difference in values of NMI among various chunk sizes, the NMI value for 100% (KSLFCM) chunk size is comparatively smaller than NMI values achieved with KSRSIO-FCM with different chunk sizes. Again, like NMI, a higher ARI value represents better clustering. On comparing the values of ARI for

Table 3.4: Results of KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM with varying chunk sizes on SUSY Dataset.

Algorithm	**Chunk Size**	**Measures**		
		NMI	ARI	F-score
KSRSIO-FCM	1%	0.0638	0.0434	0.6387
	2.5%	0.0631	0.0534	0.7372
	5%	0.0700	0.0346	0.7463
	10%	0.0836	0.1042	0.6383
	20%	0.0731	0.0535	0.6830
KSLFCM	100%	0.0603	0.0415	0.6203
SRSIO-FCM	1%	0.0287	0.0428	0.6416
	2.5%	0.0290	0.0431	0.6420
	5%	0.0287	0.0427	0.6409
	10%	0.0286	0.0430	0.6412
	20%	0.0287	0.0428	0.6416
SLFCM	100%	0.0258	0.0380	0.6079

different chunk sizes, the ARI value for chunk size 100% (KSLFCM) is comparatively lower than the values achieved by KSRSIO-FCM on different chunk sizes. Among the different chunk sizes of SUSY data, the KSRSIO-FCM achieved the highest value of ARI for a chunk size of 10%. Like NMI and ARI, F-Score also demonstrates the quality of clustering. Thus, comparing values of F-score for different chunk sizes, we analyze that the value of F-score is lower for KSLFCM compared to KSRSIO-FCM. Besides this, the values of the F-score vary for different chunk sizes of KSRSIO-FCM, and it achieves a remarkable value of F-score for chunk size 5%. Hence, we get superior clustering results in terms of F-score when compared KSRSIO-FCM with KSLFCM. Thus, comparing all the three measures, we conclude that KSRSIO-FCM performance is much better than KSLFCM in terms of NMI, ARI, and F-score, respectively. Furthermore, on comparing the values of NMI, ARI, and F-score for different chunk sizes of the SUSY dataset, the KSRSIO-FCM performs better than SRSIO-FCM and SLFCM.

In Table 3.5, we have reported the results on the Monarch-Skin dataset in terms of NMI, ARI, and F-score, respectively. On comparing NMI, KSLFCM has attained the lowest NMI value compared to the NMI values achieved on different chunk sizes of KSRSIO-FCM. Similarly, for ARI, KSLFCM has obtained a lower value compared to

Table 3.5: Results of KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM with varying chunk sizes on Monarch-Skin Dataset.

Algorithm	**Chunk Size**	**Measures**		
		NMI	ARI	F-score
KSRSIO-FCM	1%	0.0237	0.0487	0.6685
	2.5%	0.0250	0.0567	0.7068
	5%	0.0389	0.0462	0.6987
	10%	0.0439	0.0525	0.6387
	20%	0.0217	0.0472	0.7287
KSLFCM	100%	0.0194	0.0444	0.5976
SRSIO-FCM	1%	0.0127	0.0358	0.6484
	2.5%	0.0127	0.0358	0.6484
	5%	0.0127	0.0358	0.6484
	10%	0.0127	0.0358	0.6484
	20%	0.0127	0.0358	0.6484
SLFCM	100%	0.0358	0.0444	0.6484

KSRSIO-FCM for different chunk sizes. Now, the F-score value achieved by KSRSIO-FCM for all chunk sizes is comparatively much higher than the KSLFCM. Furthermore, KSRSIO-FCM achieves the remarkable value of F-score for a chunk size of 20%. Therefore, comparing all three measures, we conclude that KSRSIO-FCM performs much better than KSLFCM in terms of NMI, ARI, and F-score, respectively. Furthermore, on comparing the values of NMI, ARI, and F-score for different chunk sizes of the Monarch-Skin dataset, the KSRSIO-FCM also performs much better than SRSIO-FCM and SLFCM.

In Table 3.6, we have reported the results on the MNIST8m dataset in terms of NMI, ARI, and F-score, respectively. Observing the values of NMI, KSLFCM has obtained a very low value, whereas the KSRSIO-FCM achieved a much higher value of NMI for the different chunk sizes of the MNIST8m dataset. Furthermore, the KSRSIO-FCM achieved the highest value of NMI for a chunk size of 2.5%. In the case of ARI, the value achieved by KSRSIO-FCM is much better than KSLFCM on almost all the chunk sizes except for chunk size 1%. Furthermore, the best ARI value attained by KSRSIO-FCM is for a chunk size of 5%. In the case of F-score, we analyze that the F-score value obtained by both KSRSIO-FCM and KSLFCM, which is almost similar. Therefore, comparing all three measures, we conclude that

Table 3.6: Results of KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM with varying chunk sizes on MNIST8m Dataset.

Algorithm	**Chunk Size**	**Measures**		
		NMI	ARI	F-score
KSRSIO-FCM	1%	0.42837	0.2377	0.0658
	2.5%	0.47687	0.3188	0.0946
	5%	0.45577	0.3347	0.0835
	10%	0.41679	0.2745	0.1000
	20%	0.39779	0.3233	0.1034
KSLFCM	100%	0.27879	0.2574	0.1099
SRSIO-FCM	1%	0.0022	0.0006	0.1018
	2.5%	0.0014	0.0005	0.0928
	5%	0.0016	0.0001	0.0922
	10%	0.0016	0.0006	0.1066
	20%	0.0021	0.0005	0.1016
SLFCM	100%	0.0043	0.0004	0.0962

KSLFCM performs approximately equally well as KSRSIO-FCM in terms of the F-score. Moreover, KSRSIO-FCM performs better than KSLFCM in terms of ARI and NMI measures. Furthermore, on comparing the values of NMI, ARI, and F-score for different chunk sizes of the MNIST8m dataset, the KSRSIO-FCM performs much better than SRSIO-FCM and SLFCM in the case of NMI and ARI. On the other hand, in case of F-score, the value of F-score attained by KSRSIO-FCM for different chunk sizes are almost close to the value of F-score attained by SLFCM and SRSIO-FCM for different chunk sizes.

In Table 3.7, we have reported the results on the Reproduced-Dim32 dataset in terms of NMI, ARI, and F-score, respectively. Observing the values of NMI, KSLFCM obtained a very close result to that of KSRSIO-FCM for chunk size 10% and 1%. However, the KSRSIO-FCM for chunk size 20% achieves remarkable results in comparison with KSLFCM. In the case of ARI, a similar analogy can be drawn. The reason for a very close result is the replication of a dataset. The fraction of data samples that belong to a specific class is very high than the data samples to other classes. As explained earlier, KSRSIO-FCM achieves the clustering of whole data by dividing it into distinct subsets. Hence, this could be due to the randomly chosen subsets, which contain more data samples belonging to the majority class. So, as a result, skewed cluster

Table 3.7: Results of KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM with varying chunk sizes on Reproduced-Dim32 Dataset.

Algorithm	**Chunk Size**	**Measures**		
		NMI	ARI	F-score
KSRSIO-FCM	1%	0.5772	0.2268	1
	2.5%	0.6325	0.2046	1
	5%	0.6123	0.2697	1
	10%	0.5258	0.1601	1
	20%	0.7962	0.3403	1
KSLFCM	100%	0.5646	0.1957	0.9998
SRSIO-FCM	1%	0.4218	0.0094	0.12878
	2.5%	0.0357	0.0076	0.123
	5%	0.4517	0.0008	0.1239
	10%	0.4971	0.1198	0.125
	20%	0.5268	0.1385	0.1875
SLFCM	100%	0.4869	0.1184	0.125

centers are generated in KSRSIO-FCM in comparison with KSLFCM. On the other hand, in case of the F-score, we analyze that the F-score value achieved by KSRSIO-FCM for different chunk sizes is almost the same, and a bit low for KSLFCM. In this manner, randomly picked subsets may contain more data samples belonging to the majority class. Therefore, comparing all the three measures, we conclude that KSLFCM performs approximately equally well as KSRSIO-FCM in terms of F-score. Furthermore, on comparing the values of NMI, ARI, and F-score for different chunk sizes of the Reproduced-Dim32 dataset, the KSRSIO-FCM performs better than SRSIO-FCM and SLFCM.

Figure 3.3 demonstrates the performance analysis in terms of the ratio of time taken by KSRSIO-FCM with different chunk sizes of each dataset versus the time taken by KSLFCM on the whole dataset. Depending upon the size of the datasets, the ideal chunk size may change for various datasets. The performance analysis of KSRSIO-FCM in comparison with KSLFCM is characterized as the proportion of time taken by KSRSIO-FCM and that by KSLFCM, i.e., $t_{KSRSIO-FCM}/t_{KSLFCM}$. In any case, the performance analysis of KSRSIO-FCM and KSLFCM would likewise rely upon the number of iterations for which KSRSIO-FCM keeps running on each chunk of data.

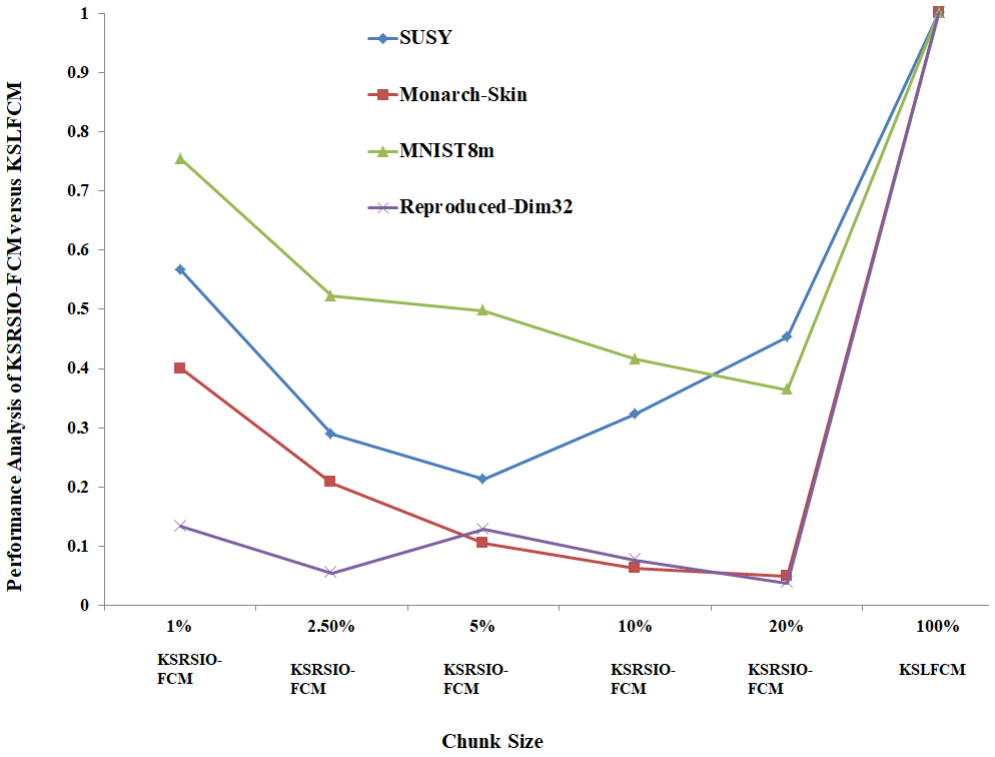


Figure 3.3: Performance analysis in terms of the ratio of time taken by KSRSIO-FCM with different chunk sizes of each dataset versus the time taken by KSLFCM on the whole dataset.

3.5 Summary

In this chapter, a novel scalable kernelized clustering algorithm is proposed, referred to as the KSRSIO-FCM algorithm for handling non-linear Big Data. The KSRSIO-FCM algorithm is designed by enhancing the SRSIO-FCM algorithm discussed earlier. The KSRSIO-FCM partitioned the Big Data into various chunks and processed the data samples present within the chunk in a parallel manner. One distinctive characteristic of the KSRSIO-FCM is that due to the parallel processing of data samples within the chunk, it significantly reduces run-time for the clustering of such a huge amount 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 for storing the large membership matrix, which significantly reduces the run-time and storage space and thus makes the execution of the proposed algorithm much faster. This is a good optimization strategy for clustering Big Data since the membership matrix is too huge to be stored. The KSRSIO-FCM algorithm is implemented using the Big Data processing framework called Apache Spark to deal with the challenges associated with fuzzy clustering for handling Big Data.

To compare our proposed KSRSIO-FCM, SLFCM is enhanced to make a kernelized SLFCM (KSLFCM) algorithm for handling Big Data. We have implemented the KSLFCM on the Apache Spark cluster, which is an integral part of the algorithm. Experimental results are evaluated on several Big Datasets in terms of various performance measures such as NMI, ARI, F-score, respectively. Moreover, we have carried out a performance analysis of KSRSIO-FCM versus KSLFCM. The efficacy of the KSRSIO-FCM is analyzed in terms of space and time complexity. It is observed that the space complexity of the KSRSIO-FCM is significantly less in comparison with the KSLFCM. The KSLFCM and KSRSIO-FCM share the same time complexity, but the run-time of KSRSIO-FCM is significantly lesser as compared to the KSLFCM. This is because KSRSIO-FCM performs clustering on a lesser amount of data by dividing the entire data into various subsets that lead to faster convergence by taking fewer iterations than the KSLFCM, which performs clustering on the entire data. The same

can be verified with the experimental evaluations carried out on several Big Datasets. The empirical evaluations show that the KSRSIO-FCM significantly outperformed the KSLFCM by achieving higher or comparable clustering results in terms of NMI, ARI, and F-score, respectively. Additionally, we also compared our proposed KSRSIO-FCM algorithm with SRSIO-FCM. Furthermore, performance analysis of KSRSIO-FCM versus KSLFCM is carried out. The performance analysis is done in terms of the ratio of time taken by KSRSIO-FCM with different chunk sizes of each dataset versus the time taken by KSLFCM on the whole dataset. The results indicate that the KSRSIO-FCM has great potential in Big Data clustering. The proposed KSRSIO-FCM is also applied on real-life plant genome datasets, i.e., Single Nucleotide Polymorphisms (SNPs) sequences of soybean and rice plant species. The performance investigation of proposed KSRSIO-FCM on plant genome datasets is reported in Chapter 5. Furthermore, the proposed KSRSIO-FCM approach is also applied on SARS-CoV-2 protein datasets, which is reported in Chapter 7.

Chapter 4

Scalable Incremental Fuzzy Consensus Clustering Algorithms for Handling Big Data

In the previous chapter, the fuzzy clustering-based proposed scalable kernelized algorithm, KSRSIO-FCM, does not consider the set of partitions as input to find the cluster. In this chapter, SRSIO-FCM [52] and Fuzzy Consensus Clustering (FCC) [53] methods are combined to develop a Scalable Incremental Fuzzy Consensus Clustering (SIFCC) algorithm using Apache Spark cluster. The SIFCC aims to identify a fuzzy consensus partition with overlapping clusters from a set of fuzzy partitions to improve the quality of clusters for Big Data. In this way, the proposed SIFCC method gets benefited from both the approaches, which is discussed in detail in the Section 4.2.

4.1 Introduction

Data clustering is the key, but it is a highly challenging issue in the areas of data mining and machine learning [240]. Because of the unstable development of huge data, scientists and experts have come to understand that a single fuzzy clustering algorithm cannot handle complex data. Henceforth, the FCC consequently develops as interesting research headings. FCC focuses on fusing various existing clustering results and has a global objective function that guides the consensus clustering. FCC expects

to find overlapping clusters from a set of crisp or fuzzy Basic Segments (BSs). In FCC, the basic partitions/segments are combined to acquire a better solution capable of managing all objectives of different partitions which sometimes are contradictory. Presently, Wu et al. [53] proposed a FCC for the application of Big Data. This approach determines a group of utility functions for FCC and updates FCC to a weighted piece-wise FCM, which increases high effectiveness using the FCM-like iterative procedure. When it comes to Big Data, FCC seems a good choice. This is mainly due to the fact that BSs are taken into consideration, where the Big Data is split into several small subsets to obtain BSs. Finally, consensus clustering is called to combine these BSs. The FCC uses the FCM approach for generating BSs. As many researchers have encountered the problem of cluster overlapping in FCM. To address the issues of FCM, in the proposed SIFCC approach, SRSIO-FCM is used to generate BSs. The SRSIO-FCM (discussed in Section 2.3.2) overcomes the sudden rise in several iterations during clustering and also handles the Big Data difficulties [52]. The proposed SIFCC is executed on an Apache Spark cluster with the objective that it can work efficiently for Big Data. The experimental results demonstrate that the proposed SIFCC approach is essentially superior to the other existing methodologies. To establish the comparison, we have designed and implemented the scalable model of the existing FCC [53] algorithm on Apache Spark cluster, named SFCC. The concepts of SRSIO-FCM, FCC, and Apache Spark framework are combined to develop scalable fuzzy consensus clustering algorithms, i.e., SIFCC and SFCC for clustering of Big Data.

4.2 Proposed Scalable Incremental Fuzzy Consensus Clustering Algorithms for Handling Big Data

In this section, we discuss the proposed Scalable Incremental Fuzzy Consensus Clustering (SIFCC) algorithm. This is a scalable, incremental version of the existing FCC [53] with the modifications needed to handle the challenges associated with fuzzy

clustering for Big Data. We worked in three stages to perform clustering of Big Data using the proposed SIFCC approach which is summarized as follows:

(i) In the first stage, an SRSIO-FCM [52] algorithm takes the dataset as an input to generate r BSs. For each dataset, we run the SRSIO-FCM algorithm r number of times by varying the number of clusters starting from the actual number of clusters, where r is different for each dataset. This leads us to have r BSs as input.

(ii) In the second stage, the outputs obtained from the first stage is combined to form concatenated basic segments. We concatenate the BSs, which are used as an input for our proposed SIFCC and SFCC algorithms.

(iii) Finally, the proposed SIFCC and SFCC algorithm executed on Apache Spark cluster for clustering of datasets to obtain final cluster centers.

The objective function of the SIFCC will be the same as that of the FCC. Now BSs for SIFCC is obtained by the result of partitioning on datasets using the SRSIO-FCM algorithm [241]. Initially, for the given dataset, SRSIO-FCM is applied r number of times to get r BSs. Each BS can be represented as $\mu^{(i)}$. The $\mu^{(i)}$ is a membership matrix of size $n \times c_i$ where c_i is a number of clusters for i^{th} BS. The membership vector for l^{th} data sample is given as: $\mu_{l.}^{(i)} = [\mu_{lj}^{(i)}]_{j=1}^{n}$ where, $l \in [1, n]$, n is a number of data samples. Now, on concatenating r BSs concerning each data sample, we get a concatenated basic segment, which is represented as Y. Moreover, the concatenated membership vector for l^{th} data sample, y_l is defined as: $y_l = [\mu_{l.}^{(1)}, \mu_{l.}^{(2)}, \ldots.., \mu_{l.}^{(r)}]$ and hence, $Y = [y_1, y_2, ...y_n]^\top$.

The algorithm begins by selecting 100% of the data without any replacement, and then data is partitioned into various subsets. First, the cluster similarity matrix is randomly initialized to cluster the first subset. After that, the cluster similarity matrix and membership matrix of the first subset is calculated. The cluster similarity matrix obtained after the first subset is used as input to cluster the second subset. It then finds the cluster similarity matrix and the membership matrix for the next subset.

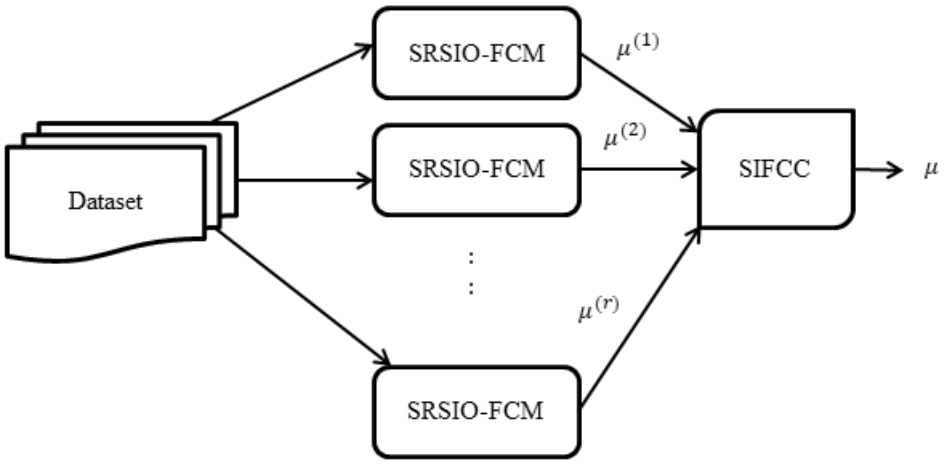


Figure 4.1: Architecture of proposed work.

Thereafter, the membership values of the first and second subsets are combined to process the new cluster similarity matrix. The obtained cluster similarity matrix is given as input to the third subset for clustering. This method is repeated to cluster all successive subsets. The combined output of the first and second subsets is specified as an input for the third subset. This is done so that the clustering in the third subset converges quickly and reduces divergence between cluster similarity matrix and third subset data, and this leads to less number of iterations (fast convergence). The gist of combined output (membership values) is stored in a particular format, named as a repository matrix (η). For the next subset, we calculate the membership value using η as an input, merge this information with the previous η, update it, and proceed until we reach the last subset. This makes it a SIFCC approach. Thus, the clustering results achieved with this approach are of significantly better quality. SIFCC is applied to the Apache Spark cluster, so it can work well for significantly larger datasets. Figure 4.1 depicts the architecture of the proposed work.

Since SIFCC is a scalable algorithm, to compare the performance of SIFCC with FCC, we have designed and implemented the scalable FCC model. The SFCC algorithm is implemented using Apache Spark cluster. The following subsections discuss the design and implementation details of these algorithms. However, before the demonstration of the SIFCC algorithm, we will show the SFCC algorithm, which is used in

Table 4.1: Main Math Symbols

Notation	Description
X	set of data samples
x_i	i^{th} data sample
M	membership matrix
r	basic segments (BSs)
V	set of initial cluster similarity matrix
V_{new}	set of final cluster similarity matrix
v_k^i	the k^{th} centroid for the i^{th} BS in the proposed SIFCC/SFCC
n	total number of data samples
s	number of subsets
c	number of clusters
c_i	the number of clusters for the i^{th} BS
p	fuzzification parameter
μ	the membership value of all data samples
μ^i	the fuzzy membership value for the i^{th} BS
μ_{new}	the final membership values
Y	concatenated basic segment
y_l	the l^{th} data sample in Y
η	repository matrix
η_{new}	updated repository matrix

the SIFCC algorithm for parallel computation of the degree of membership and the cluster similarity matrix. Table 4.1 details the symbols we use during the discussion in this chapter. The detailed description of these symbols is presented in Section 2.1.1.

4.2.1 Scalable Version of Fuzzy Consensus Clustering

The SFCC is a scalable version of the FCC [53] algorithm implemented using the Apache Spark cluster. To design a SFCC, we must first, identify the calculations of the algorithm that can be done in parallel and the calculations that can only be done in series. The most computationally intensive part of the SFCC is the calculation of the degree of membership of each data sample in relation to the entire cluster similarity matrix. Only the data sample and cluster similarity metrics are required to calculate the degree of membership. ***Algorithm 4.1*** describes how the SFCC algorithm works.

Algorithm 4.1 SFCC to Iteratively Minimize $J_p(\mu, V)$

Input: : Y, c, p, ϵ, (initial V); Y is an array of data samples such that $Y = \{y_1, y_2, ...y_n\}$.

Output: : V_{new}, η; V_{new} represents set of updated cluster similarity matrix $V_{new} = \{v_1, v_2, ..., v_{k\_new}...., v_{K\_new}\}$, η represents repository matrix.

1: **Declare** V randomly.
2: **Calculate** $\mathcal{G}$ using V, Y, and Map(V).
3: $\mathcal{G} = Y.Map(V)$
4: **Compute** V_{new} using $\mathcal{G}$.
5: $\eta = \mathcal{G}.ReduceByKey()$
6: **for** $< k, < sum\_ky, sum\_k >>$ in η **do**
7: $v_{k\_new} = \frac{sum\_ky}{sum\_k}$
8: **end for**
9: **Calculate** change in V.
10: $\epsilon_{new} = \parallel V_{new} - V \parallel$
11: **if** $\epsilon_{new} > \epsilon$ **then**
12: Repeat step 2 else stop.
13: **end if**

In Line 5 of ***Algorithm 2.4***, the membership degree calculated is independent of the other data samples. Therefore, the membership degree of different data samples can be calculated in parallel on different machines. So, in Line 2 of ***Algorithm 4.1***, we present the parallel computation of the membership value of all the data samples using the Map function. Moreover, in Line 6 of ***Algorithm 2.4***, membership degrees of all data samples are needed to update the cluster similarity matrix. Thus, Line 6 of ***Algorithm 2.4*** must be executed after the membership degrees of all samples have been determined. So, according to this, in Line 2 of ***Algorithm 4.1***, we calculate $\mathcal{G}$ using V, Y, and Map function. The Map function is called for each data sample present in Y denoted as y_l, and then it calculates μ_{lk} using Eq. (2.29). Thus, it yields $y_l * \mu_{lk}$, μ_{lk}^p for each k. In Line 5 of ***Algorithm 4.1***, η is computed using $\mathcal{G}$ and ReduceByKey function. Thereafter, the cluster similarity matrix is updated using

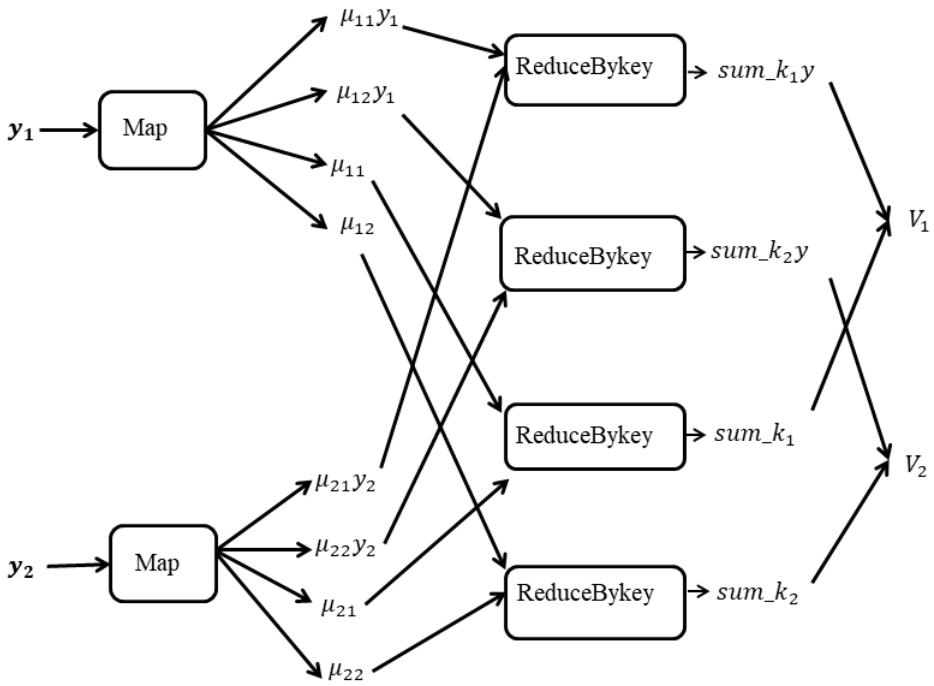


Figure 4.2: Methodology of SFCC.

v_{k_new}, where the summation of all the $\mu_{lk}^p y_l$ values is denoted as sum_ky, and all the μ_{lk}^p values are denoted as sum_k of the data samples corresponding to k^{th} cluster in $\mathcal{G}$. The difference between the previously initialized cluster similarity matrix and currently updated cluster similarity matrix value is then calculated using Line 10 of ***Algorithm 4.1***, and the value of the cluster similarity matrix changes significantly. Repeat this process until no changes are seen, according to Line 12 of ***Algorithm 4.1***. Here, ϵ is the stopping criteria. The updated cluster similarity matrix is then needed as input for the next iteration, so all iterations run in succession. When the stopping criteria are met, we obtain the final cluster similarity matrix, and from there, we obtain the membership value used for most calculations. Figure 4.2 exhibits the whole methodology of the SFCC algorithm. In this figure, $Y = \{y_1, y_2 ... y_n\}$, V represents the initial set of cluster similarity matrices, and μ represents the membership value of all data samples.

Map Function for SFCC

The slave node handles a small block of data that is organized by logically separating the data on the master. The degree of membership of a point is determined using the data sample itself and the values of the cluster similarity matrix. Therefore, the calculation of the degrees of membership of one data sample is not associated with the calculation of the degrees of membership of some other data sample. In accordance with these principles, we freely implement this action for each data sample in the slave node and merge the subsequent values in the master node. This produces a result similar to the result obtained by performing this step for all data samples on a single machine.

The Map function depicted in ***Algorithm 4.1*** calculates the membership degree of a data sample for each cluster and returns the two values, i.e., membership degree multiplied with y_l and membership degree, independently. As a result, each Map function returns the same result as the number of clusters. ***Algorithm 4.2*** shows the arrangement of tasks to be performed during the Map function to set $\mu_{lk}^{p} y_l$, and μ_{lk}^{p} for each point y_l and k^{th} to hold the row of the cluster similarity matrix, i.e., v_k using Line 6 of ***Algorithm 2.4***.

Algorithm 4.2 Map(y_l,V)

Input: y_l, V

Output: $< k, < \mu_{lk}^{p} y_l, \mu_{lk}^{p} >>$

1: **for** v_k **in** V **do**
2: k=index of cluster similarity matrix V.
3: μ_{lk}^{p}= membership degree of y_l w.r.t v_k.
4: $\mu_{lk}^{p} y_l = \mu_{lk}^{p} * y_l$
5: **yield** $< k, < \mu_{lk}^{p} y_l, \mu_{lk}^{p} >>$
6: **end for**

ReduceByKey Function

The Map function returns many sets of key-values with a similar key value. The ReduceByKey performs tasks on the key-value matches that have a similar key. To rearrange things, here we describe the tasks that are performed on two of these key-

value sets. Spark handles similar tasks on all key-value pairs and outputs on the Map function. To update the characteristics of the cluster similarity matrix, we need to provide the numerator and denominator in Line 2 of ***Algorithm 4.1***. As we have determined, $\mu_{lk}^{p} y_l$, and μ_{lk}^{p} as sum_ky and sum_k, respectively for each point y_l and cluster similarity matrix v_k. During the Map step, we must add each of these values. This is completed using the ReduceByKey function, which is described in ***Algorithm 4.3***. This gives us the numerator and denominator of Eq. (2.30) for each cluster similarity matrix v_k as sum_ky and sum_k, respectively.

Algorithm 4.3 ReduceByKey(a, b)

Input: a, b such that $a =< k, < (\mu_{lk}^{p} y_l)_a, (\mu_{lk}^{p})_a >> b =< k, < (\mu_{lk}^{p} y_l)_b, (\mu_{lk}^{p})_b >>$.

Output: $< k, sum\_ky, sum\_k >>$

1: sum_ky=$(\mu_{lk}^{p} y_l)_a$ +$(\mu_{lk}^{p} y_l)_b$

2: $sum\_k = (\mu_{lk}^{p})_a, (\mu_{lk}^{p})_b >>$

3: **Return**: $< k, < sum\_ky, sum\_k >>$

Where a and b are the result of two Map functions, corresponding to the k^{th} cluster, $(\mu_{lk}^{p} y_l)_a$ and $(\mu_{lk}^{p} y_l)_b$ signifies the estimation of $\mu_{lk}^{p} y_l$ comparing to Map function yields a and b separately, $(\mu_{lk}^{p})_a$ and $(\mu_{lk}^{p})_b$ signifies the estimation of μ_{lk}^{p} relating to Map function yields a and b individually. The yield of the ReduceByKey function is utilized to calculate the new cluster similarity matrix using Eq. (2.30) on the master node.

4.2.2 Proposed Design of a Novel SIFCC Algorithm to Handle Big Data

The SFCC algorithm, as discussed above, can be improved in terms of cluster quality, space-time complexity, and space optimization. We propose SIFCC algorithm to do this. The SIFCC algorithm is implemented on Apache Spark cluster. ***Algorithm 4.4*** describes the working of the SIFCC Algorithm. The critical task here is to find the cluster similarity matrix. To do this, we first need to find a membership degree of all data samples corresponding to each cluster center, which is represented as a membership matrix. For the computation of the membership matrix in the SIFCC

algorithm, the input data sample and cluster similarity matrix are required. Input for the SIFCC algorithm will be the same as SFCC, which is a concatenated basic segment (obtained by the column-wise union of various BSs). The SIFCC adds a step by dividing the input into subsets, and they are generated by random row sampling on the input. All subsets obtained are almost of the same size. These subsets are processed sequentially, and thus the quality of the cluster increases with each subset. ***Algorithm 4.4*** discusses the steps involved in SIFCC.

Algorithm 4.4 SIFCC to Iteratively Minimize $J_p(\mu, V)$

Input: Y, c, p, ϵ; Y is an array of data samples such that $Y = \{y_1, y_2, ...y_n\}$.

Output: μ_{new}; μ_{new} is the final membership values.

1: **Divide** Y into s number of subsets.
2: **Initialize** V randomly.
3: $\eta, V_{new} = SFCC(V, subset_1, c, p, \epsilon)$
4: **for** $<< subset\ in(2, number\ of$ subsets$)>>$ **do**
5: $\eta_{new}, V_{new} = SFCC(V_{new}, subset_s, K, p, \epsilon)$
6: $\eta = \eta \cup \eta_{new}$
7: $\eta = \eta.ReduceByKey()$
8: **for** $< k, < sum\_ky, sum\_k >>$ in η **do**
9: $v_{k\_new} = \frac{sum\_ky}{sum\_k}$
10: **end for**
11: **end for**
12: $\mu_{new} = Y.Map(V)$.

For the clustering of the first subset, the cluster similarity matrix (V) is randomly initialized. SIFCC computes updated cluster similarity matrix (V_{new}) and repository matrix (η) for the first subset Y_1 by applying SFCC. It then uses V_{new} as the initial cluster similarity matrix, i.e., V_{new} is used as V for the clustering of the second subset. The repository matrix and cluster similarity matrix are computed for the second subset Y_2 by applying SFCC and are denoted as η_{new} and V_{new}, respectively. It does not use the updated cluster similarity matrix V_{new} of the second subset, as the initial cluster similarity matrix for clustering of the third subset. This is because the SIFCC

takes into account the fact that random partitioning can result in subsets containing samples of different classes of data. Consequently, the cluster similarity matrix of these two subsets might be significantly different from each other. SIFCC avoids using such a cluster similarity matrix as the initial cluster similarity matrix. Instead, it combines the repository matrix of all the processed subsets. The repository matrix of the first two processed subsets are denoted as η and η_{new}. Now, these two, $\eta's$ are combined (union of rows), then η is computed using the ReduceByKey function (discussed in Section 4.2.1). Here, η is the reduced form of the membership matrix μ. So, $\eta = \eta \cup \eta_{new}$ gives the gist of the union of membership matrix of the previous subset and present subset. The Line 6 and Line 7 added in between ensures that the cluster similarity matrix does not diverge too much from the data and makes the algorithm incremental. Then the new cluster similarity matrix is evaluated using Line 8 of ***Algorithm 4.4***. This low computation intensive extra step guarantees good clustering quality. In Line 12 of ***Algorithm 4.4***, the Map function calculates μ_{new}, which is the final membership value, computed using the Map function discussed in ***Algorithm 4.5***.

In ***Algorithm 4.4***, we need V for the third subset, which should be obtained from the combined membership matrices of the first and second subsets. In ***Algorithm 4.1***, $\mathcal{G}$ gives numerator and denominator of V. Since the membership value of a data sample is independent of the other data samples, merging the membership matrix is the same as combining $\mathcal{G}$ of the first subset and $\mathcal{G}$ of the second subset. Instead of allocating a lot of space for $\mathcal{G}_1 \cup \mathcal{G}_2$, we can combine η_1 and η_2 without losing any information. This helps in space optimization. This optimization analogy works for the remaining subset, i.e., for all $s \in [3, s]$, where s is the number of subsets. Only the storage time $(\frac{1}{s})^{th}$ is effectively used for SFCC because the operations on the subset are performed one after another. In this way, we save a lot of space. Figure 4.3 shows a SIFCC workflow that uses SFCC to calculate all subsets of repository matrix (η) and cluster similarity matrix (V_{new}) of all subsets. It shows how the data sample is subdivided into different subdivisions and how the base cluster similarity matrix is randomly selected to cluster the Y_1. In addition, it demonstrates how to find cluster similarity matrix and

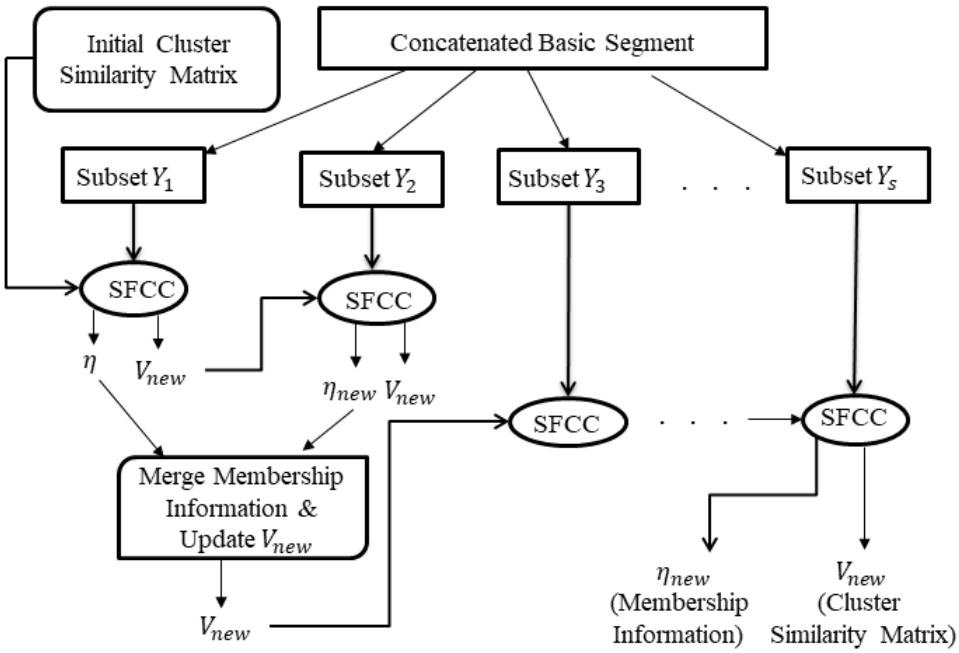


Figure 4.3: Workflow of SIFCC.

repository matrix for subset Y_1 using SFCC. Moreover, it additionally demonstrates how the cluster similarity matrix obtained from a subset Y_1 is utilized as the underlying cluster similarity matrix for the clustering of a subset Y_2 by utilizing SFCC. At this point, the cluster similarity matrix and repository matrix of the subsequent subsets are acquired by utilizing SFCC. Furthermore, the workflow demonstrates how the repository matrix of a subset Y_1 and a subset Y_2 are consolidated to process the new cluster similarity matrix. The new cluster similarity matrix is utilized as the initial cluster similarity matrix for the clustering of a subset Y_3. The same method is repeated to cluster the remaining subsets. These cluster similarity matrices are then fed as an input to the clustering of the third subset. The equivalent strategy is repeated for the clustering of all the consecutive subsets. The SIFCC works by partitioning the whole data, and the algorithm works until the clustering is performed on each and every subset. Thus, it ensures that it covers the entire object space represented by the whole dataset incrementally. Consequently, the outcomes created by SIFCC are equal to those delivered by performing the clustering over the whole data samples.

Map Function for SIFCC

The Map function for SIFCC presented in ***Algorithm 4.5***, calculates the membership degree of the data sample for each cluster and returns each independently. As a result, each Map function gives the same number of outcomes as the number of clusters. ***Algorithm 4.5*** shows the alignment of the tasks performed during the Map function to obtain μ_{lk} for each point y_l and cluster similarity matrix v_k using Line 6 of ***Algorithm 2.4***.

Algorithm 4.5 Map(y_l,V)

Input: y_l, V

Output: $< k, < \mu_{lk} >>$

1: **for** v_k **in** V **do**
2: k=index of cluster similarity matrix V.
3: μ_{lk}= membership degree of y_l w.r.t v_k.
4: **yield** $< k, < \mu_{lk} >>$
5: **end for**

4.3 Complexity Analysis

We figured out that the complexity analysis of our proposed SFCC and SIFCC algorithm is linear concerning space and time complexity in terms of the input data sample. The complexity of space is related to the amount of information stored in RAM throughout the calculation. Table 4.2 shows the complexity analysis of scalable fuzzy consensus clustering algorithms that are evolving in terms of variables.

Table 4.2: Complexity Analysis

Algorithm	**Time Complexity**	**Space Complexity**
SRSIO-FCM	$O(ncdt/w)$	$O(ncd/s)$
SFCC	$O(ncdt/w)$	$O(ncd)$
SIFCC	$O(ncdt/w)$	$O(ncd/s)$

Y is a concatenated basic segment with n number of data samples in the d dimensional space represented as $\sum_{i=1}^{r} c_i$ dimensional space and Y is randomly distributed into s number of partitions delineated as $Y = \{Y_1, Y_2, \ldots, Y_s\}$. c is the number of clusters, w is the number of slave nodes in Spark cluster. Each partition has (n/s) data samples. The number of iterations required for the termination is denoted as t. It may change, thus, for straightforwardness and clarity t represent the maximum number of iterations for one execution of SFCC and SIFCC. In these algorithms, we calculate the cost of membership degree during the Map stage. Each Map function calculates the membership degree of one data sample as to c cluster centers. Since each data sample has d-dimensions, the computation of each membership degree takes $O(d)$ time and $O(d)$ space. Each Map function takes $O(cd)$ time and $O(cd)$ space. The ReduceByKey function linearly adds the estimations of all the Map operations, corresponds to one cluster on each slave node, and joins the subsequent values on the master node. Every slave node takes $O(n'cd)$ time and $O(n'cd)$ space, where n' is the number of data samples fed as input to one slave node. We have assumed an equal distribution of jobs to the w slave nodes. Joining the outputs of Reduce action takes $O(cwd)$ time and $O(cd)$ space.

SRSIO-FCM partitions the dataset X into s equivalent subset delineated as $X = \{X_1, X_2, \ldots, X_s\}$ with n number of data samples, where every subset is of size (n/s). Since each subset is handled steadily, the time complexity is $O(ncdt/w)$ where w represents the number of slave nodes in a Spark cluster and d represents the dimensions of data samples. While the space complexity remains $O(ncd/s)$ because data corresponding to one subset is not held in the memory while processing the next subset.

SFCC executes Map and ReduceByKey tasks on the whole dataset. Every slave node processes the data samples by (n/w) in parallel. In this way, the Map stage takes an aggregate of $O(ncd/w)$ time and $O(ncd)$ space for every iteration on all the slave nodes. Expecting that SFCC keeps running for t iterations, the total time taken for the Map stage is $O(ncdt/w)$. Since Map results have been held in memory just for the term of one iteration of SFCC, hence the total space complexity for the Map stage of SFCC

is $O(ncd)$. The ReduceByKey action linearly adds (n/w) Map results, comparing to each cluster center in parallel, on every slave node. This takes $O(nd/w)$ time and $O(cd)$ space on a slave node. Since each slave node performs tasks on n/w data in the same amount of time and in parallel, thus the total required time is $O(nd/w)$. Each slave nodes give c outputs, each of dimension d, which are aggregated on the master node and added. This takes $O(cdw)$ time and $O(cdw)$ space. Along these lines, the total time complexity for the ReduceByKey stage is $O(ncdt/w)$ and space complexity is $O(ncdt/w)$. Accordingly, the time complexity of SFCC is $O(ncdt/w)$ and space complexity is $O(ncd)$ where $n >> w, c$.

SIFCC partitions the concatenated basic segment Y into s equivalent subsets with the end goal that $Y = \{Y_1, Y_2, \ldots, Y_s\}$, where every subset is of size (n/s). It executes SFCC over each one of these subsets sequentially. The time and space complexity of performing SFCC over every subset is $O(ncdt/sw)$ and $O(ncd/s)$, respectively. Since every one of the subsets is handled in a steady progression, the time complexity is $O(ncdt/w)$ while the space complexity remains $O(ncd/s)$ because data corresponding to one subset is not held in the memory while processing the next subset.

Table 4.2 demonstrates that SRSIO-FCM and SIFCC share similar time and space complexity. Nevertheless, this is not the situation. Since the time and space complexity for generating Y can be ignored, SIFCC has relatively low computational complexity similar to the SRSIO-FCM. Additionally, SFCC and SIFCC share a similar time complexity. This may look like both have similar run-time, but this is not the case. In SIFCC, we partition the whole dataset into various subsets. It performs clustering on every subset in a parallel manner and takes the lesser number of iterations. In contrast, SFCC performs clustering on whole data and not on the subsets.

4.4 Experimental Evaluation

The experiment compares the performance of the proposed SIFCC with SFCC and SRSIO-FCM algorithms in different datasets. Table 4.3, provides a detailed description of the dataset used in the experimental study.

Table 4.3: Dataset Description

Parameters	Datasets				
	Wine	**G2**	**Breast**	**SUSY**	**Reproduced-Dim32**
#instances	178	2048	699	5,000,000	1,00,000
#features	13	16	9	18	32
#classes	3	2	2	2	16
Dataset Size	10.8 KB	135.2 KB	19.9 KB	2.4 GB	12.4 GB

4.4.1 Datasets and Experimental Settings

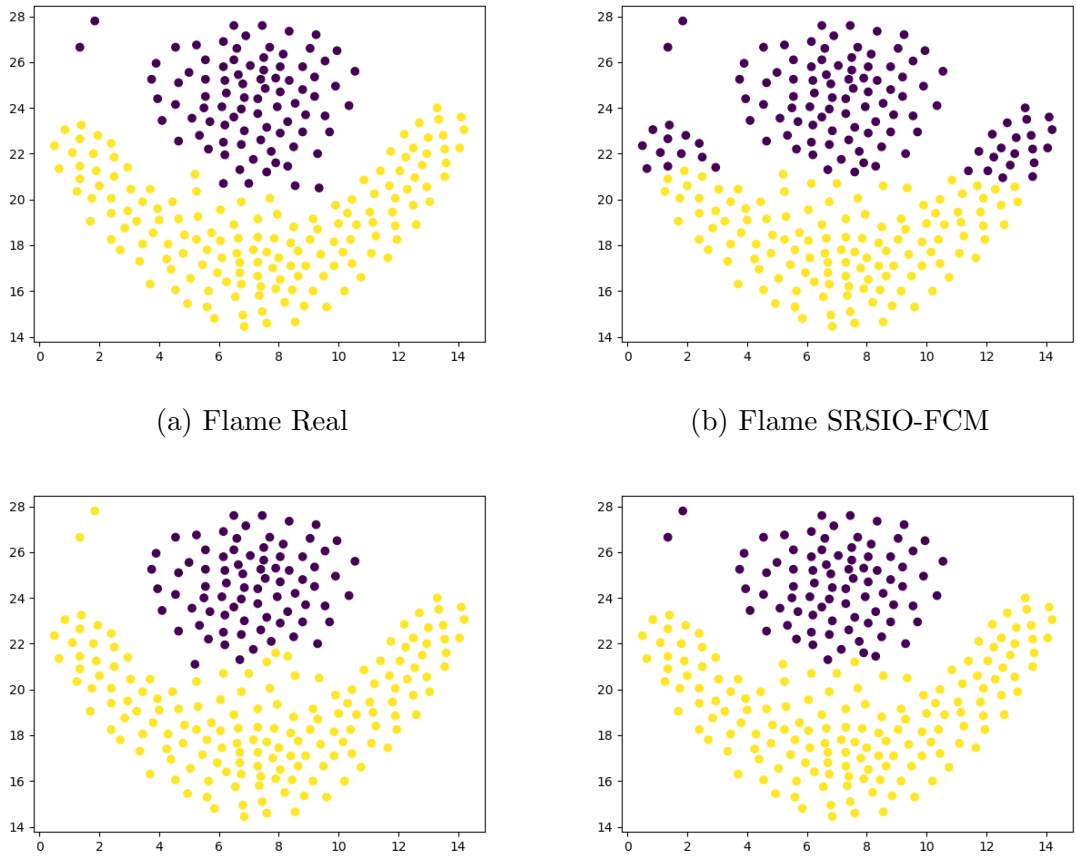
We have utilized three real-world datasets, and two big datasets openly accessible in our experiments. We have performed the execution of SIFCC, SFCC, and SRSIO-FCM algorithms on the following datasets. Table 4.3 demonstrates some of the features of these datasets for experimental analysis.

Wine: This data is a result of a chemical analysis of wines grown in the same region of Italy but from three different varieties. The analysis measures the amount of 13 ingredients contained in each of the three types of wine. There are 13 features for each data sample [242].

Breast: This breast cancer database was retrieved from the UCI machine learning repository. Instances range from 2 to 10. Each data sample is described with nine features [243].

G2: These are Gaussian cluster datasets with different cluster overlap and dimensions [244].

The SUSY and Reproduced-Dim32 datasets are explained in Section 3.4.3. For the dataset used in the experimental study, we set the fuzzification parameter values $p = 1.75$ and the stopping criteria $\epsilon = 0.001$. However, these values have proven to be effective for most datasets [245]. Table 4.3 shows the number of classes in the dataset. The proposed SIFCC and SFCC algorithms are implemented using Apache



(a) Flame Real

(b) Flame SRSIO-FCM

(c) Flame SFCC

(d) Flame SIFCC

Figure 4.4: Illustrative example of *flame* dataset.

Spark cluster, where HDFS is used as data storage. A detailed description of the experimental environment is presented in Section 3.4.2.

4.4.2 Experimental Results and Discussion

This section shows an example of a *flame* dataset to evaluate the performance of the proposed algorithms. We then discuss the performance assessment of SIFCC versus SRSIO-FCM and SFCC on three actual datasets and two large replicated datasets in terms of various metrics such as NMI, ARI, and F-score, respectively.

Illustrative Example

To illustrate the effectiveness of this algorithm, we presented a graph showing how the algorithm formed clusters from the *flame* dataset (as shown in Figure 4.4). A

synthetic dataset *flame* is used, which contains two types of two-dimensional data samples [175]. Figure 4.4(a), Figure 4.4(b), Figure 4.4(c), and Figure 4.4(d) show how the results of SIFCC are far better than those of SRSIO-FCM, SFCC, and closest to the ground truth. Note that these two classes are very close to each other and linearly inseparable, which poses challenges for clustering. Figure 4.4(a) shows how the data is divided into two classes in reality and reflects the ground truth. Figure 4.4(b) indicates how SRSIO-FCM performs with a number of clusters = 2. Because of the unusual shape of the data and the high proximity of the two classes, some yellow dots are incorrectly classified as violet dots. Figure 4.4(c) indicates how SFCC performs with number of subsets = 1 and the number of clusters = 2, very few violet dots are incorrectly classified as yellow dots. Figure 4.4(d) indicates how SIFCC performs with a number of subsets = 3 and the number of clusters = 2. We run the SRSIO-FCM algorithm ten times to obtain BSs by varying the number of clusters ranging from 2 to 11. Thereafter, these BSs are concatenated, which is used as input to SFCC/SIFCC algorithm. As we can see, SIFCC performs almost perfectly on the *flame* dataset, which shows the advantage of fuzzy consensus clustering over single clustering.

Evaluation of Clustering Performances

We evaluate the SIFCC, SFCC, and SRSIO-FCM algorithms on the three measures such as NMI, ARI, and F-score (discussed in Section 2.5.1). The number of subsets to perform clustering using SRSIO-FCM is 3 for Wine, Breast, and G2 datasets, 15 for SUSY, and 5 for Reproduced-Dim32 dataset. For each dataset, we run the SRSIO-FCM algorithm r number of times by varying the number of clusters starting from the actual number of clusters, where r is different for each dataset. This leads us to have r BSs as input. We concatenate the BSs to have the Y matrix, which is used as input for the SIFCC/SFCC algorithm. The number of subsets into which Y is divided is 3 for Wine and Breast datasets, 4 for G2, 15 for the SUSY dataset, and 5 for the Reproduced-Dim32 dataset. On the other hand, SFCC performs clustering on the whole dataset, i.e., considering complete datasets as 1 subset. A detailed discussion of the Wine, Breast, and G2 datasets is present in the subsequent section.

Wine: The number of BS here is 13, with the number of clusters ranging from 3 to

Table 4.4: Results of SRSIO-FCM, SFCC, and SIFCC on Wine Dataset.

Algorithm	**Measures**		
	NMI	**ARI**	**F-score**
SRSIO-FCM	0.4287	0.3711	0.7022
SFCC	0.4340	0.3906	0.7191
SIFCC	0.4375	0.3952	0.7247

15. The number of subsets in which Y was divided to perform SIFCC is 3 here. In Table 4.4, we have reported the results on the Wine dataset in terms of NMI, ARI, and F-score, respectively. Comparing the value of NMI for SIFCC, SFCC, and SRSIO-FCM, we analyze that NMI for SRSIO-FCM is low as compared to SFCC and SIFCC. The SIFCC performs better than SFCC and SRSIO-FCM. Now, on comparing ARI, we observe the same pattern, and it is significantly low for SRSIO-FCM and SFCC. While comparing the values of F-score it is significantly lower for SRSIO-FCM and SFCC than SIFCC. Therefore, when comparing the three measures, we concluded that SIFCC performs better than SFCC and SRSIO-FCM.

Breast: The number of basic segments here is 14, with the number of clusters ranging from 2 to 15. The number of subsets in which Y was divided to perform SIFCC is 3 here. In Table 4.5, we have reported the results on the Breast dataset in terms of NMI, ARI, and F-score, respectively. Comparing the value of NMI for SIFCC, SFCC, and SRSIO-FCM, we analyze that NMI for SRSIO-FCM is low as compared to SFCC and SIFCC. The SIFCC performs much better than SFCC and SRSIO-FCM. Now, on comparing ARI, we observe the same pattern, and it is significantly low for SRSIO-FCM and SFCC. While comparing the values of F-score, we found that the F-score is significantly lower for SRSIO-FCM and has almost equal values for SFCC and SIFCC. Therefore, when comparing the three measures, we concluded that SIFCC performs much better than SRSIO-FCM and SFCC in terms of NMI and ARI. Moreover, SIFCC and SFCC perform equally well but better than SRSIO-FCM in terms of F-score.

G2: The number of BSs here is 14, with the number of clusters ranging from 2 to 15. The number of subsets in which Y was divided to perform SIFCC is 4 here. In Table 4.6, we have reported the results on the G2 dataset in terms of NMI, ARI, F-score,

Table 4.5: Results of SRSIO-FCM, SFCC, and SIFCC on Breast Dataset.

Algorithm	**Measures**		
	NMI	**ARI**	**F-score**
SRSIO-FCM	0.7106	0.8178	0.9644
SFCC	0.7983	0.8823	0.9768
SIFCC	0.8007	0.8824	0.9767

Table 4.6: Results of SRSIO-FCM, SFCC, and SIFCC on G2 Dataset.

Algorithm	**Measures**		
	NMI	**ARI**	**F-score**
SRSIO-FCM	0.8397	0.9084	0.9765
SFCC	0.8482	0.9139	0.9779
SIFCC	0.8510	0.9158	0.9784

respectively. Comparing the value of NMI for SIFCC, SFCC, and SRSIO-FCM, we analyze that NMI for SRSIO-FCM is low as compared to SFCC and SIFCC. The SIFCC performs better than SFCC and SRSIO-FCM. Now, on comparing ARI, we observe the same pattern, and it is significantly low for SRSIO-FCM and SFCC. While comparing the values of F-score, we found that the F-score is significantly lower for SRSIO-FCM and SFCC. Therefore, when comparing the three measures, we concluded that SIFCC performs much better than SFCC and SRSIO-FCM in terms of NMI and ARI. Moreover, SIFCC performs better than SRSIO-FCM and SFCC in terms of F-score. Tables 4.4-4.6 prove how the SIFCC algorithm is better than SRSIO-FCM and SFCC. There may not be a great difference in the results of the datasets mentioned above, one reason being a very small size of data. The number of data samples is very small, so the improvement by SIFCC over SFCC does not get highlighted. When tested on Big Data, we will be able to see how the algorithm outperforms the SRSIO-FCM and SFCC.

Table 4.7: Results of SRSIO-FCM, SFCC, and SIFCC on SUSY Dataset.

Algorithm	**Measures**		
	NMI	**ARI**	**F-score**
SRSIO-FCM	0.0285	0.0423	0.6030
SFCC	0.0307	0.0437	0.6189
SIFCC	0.0310	0.0543	0.6208

4.4.3 Performance Evaluation on Big Data

The effectiveness of this algorithm is verified by testing it on Big Data. Any algorithm will be compelling to utilize, taking everything into account, in case it can manage huge amount of data. The ease of use and adaptability of an algorithm can be demonstrated when the algorithm is tested on Big Data. For that purpose, we have tested our proposed algorithms on two big datasets and the detailed descriptions of these datasets are presented next.

SUSY: In Table 4.7, we have reported the results on the SUSY dataset in terms of NMI, ARI, and F-score. The number of subsets used for the SRSIO-FCM and SIFCC algorithm is 15, and the number of BS here is 14 with the number of clusters ranging from 2 to 15. Comparing the value of NMI for SIFCC, SFCC, and SRSIO-FCM, we analyze that NMI for SRSIO-FCM is low as compared to SFCC and SIFCC. The SIFCC performs better than SFCC and SRSIO-FCM. Now, on comparing ARI, we observe the same pattern, and it is significantly low for SRSIO-FCM and SFCC. While comparing the values of F-score, we found that the F-score is significantly lower for SRSIO-FCM and SFCC than SIFCC.

Therefore, when comparing the three measures, we concluded that SIFCC performs much better than SFCC and SRSIO-FCM in terms of NMI and ARI. Moreover, SIFCC performs better than SRSIO-FCM and SFCC in terms of F-score. The SIFCC performs much better than the SFCC algorithm. Thus, the quality of the clusters obtained is high.

Reproduced-Dim32: In Table 4.8, we have presented the results for the Reproduced-Dim32 dataset in terms of NMI, ARI, and F-score. The number of subsets used for

Table 4.8: Results of SRSIO-FCM, SFCC, and SIFCC on Reproduced-Dim32 Dataset.

Algorithm	**Measures**		
	NMI	**ARI**	**F-score**
SRSIO-FCM	0.5268	0.1385	0.1875
SFCC	0.7019	0.2944	0.3127
SIFCC	0.8387	0.5791	0.5546

the SRSIO-FCM and SIFCC algorithms is 5, and the number of BS here is 10, with the number of clusters ranging from 16 to 25. Comparing the NMI value for SIFCC, SFCC, and SRSIO-FCM, we analyze that the NMI value for SRSIO-FCM is lower as compared to SFCC and SIFCC. Also, the NMI value of SFCC is lower than SIFCC. Now, on comparing ARI, we observe the same pattern, and it is lower for SRSIO-FCM as compared to SFCC and SIFCC. Moreover, the ARI value of SFCC is significantly lower than SIFCC. While comparing the values of F-score, we found that the F-score is significantly lower for SRSIO-FCM and SFCC than SIFCC.

Therefore, when comparing the three measures, we concluded SIFCC performs much better than SFCC and SRSIO-FCM in terms of NMI and ARI. Moreover, SIFCC achieves a better result than SRSIO-FCM and SFCC in terms of NMI. Thus, the quality of the clusters obtained is very high.

4.5 Summary

This chapter has proposed a SIFCC to address the challenges of the fuzzy cluster for processing large amounts of data. The SIFCC aims to identify a fuzzy consensus partition with overlapping clusters from a set of fuzzy partitions. The SIFCC uses SRSIO-FCM to generate BSs. SIFCC takes multiple BSs as input and executes SFCC with space optimization. Thus, the clustering results achieved with this approach are of significantly better quality. One of the key features of SIFCC is that it allows massive data to be clustered accurately without affecting the nature of the clustering results. Another important feature is that during the execution of the proposed

algorithm, it is unnecessary to store the membership matrix, which speeds up the execution of the proposed algorithm by reducing the execution time. This strategy improves the performance of Big Data clustering because the membership matrix is too large to even think about storing. Exact assessments on a massive dataset exhibited that SIFCC significantly outperformed the previously proposed FCC algorithm. The benefits that appeared in the tests demonstrate that SIFCC has incredible potential for use in massive data clustering. The proposed SIFCC is then investigated on real-life genome datasets, i.e., Single Nucleotide Polymorphisms (SNPs) sequences of soybean and rice plant species in Chapter 5.

Chapter 5

Design of Novel Scalable Feature Extraction Algorithm for Huge SNP Sequences with Application of Scalable Fuzzy Clustering Algorithms

In the previous Chapters 3 and 4, proposed scalable fuzzy clustering-based algorithms KSRSIO-FCM and SIFCC are tested on huge benchmark datasets which utilize the Apache Spark framework. Here, in this chapter, massive SNP data have been used to perform clustering using KSRSIO-FCM and SIFCC algorithms. The soybean SNP dataset was obtained from SoyBase[1] and rice datasets from SNP-Seek[2] database. Before clustering raw SNP sequences, there is a need to develop a method that can extract significant features from huge SNP sequences, which are used as input to the developed KSRSIO-FCM and SIFCC algorithms. Therefore, in this chapter, we have proposed a scalable SNP preprocessing approach for handling complex real-life, massive SNP data, which is discussed in detail in the subsequent section.

[1]https://soybase.org/
[2]https://snp-seek.irri.org/

5.1 Proposed Scalable Algorithm for Preprocessing of Huge SNP Sequences

This section describes the proposed scalable algorithm for preprocessing of huge SNP sequences. To propose a scalable SNP preprocessing algorithm, we followed the DNA approach discussed in [170] and applied this approach to huge SNP data to extract 12-dimensional numeric feature vectors. To make SNP preprocessing algorithm a scalable algorithm, we executed it on Apache Spark cluster and termed it as a scalable SNP preprocessing algorithm. The output obtained from scalable SNP preprocessing algorithm is used as input to the developed KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM algorithms. Furthermore, extracted 12-dimensional numeric feature vectors are applied to SIFCC and SFCC algorithms to cluster huge SNP sequences. ***Algorithm 5.1*** summarizes steps of scalable SNP preprocessing approach using the Apache Spark framework.

Algorithm 5.1 Scalable SNP Preprocessing Algorithm

Input: raw SNP data : *raw_snp.txt*

Output: processed snp data : *proc_snp.txt*

1: $y_1 = SparkContext.textFile(raw\_snp.txt).map(lambda\ z : numpy.array(z))$.

2: y_1=$rddTranspose(y_1)$.

3: y_1= $y_1.map(lambda\ z : SNP\_Preprocess(z))$

4: $y_1.saveAsTextFile(proc\_snp.txt)$

The input given is a raw SNP dataset containing nucleotides A, G, T, C. The output is a feature vector of the given input dataset that is a file containing 12-dimensional numeric feature vectors. In Line 1 of ***Algorithm 5.1***, the data is read into a resilient distributed dataset (RDD) from the Hadoop using pyspark class, i.e., *SparkContext.textFile* (discussed in Section 2.2). The obtained RDD is transposed using the *rddTranspose* function in Line 2. Then a map function is used to accomplish the preprocessing task. In Line 3 of ***Algorithm 5.1***, the map function allows calling the $SNP\_Preprocess$ function to each row of the transposed RDD. Finally, the

obtained RDD after the map function is saved using Line 4 of ***Algorithm 5.1***.

The proposed scalable SNP preprocessing approach extracts features of SNP sequences in three sets of numerical parameters: the first parameter is a calculation of the length of sequences, the second parameter is the total distances of each nucleotide base to the first nucleotide, and the third parameter is the variance of distance for each nucleic base [170]. Each set of a numerical parameter is not sufficient to denote a specific SNP sequence. Thus, the combination of all of the three sets of the numerical parameter, which contains 12-dimensional numeric feature vectors is used to characterize similarities between SNP sequences. The details of SNP feature extraction are given in ***Algorithm 5.2*** discusses the *SNP_Preprocess* algorithm, which is called by *Scalable SNP Preprocessing Algorithm* (given in ***Algorithm 5.1***).

Algorithm 5.2 *SNP_Preprocess*

Input: z; z is a numpy array.

Output: $numpy.array[\ell_A, \mathcal{T}_A, D_A, \ell_G, \mathcal{T}_G, D_G, \ell_T, \mathcal{T}_T, D_T, \ell_C, \mathcal{T}_C, D_C]$.

1: Let i denote nucleotide A, G, T, C.
2: **for** x **in** z **do**
3: **if** x is i **then**
4: **Increase** the count of nucleotide i i.e., ℓ_i++
5: **end if**
6: **end for**
7: Calculate total distance T_i using Eq. (2.31).
8: Calculate the variance of distance D_i using Eq. (2.32).

After this, extracted 12-dimensional numeric feature vectors are passed as an input to the developed algorithms, i.e., KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM algorithms. Furthermore, it is applied to the developed scalable fuzzy consensus based SIFCC and SFCC algorithm to cluster huge SNP data. The working of the *SNP_Preprocess* algorithm is shown in terms of various steps by using an example given in the subsequent subsection.

Sequences	Positions								
	1	2	3	4	5	6	7	8	9
Sequence1	G	A	A	T	G	C	T	G	G
Sequence2	T	G	C	T	G	T	T	A	A
Sequence3	T	A	C	T	G	A	T	C	G
Sequence4	G	C	G	A	T	A	T	G	T
Sequence5	T	T	A	C	G	G	A	T	G

Table 5.1: Example of SNP sequences

5.1.1 Step I: Calculation of length of sequence

The total numbers of A, T, G, and C are calculated for each sequence, represented as ℓ_A, ℓ_T, ℓ_G, and ℓ_C. The $SNP\_Preprocess$ algorithm (given in ***Algorithm 5.2***) working is being described using a very first step in which the length of the sequence of each nucleotide is calculated. Here an illustration is presented by considering an example of five sequences shown in Table 5.1. The output obtained after preprocessing of these five sequences are shown in Figure 5.1, where the first column represents the sequence number, and the numbers present in the rest of the columns represent the 12-dimensional numeric feature vectors $\langle \ell_A, T_A, D_A, \ell_G, T_G, D_G, \ell_T, T_T, D_T, \ell_C, T_C, D_C \rangle$. The total number of A, G, T, and C i.e., ℓ_A, ℓ_G, ℓ_T, and ℓ_C present in sequence1 $\langle G\ A\ A\ T\ G\ C\ T\ G\ G \rangle$ are $2, 4, 2$, and 1, respectively. Likewise, the value of ℓ_A, ℓ_G, ℓ_T, and ℓ_C for the other four sequences are shown in Figure 5.1. After calculating the total number of nucleotides (A, G, T, and C) in each sequence, the total distances of each nucleotide base to the first nucleotide is calculated, which is presented next.

```
1, 2, 5, 0.25, 4, 23, 9.67, 2, 11, 2.25, 1, 6, 0
2, 2, 17, 0.25, 2, 7, 2.25, 4, 18, 1.625, 1, 3, 0
3, 2, 8, 4, 2, 14, 4, 3, 12, 6, 2, 11, 6.25
4, 2, 10, 7.25, 3, 12, 6.5, 3, 21, 6.66, 1, 2, 0
5, 2, 10, 4, 3, 20, 2.89, 3, 11, 9.56, 1, 4, 0
```

Figure 5.1: Preprocessing result of SNP sequences present in Table 5.1

5.1.2 Step II: Total distances of each nucleotide base to the first nucleotide

The second numerical parameter, i.e., the total distance of each nucleotide base to the first nucleotide $\mathcal{T}_i$ is calculated using Eq. (2.31). Then, add the position values corresponding to each nucleotide as shown in Table 5.1, as nucleotide G appears at 1, 5, 8, and 9. So, the value of $\mathcal{T}_G$ is calculated as follows:

$$T_G = 1 + 5 + 8 + 9 = 23$$

Likewise, the value of $\mathcal{T}_A, \mathcal{T}_T$, and $\mathcal{T}_C$ for sequence1 and the total distance of each nucleotide base to the first nucleotide for all other four sequences is shown in Figure 5.1. After calculating the total distances of each nucleotide base to the first nucleotide, the variance of distance for each nucleic base is calculated, which is presented next.

5.1.3 Step III: Variance of distance for each nucleic base

The third numerical parameter is the variance of distance for each nucleic base D_i is calculated using Eq. (2.32). The first step is to compute d_i using $d_i = \frac{T_i}{\ell_i}$. The value of d_G in sequence1 is calculated as follows:

$$d_G = \frac{T_G}{\ell_G} = \frac{23}{4} = 5.75$$

The second step is to compute the variance of distance for each nucleic base, i.e., D_i, for example, the value of D_G in sequence1 is 9.67, calculations of D_G is as follows:

$$D_G = \frac{[(1-5.75)+(5-5.75)+(8-5.75)+(9-5.75)]}{4} = 9.67$$

Hence, the result obtained from sequence1 which contains a 12-dimensional numeric feature vector is shown below:

$$\langle 2, 5, 0.25, 4, 23, 9.67, 2, 11, 2.25, 1, 6, 0 \rangle$$

The results of all the other four sequences are shown in Figure 5.1. The significant characteristic of the proposed scalable SNP preprocessing algorithm is that it takes raw SNP sequences as input and produces 12-dimensional numeric feature vectors as output.

5.2 Experimental Evaluation on SNP Datasets

In this section, we present the experimental results on various SNP datasets applied to developed scalable fuzzy clustering algorithms. First, we have evaluated the performance of developed scalable fuzzy based KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM algorithms on SNP datasets. Second, we have tested the results of SNP data on developed scalable fuzzy consensus-based SIFCC and SFCC algorithms. These clustering methods take the preprocessed 12-dimensional numeric feature vectors of huge SNP sequences as input and produce output in terms of clusters. Figure 5.2 summarizes the preprocessing and clustering of SNP sequences diagrammatically. It shows how KSRSIO-FCM/SIFCC takes 12-dimensional numeric feature vectors as input and then huge SNP sequences are partitioned randomly across various nodes using Apache Spark cluster.

5.2.1 Datasets and Experimental Settings

In this section, we have analyzed the exhibition of KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM algorithm on SNP-seek, MAGIC-rice, and 248Entries of SNP datasets. A detailed description of all datasets is presented in Section 2.6.1. The experimental evaluation is performed on Apache Spark cluster. The detailed description of Apache Spark setup is presented in Section 3.4.2. The High-Performance Computing (HPC) server machine is added in a cluster to preprocess genome data with the following configuration; A total number of cores: 32, Total memory: 187 GB, Total disk: 12 TB.

5.2.2 Experimental Results and Discussion on Scalable Fuzzy Clustering Algorithms

In this section, we discuss the effectiveness of KSRSIO-FCM in comparison with KSLFCM, SRSIO-FCM, and SLFCM evaluated on SNP data in terms of the Silhouette Index (SI) and Davies Bouldin Index (DBI). The detailed description of SI and DBI

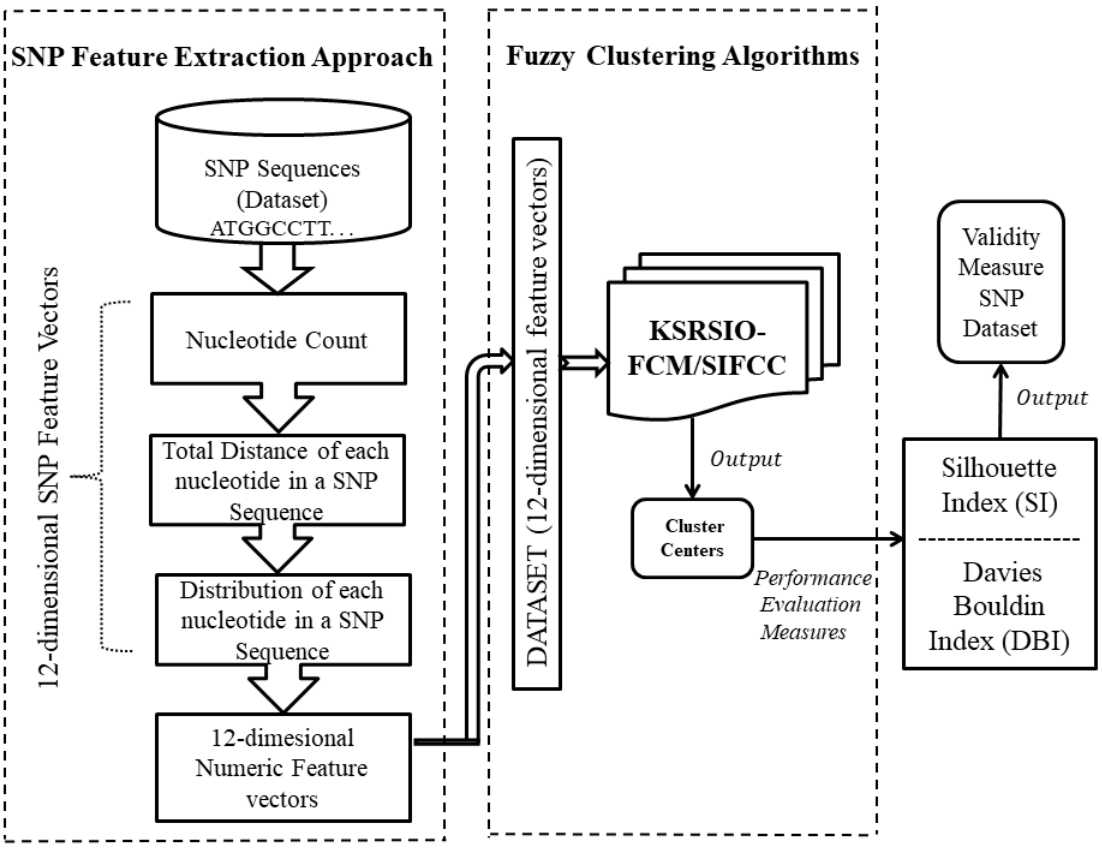


Figure 5.2: Workflow of KSRSIO-FCM and SIFCC algorithms with the preprocessing steps of huge SNP sequences.

is presented in Section 2.5.2. The KSRSIO-FCM and KSLFCM algorithms take 12-dimensional numeric feature vectors extracted after preprocessing of the huge SNP sequences as input and then both the algorithms cluster huge SNP sequences at high speed with high accuracy. A detailed description of KSRSIO-FCM and KSLFCM algorithms are already presented in Chapter 3.

To show the effectiveness of the KSRSIO-FCM algorithm in comparison with KSLFCM, SRSIO-FCM, and SLFCM, we have presented the diagrammatic representation, which shows how the algorithm creates clusters out of the SNP data of soybean 31 sequences. These soybean 31 sequences[3] of SNP data contains 6,289,747 SNPs [178]. A soybean dataset consisting of 31 sequences is used, which includes samples of two categories, i.e., wild and cultivated [246]. The comparison between

[3]http://chibba.pgml.uga.edu/snphylo/

the four algorithms are depicted in Figure 5.3 and Figure 5.4, which clearly show how the results of KSRSIO-FCM are better than those of KSLFCM, SRSIO-FCM, and SLFCM, respectively.

Figure 5.3: Cluster formation of soybean 31 sequences for KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM with the number of clusters = 5

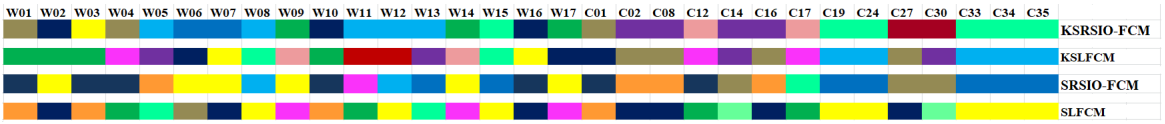


Figure 5.4: Cluster formation of soybean 31 sequences for KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM with the number of clusters = 10

From Figure 5.3, we can infer that the clusters formed by KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM are well separated into five respective groups (i.e., different colors represent different clusters). But, the clusters formed by KSLFCM, SRSIO-FCM, and SLFCM creates three overlapped clusters consists of wild and cultivated data samples. The number of data samples of wild and cultivated categories is perfectly clustered by KSLFCM, SRSIO-FCM, and SLFCM are 9, 9, and 7, respectively. On the other hand, KSRSIO-FCM only creates two overlapped clusters and 15 data samples of wild and cultivated categories are perfectly clustered. Likewise, from Figure 5.4, we can infer that the KSRSIO-FCM formed a total of ten clusters. Out of ten clusters, two clusters are overlapping and 8 clusters are perfectly formed consist of 22 data samples of wild and cultivated categories. On the other hand, KSLFCM generates three overlapped clusters out of a total of 10 clusters and SRSIO-FCM forms a total of 8 clusters with 3 overlapped clusters, and SLFCM forms a total of 7 clusters out of which 5 clusters are overlapped consist of wild and cultivated data samples. Hence, we can conclude that the KSRSIO-FCM outperforms KSLFCM, SRSIO-FCM, and SLFCM, respectively. The effectiveness of our proposed algorithm can be proved by

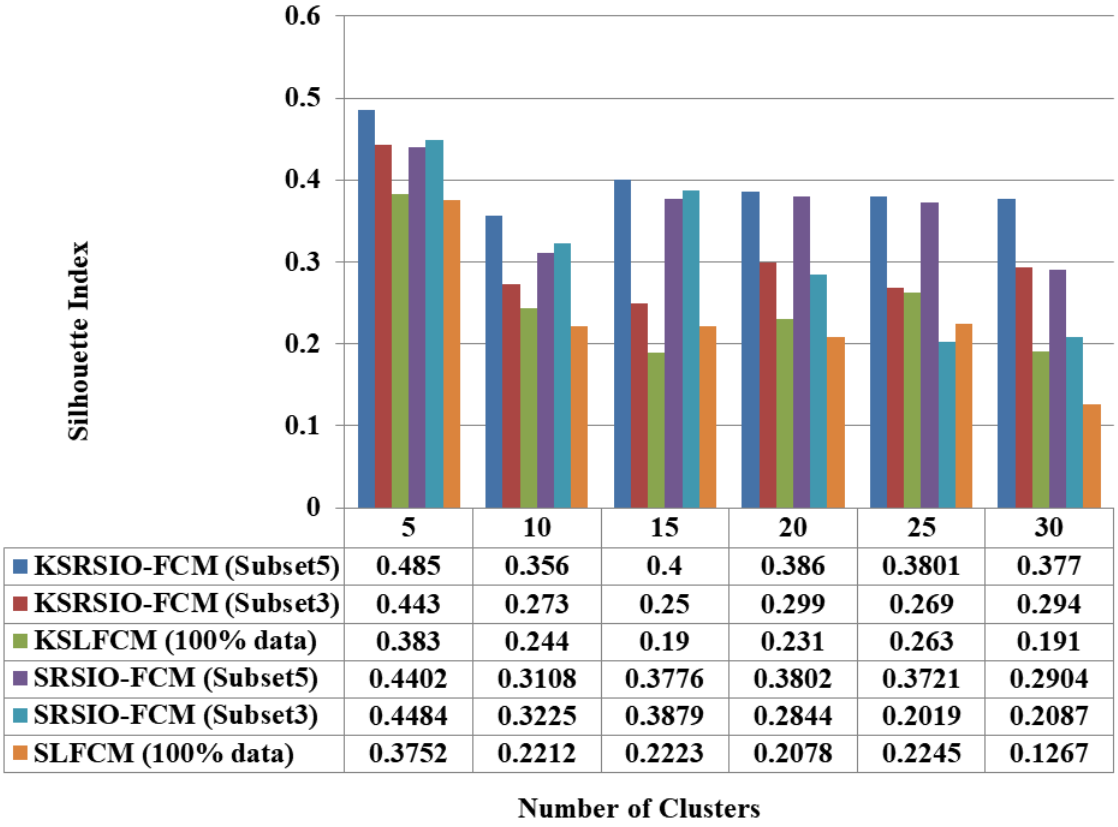
testing it on huge SNP data. For that purpose, we have tested our proposed algorithms on a huge SNP dataset in the subsequent subsection.

5.2.3 Clustering Performance of Scalable Fuzzy Clustering Algorithms

This section presents the discussion of the effectiveness of KSRSIO-FCM in comparison with KSLFCM, SRSIO-FCM, and SLFCM evaluated on three SNP datasets as per the estimates, such as SI and DBI, respectively. We perform clustering with the number of subsets 5 and 3, where the subset means the entire data is data divided into chunks. The clustering is performed on the number of clusters 5, 10, 15, 20, 25, and 30, respectively. In this section, subset5 and subset3 depict the number of subsets equal to 5 and 3, respectively. Likewise, the cluster5, cluster10, cluster15, cluster20, cluster25, and cluster30 depicts the number of clusters equal to 5, 10, 15, 20, 25, and 30, respectively. Subset5 and subset3 indicate that the entire dataset is divided into 5 chunks and 3 chunks, respectively, using KSRSIO-FCM and SRSIO-FCM algorithms. On the contrary, the KSLFCM and SLFCM perform the clustering on whole data. We have compared the performance of kernelized scalable algorithms, i.e., KSRSIO-FCM with SRSIO-FCM, KSLFCM, and SLFCM.

Clustering performances on the SNP-seek rice dataset:

Figure 5.5 highlights the results of the SNP-seek rice dataset in terms of SI that demonstrates the quality of clustering. Subsequently, a superior clustering would bring about higher SI. Observing the values of SI, SRSIO-FCM has obtained a lower value, whereas the KSRSIO-FCM achieved a higher value for subset5. Also, the KSRSIO-FCM achieved the highest value of SI for cluster5 of subset5. Also, we analyzed that SI obtained by KSRSIO-FCM for subset3 is higher for clusters 20, 25, and 30 in comparison with SRSIO-FCM. The figure shows that the estimation of SI for KSLFCM is higher for all the clusters except cluster15 in comparison with SLFCM for the SNP-seek rice dataset. While the SI value is essentially lower for KSLFCM and SLFCM when compared with KSRSIO-FCM in most of the clusters. Along these lines, we

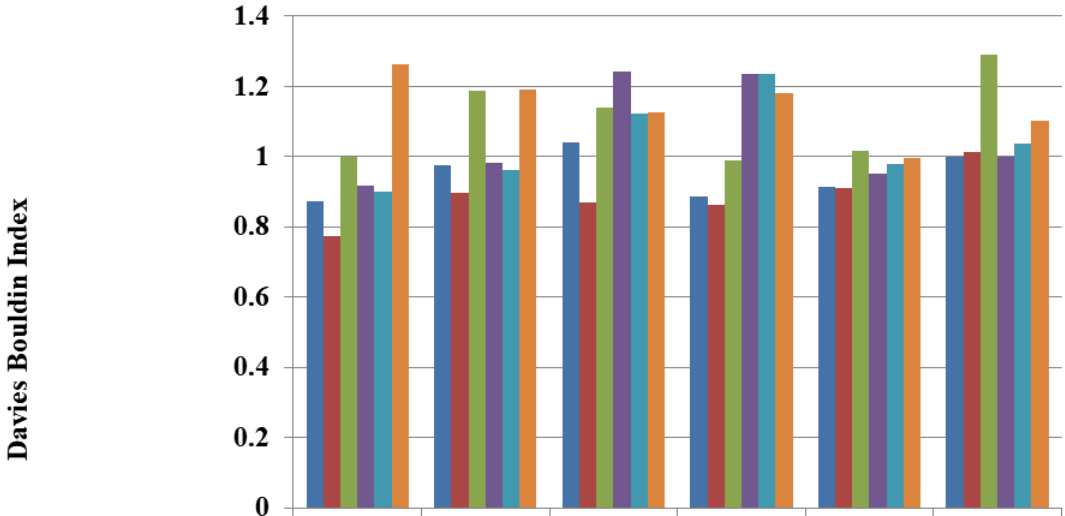


	5	10	15	20	25	30
KSRSIO-FCM (Subset5)	0.485	0.356	0.4	0.386	0.3801	0.377
KSRSIO-FCM (Subset3)	0.443	0.273	0.25	0.299	0.269	0.294
KSLFCM (100% data)	0.383	0.244	0.19	0.231	0.263	0.191
SRSIO-FCM (Subset5)	0.4402	0.3108	0.3776	0.3802	0.3721	0.2904
SRSIO-FCM (Subset3)	0.4484	0.3225	0.3879	0.2844	0.2019	0.2087
SLFCM (100% data)	0.3752	0.2212	0.2223	0.2078	0.2245	0.1267

Figure 5.5: Silhouette Index of SNP-seek rice dataset

can conclude that KSRSIO-FCM performs better than KSLFCM, SRSIO-FCM, and SLFCM in terms of SI values for the SNP-seek rice dataset.

Conversely to the SI, the DBI is not bounded within a given range. As a general rule, the lower the DBI value is, the better the clustering result will be. In Figure 5.6, we have reported the results on the SNP-seek rice dataset in terms of DBI. The value achieved by KSRSIO-FCM is much better than SRSIO-FCM on almost all the clusters for subset5. Moreover, KSRSIO-FCM attained a very low value on all the clusters for subset3. Additionally, KSRSIO-FCM achieves the remarkable value of DBI for cluster5 of subset3. On comparing KSLFCM with SLFCM, the DBI values for most of the clusters are lower for KSLFCM. Therefore, comparing DBI, we conclude that KSRSIO-FCM performs much better than KSLFCM, SRSIO-FCM, and SLFCM in terms of SI for subset5 and DBI for subset3. Overall, we can say that KSRSIO-FCM



	5	10	15	20	25	30
KSRSIO-FCM (Subset5)	0.8735	0.9768	1.0399	0.8854	0.9156	1.0009
KSRSIO-FCM (Subset3)	0.7741	0.8953	0.8689	0.8614	0.9117	1.012
KSLFCM (100% data)	0.9995	1.1868	1.1399	0.9895	1.0156	1.2909
SRSIO-FCM (Subset5)	0.916	0.9809	1.2426	1.2351	0.9531	0.9991
SRSIO-FCM (Subset3)	0.8999	0.9627	1.1225	1.2363	0.9774	1.0363
SLFCM (100% data)	1.2616	1.1914	1.1269	1.1793	0.9976	1.1032

Number of Clusters

Figure 5.6: Davies Bouldin Index of SNP-seek rice dataset

for subset3 or subset5 for each cluster performs better than KSLFCM. As depicted in the figure, the estimation of DBI for KSRSIO-FCM is significantly better than SRSIO-FCM and SLFCM. In this way, we can conclude that KSRSIO-FCM performs better than KSLFCM, SRSIO-FCM, and SLFCM in terms of DBI values for the SNP-seek rice dataset.

Clustering performances on the MAGIC-rice dataset:

Figure 5.7 shows the results of the MAGIC-rice dataset in terms of SI. On comparing SI, SRSIO-FCM has attained the lowest SI value on all the clusters compared to the SI values achieved on KSRSIO-FCM for subset5 and subset3. Besides this, the SI of KSRSIO-FCM achieves a higher value for cluster15 of subset5. Furthermore, the figure shows that the estimation of SI for KSLFCM and SLFCM is close for various clusters. While the SI value is essentially lower for KSLFCM and SLFCM when

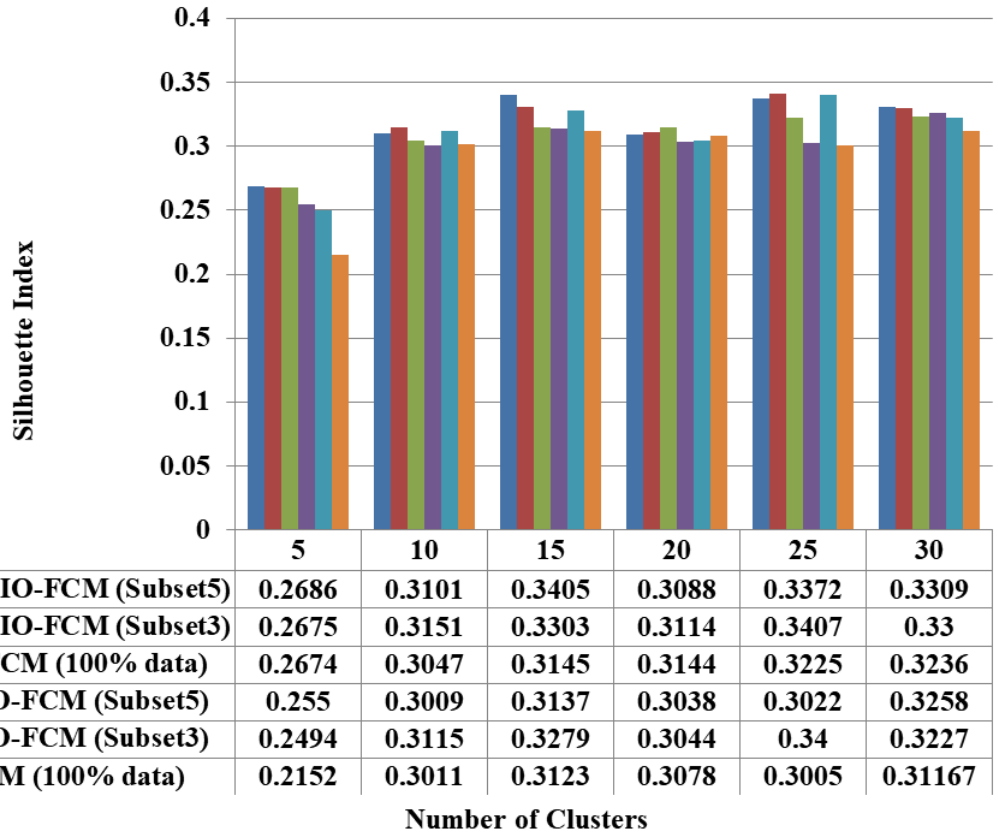
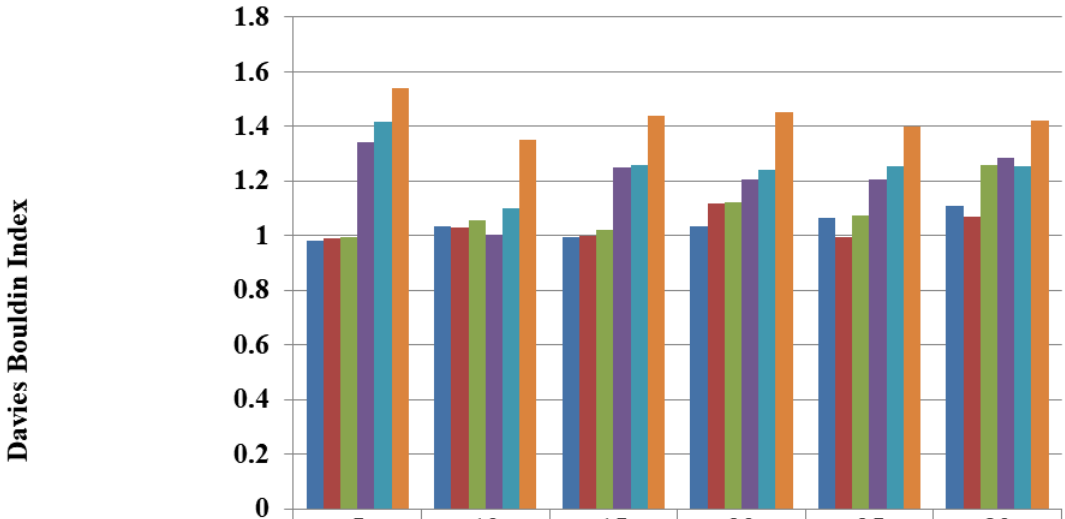


Figure 5.7: Silhouette Index of MAGIC-rice dataset

compared with KSRSIO-FCM. Along these lines, we can conclude that KSRSIO-FCM performs better than KSLFCM, SRSIO-FCM, and SLFCM in terms of SI values for the MAGIC-rice dataset.



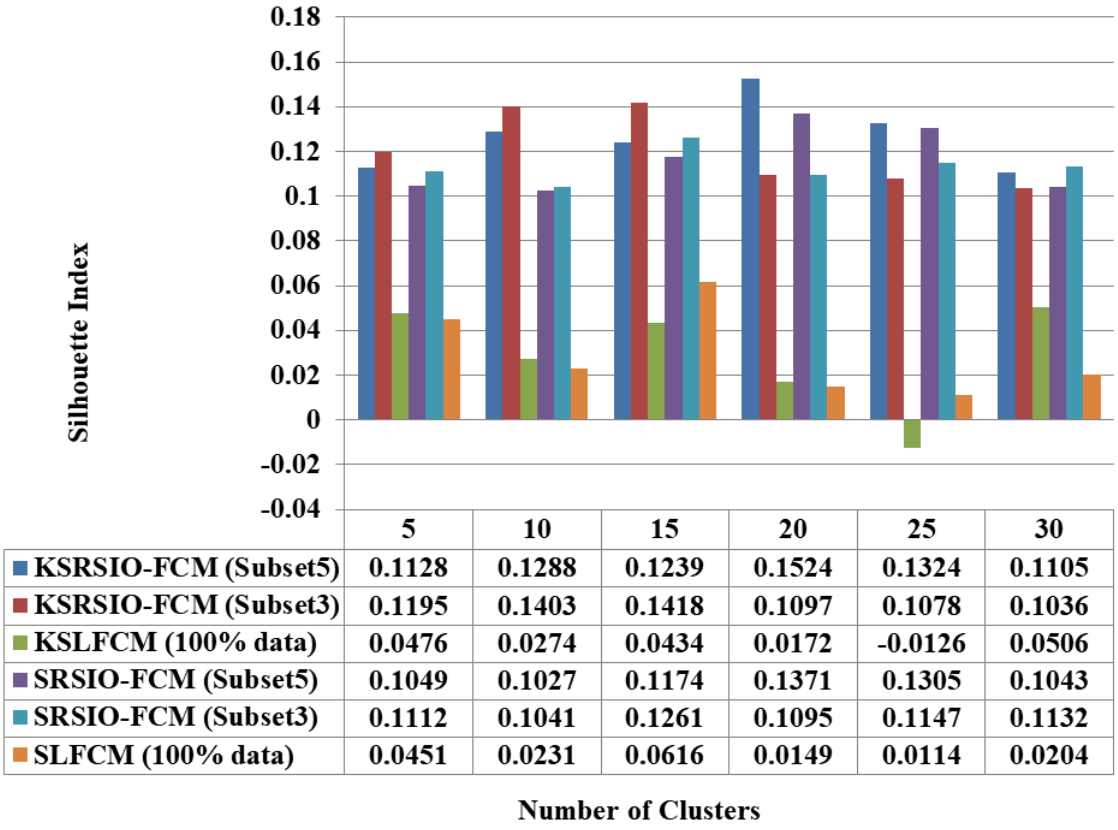
	5	10	15	20	25	30
KSRSIO-FCM (Subset5)	0.9799	1.0327	0.9963	1.0347	1.0656	1.1085
KSRSIO-FCM (Subset3)	0.9923	1.0283	0.9998	1.1164	0.9945	1.07
KSLFCM (100% data)	0.9942	1.0543	1.0204	1.1219	1.0726	1.2573
SRSIO-FCM (Subset5)	1.3415	1.0016	1.2497	1.2066	1.206	1.286
SRSIO-FCM (Subset3)	1.4152	1.1013	1.2598	1.2417	1.2539	1.2546
SLFCM (100% data)	1.5416	1.3512	1.4375	1.4531	1.3974	1.4197

Figure 5.8: Davies Bouldin Index of MAGIC-rice dataset

Similarly, for DBI, as shown in Figure 5.8, SRSIO-FCM has obtained a higher value compared to KSRSIO-FCM on subset5 and subset3 for the different number of clusters. Additionally, KSRSIO-FCM achieves the remarkable value of DBI for cluster5 of subset5. Therefore, we conclude that KSRSIO-FCM performs much better than SRSIO-FCM. Furthermore, the figure shows that the estimation of DBI for KSLFCM is significantly better than SLFCM. In this way, we can conclude that KSRSIO-FCM performs much better than KSLFCM, SRSIO-FCM, and SLFCM in terms of DBI values for the MAGIC-rice dataset.

Clustering performances on the 248Entries rice dataset:

Figure 5.9 shows the results of the 248Entries rice dataset in terms of SI.

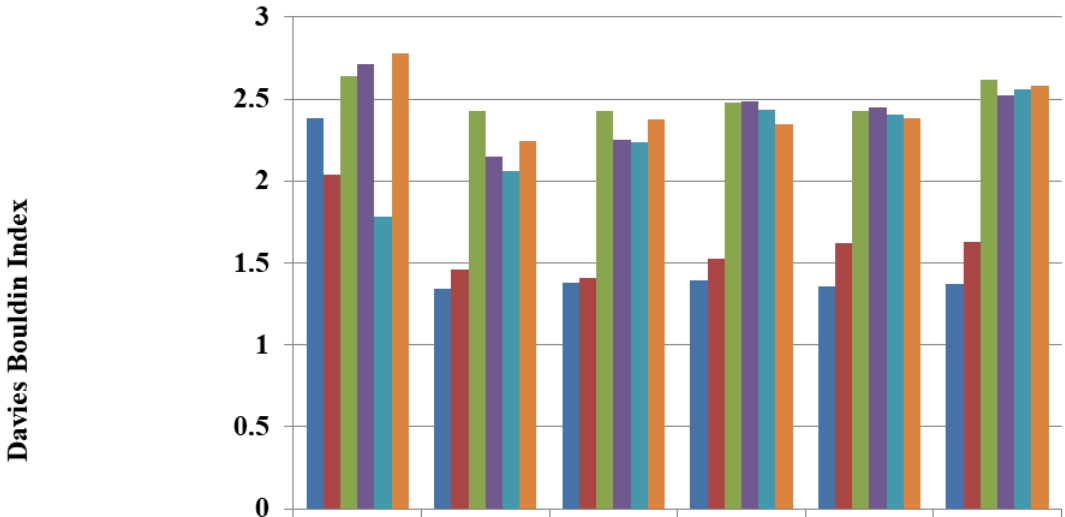


	5	10	15	20	25	30
KSRSIO-FCM (Subset5)	0.1128	0.1288	0.1239	0.1524	0.1324	0.1105
KSRSIO-FCM (Subset3)	0.1195	0.1403	0.1418	0.1097	0.1078	0.1036
KSLFCM (100% data)	0.0476	0.0274	0.0434	0.0172	-0.0126	0.0506
SRSIO-FCM (Subset5)	0.1049	0.1027	0.1174	0.1371	0.1305	0.1043
SRSIO-FCM (Subset3)	0.1112	0.1041	0.1261	0.1095	0.1147	0.1132
SLFCM (100% data)	0.0451	0.0231	0.0616	0.0149	0.0114	0.0204

Figure 5.9: Silhouette Index of 248Entries rice dataset

On comparing SI, SRSIO-FCM has attained the lowest SI values on most of the clusters corresponded to the SI values achieved by KSRSIO-FCM for subset5 and subset3. Besides this, KSRSIO-FCM attains the higher value of SI for cluster20 of subset5. The SI value of KSLFCM is better than SLFCM except for cluster15. The estimation of SI for various subsets of KSRSIO-FCM is better than SLFCM. Furthermore, the figure shows that the SI values obtained by SRSIO-FCM and SLFCM are close for the 248Entries dataset. Additionally, the estimation of SI for different subsets of KSRSIO-FCM is better than KSLFCM. While speaking about the difference in SI values among various clusters, SI for KSLFCM (100% data) is comparatively smaller than SI values achieved with KSRSIO-FCM on subset3 and subset5 with varying number of clusters.

Moreover, we observed that KSRSIO-FCM has attained positive SI for each cluster, whereas KSLFCM obtained negative SI values for cluster25. Along these lines, we can conclude that KSRSIO-FCM performs better than KSLFCM, SRSIO-FCM, and SLFCM in terms of SI values for the 248Entries rice dataset.



	5	10	15	20	25	30
KSRSIO-FCM (Subset5)	2.3866	1.341	1.38172	1.3905	1.3605	1.3703
KSRSIO-FCM (Subset3)	2.04283	1.4634	1.4071	1.523	1.6242	1.6285
KSLFCM (100% data)	2.6384	2.4247	2.4313	2.4792	2.4302	2.6162
SRSIO-FCM (Subset5)	2.716	2.1508	2.2501	2.4871	2.4506	2.523
SRSIO-FCM (Subset 3)	1.7816	2.0639	2.2347	2.4368	2.4042	2.5617
SLFCM (100% data)	2.7816	2.2409	2.3732	2.3492	2.3823	2.5821

Figure 5.10: Davies Bouldin Index of 248Entries rice dataset

Similarly, for DBI, as shown in Figure 5.10, SRSIO has obtained a higher value compared to KSRSIO-FCM on each cluster except cluster5 of subset3. While speaking about the difference in DBI values among various clusters of subset5 and subset3, DBI for KSLFCM (100% data) is comparatively higher than DBI values achieved with KSRSIO-FCM. Furthermore, KSRSIO-FCM achieves the remarkable value of DBI for subset5 and subset3. Moreover, KSLFCM has attained lower DBI values than SLFCM. Though the DBI is essentially lower for SLFCM for some of the clusters when compared with KSLFCM, the estimation of DBI for different subsets of KSRSIO-FCM are

Table 5.2: Run-time analysis (in seconds) of KSRSIO-FCM and KSLFCM algorithms.

c	**Datasets**								
	SNP-seek rice			MAGIC-rice			248Entries rice		
	Subset5	Subset3	100% data	Subset5	Subset3	100% data	Subset5	Subset3	100% data
5	80	76	96	120	120	300	74	56	76
10	72	70	120	900	720	960	68	76	80
15	220	190	310	960	1680	1980	207	160	260
20	240	200	340	1268	1140	1560	196	114	182
25	260	275	365	2460	2400	3615	154	178	192
30	300	310	390	2340	6300	6600	120	277	290

better than SLFCM. In this way, we can conclude that KSRSIO-FCM performs better than KSLFCM, SRSIO-FCM, and SLFCM in terms of DBI values for the 248Entries dataset.

Table 5.2 tabulates the run-time analysis of KSRSIO-FCM and KSLFCM algorithms. In this table, c represents the number of clusters. The KSRSIO-FCM algorithm is performed on subset5 and subset3, whereas KSLFCM works on the whole dataset (100% data). In any case, the run-time analysis of KSRSIO-FCM and KSLFCM would likewise rely upon the total number of nodes and their configuration. Here, the total number of nodes is 6 and cores are 52. According to the run-time analysis given in Table 5.2, the KSRSIO-FCM takes less time in comparison with KSLFCM for computation. We have also tested proposed SIFCC and SFCC algorithms for the clustering of massive SNP datasets. The experimental analysis of SNP datasets applied on SIFCC and SFCC is presented in a subsequent section.

5.2.4 Experimental Results and Discussion on Scalable Fuzzy Consensus Clustering

In the experiments, we analyze the performance of SIFCC and SFCC using the SI and DBI indexes. A detailed description of SIFCC and SFCC algorithms is already presented in Chapter 4. We analyze the exhibition of SIFCC and SFCC algorithms on SNP50K Wm82.a1, SNP-seek, MAGIC-rice, and 248Entries of SNP datasets. A detailed description of all datasets is presented in Section 2.6.1. Figure 5.2 summarizes

the preprocessing and clustering of SNP sequences diagrammatically. It shows how SIFCC takes 12-dimensional numeric feature vectors as input and then huge SNP sequences are partitioned randomly across various nodes using Apache Spark cluster.

To show the effectiveness of the SIFCC algorithm, we present the diagrammatic representation that shows how the algorithm creates clusters out of the SNP dataset of soybean 31 sequences. For the experimental study, we have used soybean 31 sequences[4] of SNP data that consists of samples from two categories, i.e., wild and cultivated [246, 178]. Figure 5.11 indicates how SIFCC performs better than SFCC with 5 clusters. As depicted in this figure, SIFCC formed five clusters efficiently with less overlapping of wild and cultivated data samples, i.e., out of five clusters only in two clusters the overlapping among data samples exists, whereas SFCC formed five clusters out of which data samples in three clusters are overlapping. Therefore, on comparing both the algorithms, we conclude that the clustering results obtained from SIFCC are better than the SFCC. The effectiveness of our proposed algorithm can be proved by testing it on huge SNP data. For that purpose, we have tested our proposed algorithms on a huge SNP dataset in the subsequent subsection.

Figure 5.11: Cluster formation of soybean 31 sequences for SIFCC and SFCC with the number of clusters = 5.

5.2.5 Clustering performance of Scalable Fuzzy Consensus Clustering Algorithms

This section presents the discussion of the effectiveness of SIFCC in comparison with SFCC evaluated on four SNP datasets (soybean and rice) as per the estimates, such as SI and DBI, respectively. The estimations of SI and DBI for SIFCC, on various subsets of three SNP datasets in comparison with SFCC, are shown in tables.

[4]http://chibba.pgml.uga.edu/snphylo/

Table 5.3: Values of the SI in the range of cluster = 2....12 for all the four SNP datasets using the SIFCC and SFCC algorithm (Entries in boldface indicate the optimal values for respective indices)

Cluster	**Datasets**							
	SoySNP50k Wm82.a1		SNP-seek rice		MAGIC-rice		248Entries rice	
	SIFCC	**SFCC**	**SIFCC**	**SFCC**	**SIFCC**	**SFCC**	**SIFCC**	**SFCC**
2	**0.3444**	0.3434	0.2090	0.2090	0.0860	0.0860	**0.1253**	0.1253
3	0.2079	0.2078	0.3577	0.3577	0.3121	0.3074	0.0438	0.0348
4	0.2784	0.1211	0.3186	0.3100	**0.6484**	0.6477	0.0910	0.0910
5	0.1755	0.1725	0.4293	0.4059	0.5814	0.5447	0.0782	0.0799
6	0.1845	0.1827	**0.5565**	0.5465	0.4795	0.4790	0.0137	-0.0109
7	0.1440	0.1205	0.4485	0.4270	0.4332	0.4231	0.0137	-0.0109
8	0.1445	0.1370	0.4874	0.4197	0.4301	0.4205	0.0034	-0.0179
9	0.2325	0.1706	0.4881	0.3999	0.4053	0.4001	0.0151	-0.0374
10	0.1667	0.1660	0.3973	0.3899	0.4045	0.4006	0.0428	0.0346
11	0.1937	0.1496	0.3662	0.3293	0.3805	0.3674	0.0868	-0.0748
12	0.1412	0.1334	0.3636	0.3329	0.3639	0.3621	0.0900	0.0600

In this section, the clusters = 2, 3,....., and 12 depicts the number of clusters equal to 2, 3,....., and 12, respectively. We evaluate both algorithms on the two measures mentioned above. For each SNP dataset, we run the SRSIO-FCM [241] algorithm ten times by varying the number of cluster centers from 4 to 13. This leads us to have BSs as input to SIFCC and SFCC. We concatenate the BSs to have the Y matrix, which is used as input for the SIFCC/SFCC algorithm. The number of subsets into which Y is divided is 3. The SIFCC partitions the data into three subsets in comparison

with the SFCC (100% data) that performs the clustering on whole data. A detailed discussion of the SNP datasets is present in the subsequent section.

In Table 5.3, we have reported the results on the SoySNP50K Wm82.a1, SNP-seek rice, MAGIC-rice, and 248Entries rice datasets in terms of SI. This index demonstrates the quality of clustering, a negative value indicates poor clustering, and a positive value indicates good clustering quality. Subsequently, a superior clustering would bring about higher SI. The table shows that the value of SI for SFCC and SIFCC are very close for the SoySNP50K Wm82.a1 dataset. Besides this, SIFCC attains a higher value of SI for cluster2. Observing the values of SI for the SNP-seek rice dataset, SIFCC has obtained a remarkable value than SFCC. Furthermore, the SIFCC achieved the highest value of SI for cluster6. Observing the values of SI for the MAGIC-rice dataset, SIFCC has obtained a higher value than SFCC. Additionally, the SIFCC achieved the highest value of SI for cluster4. Moreover, comparing the values of SI for the 248Entries rice dataset, we observed that SIFCC has attained positive SI for each cluster, whereas SFCC obtained negative SI values for most of the clusters as shown in the table. The experimental analysis shows that SIFCC performs good clustering quality than SFCC for the 248Entries rice dataset. While speaking about the difference in values of SI among various clusters, SI for SFCC (100% data) is comparatively smaller than the SI values achieved with SIFCC.

In Table 5.4, we have reported the results on the SoySNP50K Wm82.a1, SNP-seek rice, MAGIC-rice, and 248Entries rice datasets in terms of DBI. The DBI is not limited inside a given range, and thus the lower DBI indicates good clustering quality. The table shows that the value of DBI for SFCC is higher than SIFCC for the SoySNP50K Wm82.a1 dataset. Besides this, SIFCC attains a lower value of DBI for cluster9. Observing the values of DBI for the SNP-seek rice dataset, SIFCC has obtained a remarkable value than SFCC. Furthermore, the SIFCC achieved a lower value of DBI for cluster11. Observing the values of DBI for the MAGIC-rice dataset, SIFCC has obtained a lower value than SFCC. Additionally, the SIFCC achieved a lower value of DBI for cluster4. Moreover, comparing the values of DBI for the 248Entries rice dataset, we observed that SIFCC has attained lower DBI values than

Table 5.4: Values of the DBI in the range of cluster = 2....12 for all the four SNP datasets using the SIFCC and SFCC algorithm (Entries in boldface indicate the optimal values for respective indices)

Cluster	**Datasets**							
	SoySNP50k Wm82.a1		SNP-seek rice		MAGIC-rice		248Entries rice	
	SIFCC	**SFCC**	**SIFCC**	**SFCC**	**SIFCC**	**SFCC**	**SIFCC**	**SFCC**
2	3.1509	3.1382	0.7799	0.7799	1.9884	1.9884	**1.0876**	**1.0876**
3	1.8902	1.8787	0.7393	0.7393	1.7028	1.8698	1.1904	1.1960
4	1.5664	2.1276	2.4978	2.5817	**0.4321**	0.4336	1.3015	1.3015
5	7.4855	11.7560	0.9276	0.9975	0.6585	0.7285	1.3665	1.4064
6	2.0356	2.8473	0.6844	0.6944	0.8601	0.8602	1.4498	3.2431
7	1.5318	5.2681	0.7226	0.7355	0.8639	0.8663	1.5255	2.9410
8	5.6977	6.3448	1.6441	1.9869	1.3664	1.3726	1.9973	5.5571
9	**1.3152**	1.5507	1.7945	1.9901	1.5923	1.5988	1.7689	2.2174
10	2.5039	3.5427	1.1522	1.9901	2.1753	2.9134	2.6644	2.7330
11	2.4282	2.6192	**1.1222**	1.9586	2.0708	2.5117	1.7728	2.4235
12	2.5039	2.8085	2.0500	2.8695	1.9660	1.9821	1.6618	1.9300

SFCC from cluster3 onward. Besides this, the value of DBI for SFCC and SIFCC is almost similar for cluster2 and the lowest one in comparison with other clusters. While speaking about the difference in values of DBI among various clusters, DBI for SFCC (100% data) is comparatively higher than the DBI values achieved with SIFCC. Thus, comparing both the measures, we conclude that SIFCC performs better than the SFCC.

5.3 Summary

In this chapter, a scalable preprocessing approach is proposed for the huge SNP data which is obtained from sequenced plant genomes. The proposed SNP preprocessing method provides 12-dimensional numeric feature vectors from huge SNP sequences using the Apache Spark cluster. The preprocessed SNP sequences are further used as an input to the earlier proposed scalable fuzzy clustering algorithms (presented in Chapters 3 and 4) for clustering massive SNP sequences. First, we directed the exact evaluation of the proposed KSRSIO-FCM on the different SNP datasets. This exhibited potential advantages for utilizing our methodology for clustering of SNP sequences and compared it with other scalable fuzzy clustering approaches. Additionally, the KSRSIO-FCM performed clustering of preprocessed SNP sequences in lesser time than the KSLFCM algorithm. Second, we used both the proposed approaches, i.e, SIFCC and SFCC to efficiently cluster SNP sequences at high speed and high accuracy. Then, the accurate assessments of SIFCC on the various SNP datasets are conducted, which demonstrated potential benefits for using our approach for clustering of SNP sequences. The significant characteristic of the proposed SNP preprocessing approach is that it takes raw SNP sequences as input and distributes the dataset into various slave nodes and produces a 12-dimensional numerical feature vector as output.

At this point in this thesis, we have developed preprocessing approach for huge SNP data. The next chapter is focused on preprocessing methods for massive protein sequences using Apache Spark cluster.

Chapter 6

Design of a Novel Scalable Feature Extraction Algorithms for Huge Protein Sequences with Application of Scalable Fuzzy Clustering Algorithm

In this chapter, we have proposed scalable feature extraction algorithms for preprocessing of massive protein sequences using Apache Spark cluster. The protein datasets are obtained from SoyBase database[1]. The proposed two scalable preprocessing methods for massive protein sequences are 60-dimensional Scalable Protein Feature (60d-SPF) and 6-dimensional Scalable Co-occurrence-based Probability-Specific Feature (6d-SCPSF) extraction approach. The high dimensionality of protein data creates several crucial problems for researchers during the implementation of machine learning algorithms. A good input representation (extraction of features) is necessary for the proper clustering of protein sequences. The feature extraction techniques [186, 189] have been introduced in the past, but, none of them is scalable. There is a need for a scalable method that can select statistically significant features from a huge protein sequence. Neha et al. [186] developed an approach to extract a six-dimensional numerical feature vector from a protein sequence. Gupta et al. [189] developed an alignment-free method to find the similarity among protein sequences via the general

[1]https://soybase.org/

form of pseudo amino acid composition [187], which is a sixty-dimensional numerical feature vector of the protein sequence. Despite the wide popularity and the existence of many feature extraction approaches, there is a still need to develop a highly accurate and efficient feature extraction approach for the interpretation of a huge volume of variable length protein sequences.

This section describes the scalable feature extraction methods for massive protein sequences implemented on Apache Spark cluster. To propose a scalable protein preprocessing algorithm, we followed the *PseAAC* approach discussed in [189] named 60d-SPF and applied this approach to huge protein data to extract 60-dimensional numeric feature vectors. Additionally, we have proposed 6d-SCPSF by making the CPSF approach [186] scalable. To make preprocessing algorithm for protein sequence a scalable algorithm, we executed it on Apache Spark cluster. The results obtained from the proposed scalable feature extraction techniques are used as input to the well-known SRSIO-FCM [52] for clustering of protein sequences. A detailed description of the SRSIO-FCM algorithm is discussed in Section 2.3.2. The proposed approaches are discussed in the subsequent section.

6.1 60-dimensional Scalable Protein Feature Extraction (60d-SPF) Approach

The proposed scalable 60d-SPF a protein preprocessing approach is being implemented using Apache Spark framework to represent all the protein sequences in terms of 60-dimensional numeric feature vectors. ***Algorithm 6.1*** summarizes the steps of 60d-SPF using the Apache Spark framework. The input given to the 60d-SPF approach is a raw protein dataset containing 20 amino acids $\sum$={A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}. The output generated by the 60d-SPF approach is a feature vector of the given input dataset, which is a file containing 60-dimensional numeric feature vectors. In Line 1 of ***Algorithm 6.1***, the data is read into a RDD from the Hadoop using the *pyspark* class, i.e., *SparkContext.textFile*.

Then a map function is used to accomplish the preprocessing task. In Line 2 of ***Algorithm 6.1***, the map function allows calling the *Protein_Preprocess* function to each RDD row. Finally, the obtained RDD after the map function is saved using Line 3 of ***Algorithm 6.1***.

Algorithm 6.1 60d-SPF

Input: raw protein data : *raw_Protein.txt*

Output: processed protein data : *proc_Protein.txt*

1: $y_1 = SparkContext.textFile(raw\_Protein.txt).map(lambda\ z : numpy.array(z))$

2: $y_1 = y_1.map(lambda\ z : Protein\_Preprocess(z))$

3: $y_1.saveAsTextFile(proc\_Protein.txt)$

Algorithm 6.2 *Protein_Preprocess*

Input: z; z is a numpy array

Output: *numpy.array*[A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y].

1: Let i denote amino $\sum$={A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}.

2: **for** x **in** z **do**

3: **if** x is i **then**

4: Increase the count of amino i i.e., ℓ_i++.

5: **end if**

6: **end for**

7: **Calculate** the total distance $\mathcal{T}_i$ using Eq. (2.34).

8: **Calculate** the variance of distance D_i using Eq. (2.35).

The proposed scalable protein preprocessing approach extracts features of protein sequences in three sets of the numerical parameter. The first parameter is a calculation of the length of sequences. The second parameter is the total distances of each amino base to the first amino acid. The third parameter is the variance of distance for each amino acid [189]. Each set of a numerical parameter is not sufficient to denote a specific protein sequence. Thus, the combination of all of the three sets of the numerical parameter, which contains 60-dimensional numeric feature vectors is

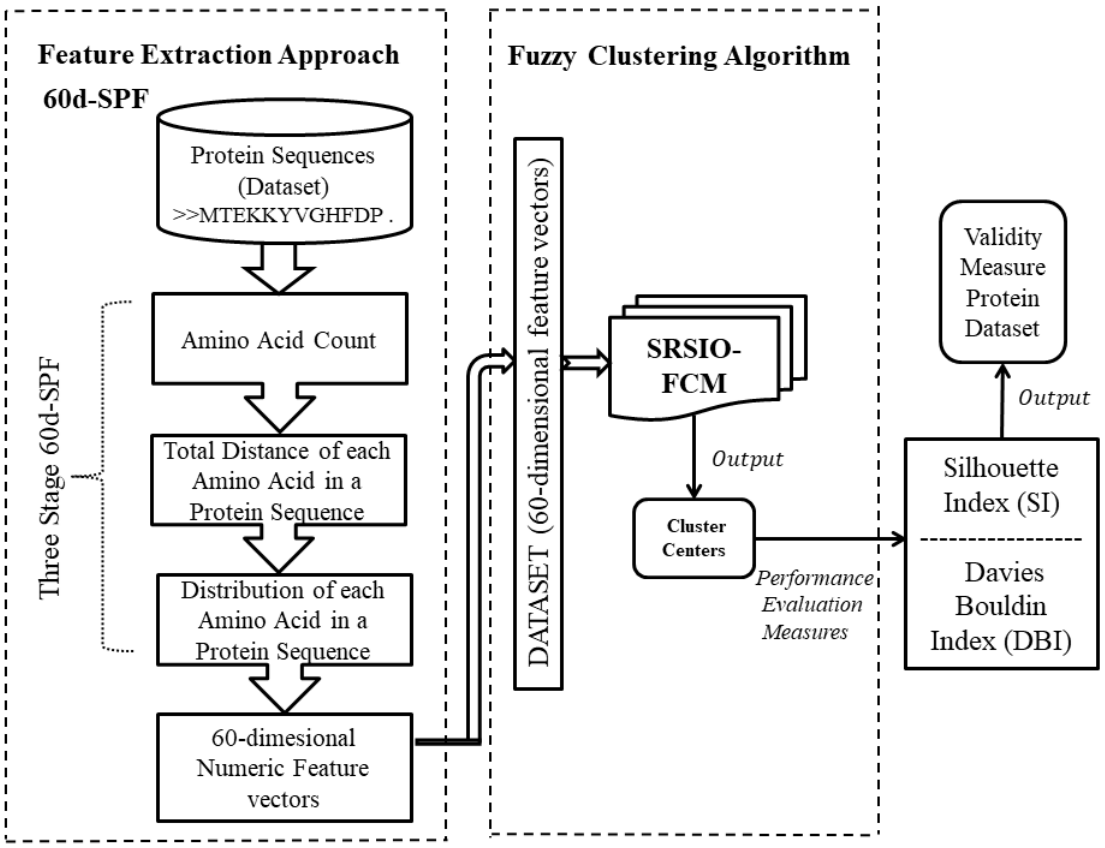


Figure 6.1: Workflow of 60d-SPF Architecture.

used to characterize similarities between protein sequences. The details of the protein feature extraction approach are given in ***Algorithm 6.2***. In this algorithm, we discuss the *Protein_Preprocess* algorithm, which is called by 60d-SPF given in ***Algorithm 6.1***. The working of the *Protein_Preprocess* algorithm is shown in terms of various steps by using an example given in the subsequent subsection. The proposed scalable protein preprocessing algorithm takes raw protein sequences as input and produces 60-dimensional numeric feature vectors as an output using Apache Spark framework. The architecture of the proposed 60d-SPF extraction approach is shown in Figure 6.1. The SRSIO-FCM algorithm takes the preprocessed 60-dimensional numeric feature vectors of huge protein sequences as input and produces output in terms of clusters. The working of the *Protein_Preprocess* algorithm is shown in terms of various steps by using an example given in the subsequent subsection.

6.1.1 Stage I: Calculation of length of sequence

The *Protein_Preprocess* algorithm (given in ***Algorithm 6.2***) working is being described using a first step in which the length of each amino acid sequence is calculated. Here an illustration is presented by considering an example of five sequences as shown in Figure 6.2. The output obtained after preprocessing of these five sequences are shown in Figure 6.3, where the first column represents the sequence number and the other columns present the 60-dimensional numeric feature vector. $\langle \ell_A, \ell_C, \ell_D, \ell_E, \ell_F, \ell_G, \ell_H, \ell_I, \ell_K, \ell_L, \ell_M, \ell_N, \ell_P, \ell_Q, \ell_R, \ell_S, \ell_T, \ell_V, \ell_W, \ell_Y, \mathcal{T}_A, \mathcal{T}_C, \mathcal{T}_D, \mathcal{T}_E, \mathcal{T}_F, \mathcal{T}_G, \mathcal{T}_H, \mathcal{T}_I, \mathcal{T}_K, \mathcal{T}_L, \mathcal{T}_M, \mathcal{T}_N, \mathcal{T}_P, \mathcal{T}_Q, \mathcal{T}_R, \mathcal{T}_S, \mathcal{T}_T, \mathcal{T}_V, \mathcal{T}_W, \mathcal{T}_Y, D_A, D_C, D_D, D_E, D_F, D_G, D_H, D_I, D_K, D_L, D_M, D_N, D_P, D_Q, D_R, D_S, D_T, D_V, D_W, D_Y \rangle$.

The total number of a particular amino acid $\sum = \{\ell_A, \ell_C, \ell_D, \ell_E, \ell_F, \ell_G, \ell_H, \ell_I, \ell_K, \ell_L, \ell_M, \ell_N, \ell_P, \ell_Q, \ell_R, \ell_S, \ell_T, \ell_V, \ell_W, \ell_Y\}$ is present in sequence1: $\langle TAKHTGPGKVIVNTTHGPIDVELWPKEAPKSVRNFVQCLCL CL \rangle$ are $[2, 2, 1, 2, 1, 3, 2, 2, 2, 4, 3, 0, 2, 4, 1, 1, 1, 4, 4, 4, 1, 0]$, respectively. Likewise, we calculate the value of the other four sequences as shown in Figure 6.3. After calculating the total number of amino acids in each sequence, the total distances of each amino acid to the first amino acid is calculated, which is presented next.

```
1 TAKHTGPGKVIVNTTHGPIDVELWPKEAPKSVRNFVQCLCL
2 KANPKVFFYLAVDGRLAGRIVIELFADTTPRTAENFRAALC
3 MSVLIVTSLGDIVIDLYPNKCPLTCKNFLGQRLGRIVIGLY
4 EITNRVFLDVDIDGQRLGRIVIGLYGTVVPKTVENFRAELC
5 MMVVDKFEDDNFTAKHTGPGLLSTGVKMRIVARKEKHLHIA
```

Figure 6.2: Example of protein sequences.

```
1 [2, 28, 169.0, 2, 76, 1.0, 1, 19, 0.0, 2, 47, 6.25, 1, 34, 0.0, 3, 28, 22.8, 2, 18, 36.0, 2, 28, 16.0, 4, 64, 127.5, 3, 100, 64.8, 0, 0, 0, 2, 45, 110.25, 4, 75, 69.6, 1, 36, 0.0, 1, 32, 0.0, 1, 30, 0.0, 4, 31, 35.1, 5, 106, 108.2, 1, 23, 0.0, 0, 0, 0]
2 [7, 159, 172.4, 1, 40, 0.0, 2, 38, 49.0, 2, 55, 30.25, 4, 72, 147.5, 2, 30, 4.0, 0, 0, 0, 2, 40, 1.0, 2, 4, 4.0, 4, 86, 126.7, 0, 0, 0, 2, 36, 256.0, 2, 32, 169.0, 0, 0, 0, 4, 98, 78.7, 0, 0, 0, 3, 86, 2.8, 3, 36, 38.0, 0, 0, 0, 1, 8, 0.0]
3 [0, 0, 0, 2, 44, 4.0, 2, 24, 4.0, 0, 0, 0, 1, 27, 0.0, 4, 109, 121.2, 0, 0, 0, 5, 100, 180.0, 2, 44, 9.0, 7, 147, 146.2, 1, 0, 0.0, 2, 44, 16.0, 2, 38, 4.0, 1, 30, 0.0, 2, 65, 2.25, 2, 8, 9.0, 2, 29, 72.3, 4, 55, 178.2, 0, 0, 0, 2, 56, 144.0]
4 [1, 37, 0.0, 1, 40, 0.0, 3, 30, 2.6, 3, 71, 284.2, 2, 41, 210.25, 4, 77, 21.2, 0, 0, 0, 4, 52, 62.0, 1, 30, 0.0, 4, 85, 137.2, 0, 0, 0, 2, 37, 240.3, 1, 29, 0.0, 1, 14, 0.0, 4, 73, 132.2, 0, 0, 0, 3, 59, 160.2, 6, 121, 100.5, 0, 0, 0, 1, 24, 0.0]
5 [3, 84, 126.0, 0, 0, 0, 3, 21, 4.6, 2, 41, 182.25, 2, 17, 6.25, 3, 60, 8.6, 3, 89, 108.2, 2, 68, 25.0, 5, 113, 131.44, 3, 78, 60.6, 3, 28, 156.2, 1, 10, 0.0, 1, 18, 0.0, 0, 0, 0, 2, 60, 4.0, 1, 22, 0.0, 3, 51, 20.6, 4, 60, 159.5, 0, 0, 0, 0, 0, 0]
```

Figure 6.3: Preprocessed result using proposed 60d-SPF extraction method for the protein sequences given in Figure 6.2.

6.1.2 Stage II: Total distances of each amino acid to the first amino acid

The second numerical parameter, i.e., the total distances of each amino acid-base to the first amino acid $\mathcal{T}_i$, is calculated using Eq. (2.34). Then the position values corresponding to each amino acid are added as shown in Figure 6.2. Like, amino acid G appears at 5, 7, and 16 (index starts from 0). So, the value of $\mathcal{T}_G$ is calculated as follows:

$$\mathcal{T}_G = 5 + 7 + 16 = 28$$

Likewise, we calculate the value of $\mathcal{T}_A, \mathcal{T}_C, \mathcal{T}_D, \mathcal{T}_E, \mathcal{T}_F, \mathcal{T}_G, \mathcal{T}_H, \mathcal{T}_I, \mathcal{T}_K, \mathcal{T}_L, \mathcal{T}_M, \mathcal{T}_N, \mathcal{T}_P, \mathcal{T}_Q, \mathcal{T}_R, \mathcal{T}_S, \mathcal{T}_T, \mathcal{T}_V, \mathcal{T}_W, \mathcal{T}_Y$ for sequence1 and thus the total distances of each amino acid to the first amino acid for all other four sequences is shown in Figure 6.3. After calculating the total distances of each amino acid to the first amino acid, the variance of distance for each amino acid is calculated, which is presented next.

6.1.3 Stage III: Variance of distance for each amino acid

The third numerical parameter D_i represents the variance of distance for each amino acid, which is calculated using Eq. (2.35). The first step is to compute d_i using $d_i = \frac{\mathcal{T}_i}{\ell_i}$. The value of d_G in sequence1 is calculated as follows:

$$d_G = \frac{\mathcal{T}_G}{\ell_G} = \frac{31}{3} = 10.33$$

The second step computes the variance of distance for each amino acid-base, i.e., D_i, for example, the value of D_G in sequence1 is 22.88, which is calculated as follows:

$$D_G = \frac{[(6-10.33)^2+(8-10.33)^2+(17-10.33)^2]}{3} = 22.88$$

Hence, the result obtained from sequence1, which contains 60-dimensional numeric feature vector as follows: $[2, 28, 169.0, 2, 76, 1.0, 1, 19, 0.0, 2, 47, 6.25, 1, 34, 0.0, 3, 28, 22.8, 2, 18, 36.0, 2, 28, 16.0, 4, 64, 127.5, 3, 100, 64.8, 0, 0, 0, 2, 45, 110.25, 4, 75, 69.6, 1, 36, 0.0, 1, 32, 0.0, 1, 30, 0.0, 4, 31, 35.1, 5, 106, 108.2, 1, 23, 0.0, 0, 0, 0]$

The results of all the other four sequences are shown in Figure 6.3. The proposed scalable protein preprocessing algorithm (60d-SPF) has the significant characteristic that it takes raw protein sequences as input and produces 60-dimensional numeric feature vectors as output. The SRSIO-FCM algorithm takes the preprocessed 60-dimensional numeric feature vectors of massive protein sequences as input and produces output in the form of a cluster belonging to each data sample. Additionally, we have proposed scalable 6-dimensional feature vectors (6d-SCPSF) for preprocessing of massive protein sequences as presented in the subsequent section.

6.2 6-dimensional Scalable Co-occurrence-based Probability-Specific Feature (6d-SCPSF) Extraction Approach

The proposed scalable 6d-SCPSF a protein preprocessing approach is being implemented using Apache Spark framework to represent all the protein sequences in 6-dimensional numeric feature vectors. The architecture of the proposed 6d-SCPSF extraction approach is shown in Figure 6.4. ***Algorithm 6.3*** summarizes steps of scalable 6d-SCPSF protein preprocessing technique using the Apache Spark framework. The proposed 6d-SCPSF approach takes input a raw protein dataset containing 20 amino acids $\sum$={A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}. Then in output, it generates a feature vector of the given input dataset, a file containing 6-dimensional numeric feature vector. Line 1 of ***Algorithm 6.3***, calls the Scalable Protein Sequence Encoding (SPSE) algorithm (***Algorithm 6.4***), which distributes *raw_Protein.txt* protein dataset on Apache Spark cluster. The encoding of protein amino acid is performed using stage one technique, i.e., PSE, as explained in Section 2.4.2. In Line 2, the Global Similarity Measures (GSM) algorithm (***Algorithm 6.5***) is called, which calculates the probability matrix of the sequences on a master machine without distributing the dataset in spark clusters. The GSM approach is explained in Section 2.4.2. The output obtained from the SPSE algorithm is used as an input to

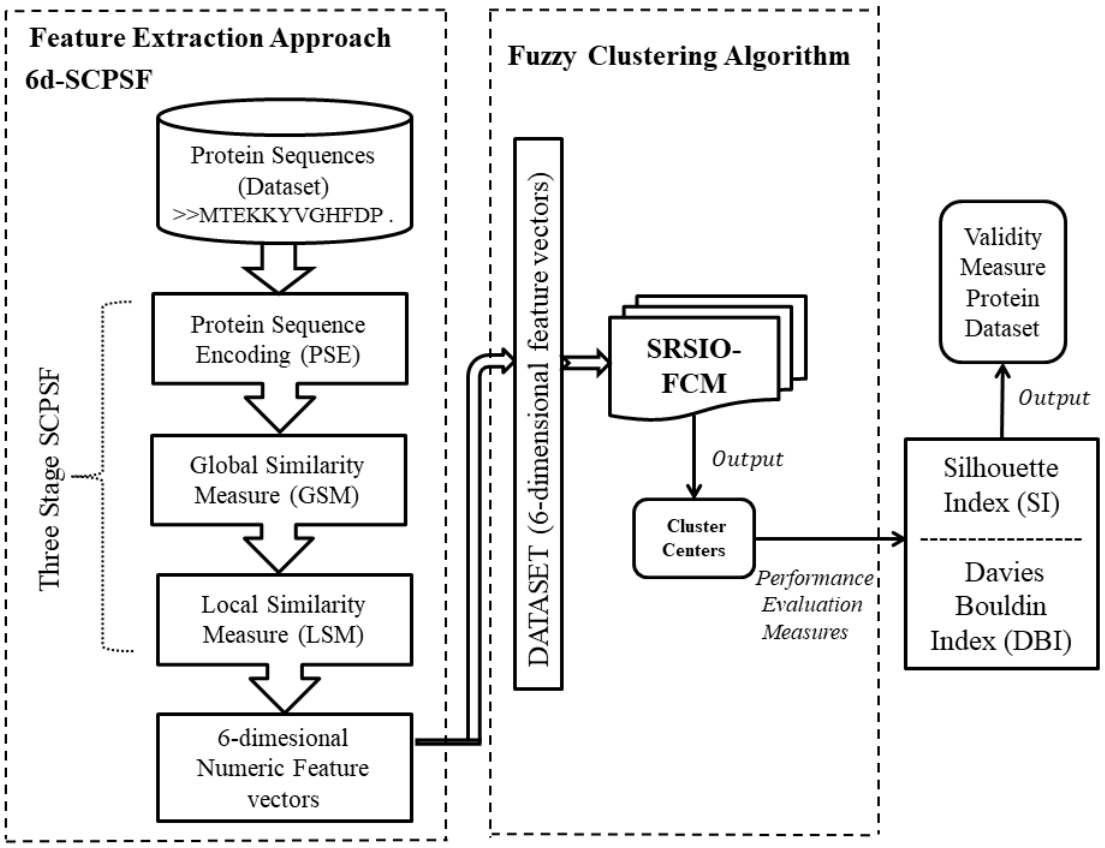


Figure 6.4: 6d-SCPSF Architecture embedded using SRSIO-FCM with Performance Measure Evaluation.

the GSM algorithm. In Line 3, the Scalable Local Similarity Measures (SLSM) algorithm (***Algorithm 6.6***) is called, which again distributes the output obtained from the GSM algorithm using Apache Spark cluster to find LSM as explained in 2.4.2. Finally, the 6-dimensional numerical feature vectors are saved in a file using Line 4. The subsequent section explains the SPSE, GSM, and SLSM algorithms.

6.2.1 SPSE Algorithm

The PSE algorithm encodes each amino acid into particular encoding groups, as discussed in Section 2.4.2. This section presents the scalable version of PSE (SPSE). ***Algorithm 6.4*** shows the steps of the SPSE approach. The data is read into a resilient distributed dataset (RDD) from the Hadoop in Line 1. In Line 2, the file

Algorithm 6.3 6d-SCPSF algorithm

Input: raw protein data : $raw\_Protein.txt$

Output: preprocessed protein data : $pre\_Protein.txt$

1: **Call** Scalable Protein Sequence Encoding (SPSE) algorithm.

2: **Call** Global Similarity Matrix (GSM) algorithm.

3: **Call** Scalable Local Similarity Measures (SLSM) algorithm.

4: $saveAsTextFile(Feature\_Vectors.txt)$

($raw\_Protein.txt$) is pass to the Encode function and store as the return values in other Spark RDD. Then the map function distributes Encode() method to every worker node for parallel execution of the task in Line 3. Save the data of encoded RDD to the text file in Line 4. From Line 5-12, the working of Encode() function is given. The encoding of sequences is exchanged with the values from Line 7-12. The DataFrame is used to store the computation of each step. In Spark, a DataFrame is a distributed collection of data organized into named columns. Section 2.4.2 discusses the steps used to perform protein sequence encoding. In Line 13, the output of the SPSE algorithm is saved in the $enc\_Protein.txt$ file. After that, the $enc\_Protein.txt$ file is taken as input to the GSM algorithm, which is presented next.

6.2.2 GSM Algorithm

The GSM is used to calculate the instance probability of all exchange groups at each position relative to the total number of protein sequences. ***Algorithm 6.5*** discusses the steps of the GSM approach. The input of this algorithm is $enc\_Protein.txt$, which is obtained from ***Algorithm 6.4***. Line 1 reads the data from a file and saves it to a DataFrame. Line 2 finds the number of columns of DataFrame, and Line 3 locates the number of rows of DataFrame. An empty DataFrame is created for the estimation of the probability DataFrame of sequences in Line 4. Then, calculate the occurrence of the exchange group for each column and add this to the DataFrame in Line 8. In Line 9, the value of the probability of exchange group is computed using Eq. (2.36). Line 10, add the probability to the list. In Line 12, a list is added as a new

Algorithm 6.4 Scalable Protein Sequence Encoding (SPSE)

Input: raw protein sequence: *raw_Protein.txt*

Output: encoded protein sequence: *enc_Protein.txt*

1: **Read** the protein sequences from the file and parallelize them with the help of Spark RDD.
2: **Pass** the file (*raw_Protein.txt*) to **Encode** function and store the return values in other Spark RDD.
3: $y_1 = y_1 : map(lambda\ z : Encode(z))$
4: **Save** data of encoded RDD to textFile.
5: *Function* Encode(z){
6: **Store** the data in the DataFrame and split each letter in the sequence to different columns.
7: **Replace** the $'H', 'R', 'K'$ with e_1.
8: **Replace** the $'D', 'E', 'N', 'Q'$ with e_2.
9: **Replace** the $'C'$ with e_3.
10: **Replace** the $'S', 'T', 'P', 'A', 'G'$ with e_4.
11: **Replace** the $'M', 'I', 'L', 'V'$ with e_5.
12: **Replace** the $'F', 'Y', 'W'$ with e_6.
13: **Returns** the encoded protein sequence: *enc_Protein.txt* }

column in the probability DataFrame. Line 14 saves the result to *prob_Protein.txt* file that is used as input to the SLSM algorithm. Section 2.4.2 discusses the steps used to find global similarity measures of the protein-encoding sequence. After that, *enc_Protein.txt* and *prob_Protein.txt* files are taken as input to the next stage in the SLSM algorithm, which is presented next.

6.2.3 SLSM Algorithm

This section presents the scalable version of LSM (SLSM). The SLSM determines the location-specific weight of each exchange group within the sequence and produces the weight factors. These weight factors ultimately represent the numeric feature vectors for each protein sequence. ***Algorithm 6.6*** discusses the steps of

Algorithm 6.5 Global Similarity Measures (GSM)

Input: encoded protein sequences: *enc_Protein.txt*

Output: probability DataFrame of sequences: *prob_Protein.txt*

1: **Read** the encoded sequences data from the file and store it in the DataFrame.
2: **Find** the number of columns of the DataFrame(col).
3: **Find** the number of rows of the DataFrame(row).
4: **Create** an empty probability DataFrame with index names $e_1, e_2, e_3, e_4, e_5, e_6$.
5: **for** each column in range 0 to col **do**
6: **Create** an empty list.
7: **for** each exchange group **do**
8: **Get** the occurrence of exchange group in that column.
9: **Get** the probability of the exchange group using Eq. (2.36).
10: **Add** probability to the list.
11: **end for**
12: **Add** list as a new column in the probability DataFrame.
13: **end for**
14: **Save** probability DataFrame as textFile(*prob_Protein.txt*).

the SLSM approach. This algorithm takes input two files, i.e., *enc_Protein.txt* and *prob_Protein.txt*, which are obtained from ***Algorithm 6.4*** and ***Algorithm 6.5***, respectively. The data (*enc_Protein.txt*) is read into an RDD from the Hadoop in Line 1. Then, the data from the file (*prob_Protein.txt*) is accessed and stored in DataFrame (probability) in Line 2. The map function distributes FeatureVector() method to every worker node for parallel execution of the task in Line 3. Line 4 saves the data obtained from featureVector RDD to textFile. From Line 5-15, the working of FeatureVector() function is given, where the DataFrame is used to store the computation of each step. An empty DataFrame is created with given column names in Line 6. Then, a DataFrame is created using the data in Line 7. Line 8 finds the number of cols, and Line 9 locates the number of rows of the DataFrame. The exchange group of rows and columns from DataFrame is acquired in line 13. In Line 14, modify the key value of the exchange group from featureVector by adding previous value with the

Algorithm 6.6 Scalable Local Similarity Measures (SLSM)

Input: Probability matrix of sequences: *prob_Protein.txt*, encoded protein sequences: *enc_Protein.txt*

Output: Feature Vectors: $Feature\_Vectors.txt$

1: **Read** the sequences from the file (*enc_Protein.txt*) and parallelize using Spark RDD.
2: **Read** the data from the *prob_Protein.txt* and store it in DataFrame (probability).
3: $y_2 = y_2 : map(lambda\ y_3 : FeatureVector(y_3 : probProtein.txt))$
4: **Save** data of FeatureVector RDD to textFile.
5: *Function* **FeatureVector**(probability, data)
6: **Create** an empty featureVector DataFrame with column names as $e_1, e_2, e_3, e_4, e_5, e_6$.
7: **Create** DataFrame using data.
8: **Find** the number of columns of the DataFrame(cols).
9: **Find** the number of rows of the DataFrame(rows).
10: **for** row in range of 0 to rows **do**
11: **Create** a dictionary in as featureVector=$'e_1' : 0,' e_2' : 0,' e_3' : 0,' e_4' : 0,' e_5' : 0,' e_6' : 0.$
12: **for** col in range of 0 to cols **do**
13: **Get** the exchange group from the position of rows and cols in DataFrame.
14: **Modify** the key value of the exchange group from featureVector by adding previous value with the value of probability at the position of exchange group and cols from probability DataFrame.
15: **end for**
16: **Add** the featureVector to the featureVector DataFrame.
17: **end for**
18: **Save** the featureVector DataFrame to textFile ($Feature\_Vectors.txt$).

value of probability at the position of the exchange group and columns from probability DataFrame. Line 10-17 is computed using Eq. (2.37). Section 2.4.2 discusses the steps used to find local similarity measures of the protein-encoding sequence. Finally, the results are saved in the $Feature\_Vectors.txt$ file in Line 18.

The proposed scalable protein preprocessing algorithm (6d-SCPSF) has the sig-

nificant characteristics that it takes raw protein sequences as input and produces 6-dimensional numeric feature as an output. In Section 6.3, we present the experimental results applied to various protein datasets. The SRSIO-FCM algorithm takes the preprocessed 6-dimensional numeric feature vectors of huge protein sequences as input and produces output in terms of clusters.

6.3 Experimental Evaluation on Protein Datasets

In this section, we present the experimental results on various protein datasets applied to scalable fuzzy clustering algorithm. We have analyzed the SRSIO-FCM algorithms [52] exhibition, where the input data is represented as the numerical feature vectors which is preprocessed using proposed 60d-SPF and 6d-SCPSF extraction methods on the Lee, Williams82, PI483463, and W05 protein datasets[2]. A detailed description of all these datasets is presented in Section 2.6.2. The experimental evaluation is performed on Apache Spark cluster. The detailed description of Apache Spark cluster setup is presented in Section 3.4.2. In the experiments, we have analyzed the performance of the proposed 60d-SPF and 6d-SCPSF extraction method applied to the SRSIO-FCM algorithm using the Silhouette Index (SI) and Davies Bouldin Index (DBI) on Apache Spark cluster. The experimental results and discussion is presented next.

Comparative Analysis of 60d-SPF and 6d-SCPSF applied to SRSIO-FCM Algorithm in terms of SI and DBI measures

We have done the comparative analysis of proposed 60d-SPF and 6d-SCPSF extraction methods applied to the SRSIO-FCM algorithm. The input data fed to the SRSIO-FCM approach is the output obtained from proposed 60d-SPF and 6d-SCPSF extraction methods. We have performed clustering with the number of subsets 3, where the subsets represent the chunks of the entire data. In this section, subset3 depicts the number of subsets equal to 3. The SRSIO-FCM algorithm partitions the dataset into three subsets. The clustering is performed on the number of clusters 5, 10,

[2]https://soybase.org/

Table 6.1: Values of the SI for 6d-SCPSF and 60d-SPF on all the four protein datasets using the SRSIO-FCM algorithm.

	Datasets							
Cluster	**Lee**		**Williams82**		**PI48346**		**W05**	
	6d-SCPSF	60d-SPF	6d-SCPSF	60d-SPF	6d-SCPSF	60d-SPF	6d-SCPSF	60d-SPF
5	0.4501	0.6678	0.4607	0.6672	0.4492	0.6329	0.4526	0.6505
10	0.3361	0.4826	0.3391	0.5055	0.3358	0.478	0.3341	0.4895
15	0.2721	0.3922	0.2706	0.4174	0.2533	0.3894	0.2732	0.4063
20	0.2511	0.3522	0.2538	0.3918	0.24	0.3471	0.2494	0.3636
25	0.2303	0.3074	0.2252	0.331	0.2244	0.3063	0.2358	0.3087
30	0.199	0.2131	0.2201	0.2868	0.2013	0.2699	0.217	0.2719

15, 20, 25, and 30, respectively. Likewise, the cluster5, cluster10, cluster15, cluster20, cluster25, and cluster30 depicts the number of clusters equal to 5, 10, 15, 20, 25, and 30, respectively.

(i) **SI values for all four protein datasets applied on SRSIO-FCM using 60d-SPF and 6d-SCPSF extraction method:**

Table 6.1 highlights SI values for the Lee, Williams82, PI483463, and W05 Soybean protein datasets. SI demonstrates the quality of clustering. Subsequently, a superior clustering would bring about higher SI. Observing the SI values for the Lee dataset, the SRSIO-FCM algorithm has obtained a lower value for the 6d-SCPSF extraction method than the 60d-SPF extraction method. On the other hand, the SRSIO-FCM achieved a higher value on cluster5 for the 60d-SPF approach. Along these lines, we can conclude that 60d-SPF performs better than 6d-SCPSF when applied on the SRSIO-FCM algorithm in terms of SI values for the Lee protein dataset.

Observing the SI values for Williams82, the SRSIO-FCM algorithm has obtained a lower value for the 6d-SCPSF extraction method than for the 60d-SPF extrac-

tion method. On the other hand, the SRSIO-FCM achieved a higher value on cluster5 for 60d-SPF. Along these lines, we can conclude that 60d-SPF performs better than 6d-SCPSF when applied on the SRSIO-FCM algorithm in terms of SI values for the Williams82 protein dataset.

Observing the SI values for the PI483463 dataset, the SRSIO-FCM algorithm has obtained a higher value for the 60d-SPF extraction method than the 6d-SCPSF extraction method. On the other hand, the SRSIO-FCM achieved a higher value for cluster5 for 60d-SPF. Along these lines, we can conclude that 60d-SPF performs better than 6d-SCPSF when applied on the SRSIO-FCM algorithm in terms of SI values for the PI483463 protein dataset.

Observing the SI values for the W05 dataset, the SRSIO-FCM algorithm has obtained a higher value for the 60d-SPF extraction method than the 6d-SCPSF extraction method. On the other hand, the SRSIO-FCM achieved a higher value for cluster5 for 60d-SPF. Along these lines, we can conclude that 60d-SPF performs better than 6d-SCPSF when applied on the SRSIO-FCM algorithm in terms of SI values for the W05 protein dataset.

(ii) **DBI values for all four protein datasets applied on SRSIO-FCM using 60d-SPF and 6d-SCPSF extraction method:**

Table 6.2 highlights DBI values for Lee, Williams82, PI483463, and W05 Soybean protein datasets. Conversely to the SI, the DBI is not bounded within a given range. As a general rule, the lower the DBI value better the clustering result. Observing the DBI values for the Lee dataset, the value achieved by SRSIO-FCM is much better for the 60d-SPF than 6d-SCPSF. As we can see, for 60d-SPF, the DBI values are lower than 6d-SCPSF when clustered using SRSIO-FCM on almost all the clusters. Moreover, SRSIO-FCM attained a very low value on cluster5. In this way, we can conclude that 60d-SPF performs better than 6d-SCPSF when applied on the SRSIO-FCM algorithm in terms of DBI values for the Lee protein dataset.

Observing the DBI values for the Williams82 dataset, the value achieved by

Table 6.2: Values of the DBI for 6d-SCPSF and 60d-SPF on all the four protein datasets using the SRSIO-FCM algorithm.

	Datasets							
Cluster	**Lee**		**Williams82**		**PI48346**		**W05**	
	6d-SCPSF	60d-SPF	6d-SCPSF	60d-SPF	6d-SCPSF	60d-SPF	6d-SCPSF	60d-SPF
5	0.7227	0.6654	0.7409	0.603	0.7243	0.6434	0.7259	0.6563
10	0.9299	0.8207	1.0339	0.7689	0.9302	0.8636	1.0193	0.8644
15	1.1504	1.1391	1.2841	1.0555	1.2587	1.137	1.1937	1.0727
20	1.1829	1.1382	1.2863	1.1908	1.2611	1.153	1.2546	1.2559
25	1.1861	1.1725	1.2663	1.1741	1.244	1.176	1.1998	1.1767
30	1.2772	1.2456	1.3602	1.629	1.3	1.1778	1.2694	1.2058

SRSIO-FCM is much better for the 60d-SPF than 6d-SCPSF. As we can see, for 6d-SCPSF DBI values are higher than 60d-SPF when clustered using SRSIO-FCM on all the clusters except cluster30. Moreover, SRSIO-FCM attained a low value on cluster5 for 60d-SPF. In this way, we can conclude that 60d-SPF performs better than 6d-SCPSF when applied on the SRSIO-FCM algorithm in terms of DBI values for the Williams82 protein dataset.

Observing the DBI values for the PI483463 dataset, the value achieved by SRSIO-FCM is much better for the 60d-SPF than 6d-SCPSF. As we can see, for 6d-SCPSF DBI values are higher than 60d-SPF when clustered using SRSIO-FCM on almost all the clusters. Moreover, SRSIO-FCM attained a low value on cluster5 for 60d-SPF. In this way, we can conclude that 60d-SPF performs better than 6d-SCPSF when applied on SRSIO-FCM algorithm in terms of DBI values for the PI483463 protein dataset.

Observing the DBI values for the W05 dataset, the value achieved by SRSIO-FCM is much better for the 60d-SPF than 6d-SCPSF. As we can see, with the 6d-SCPSF approach the DBI values are higher than 60d-SPF when clustered

using SRSIO-FCM on almost all the clusters. Moreover, SRSIO-FCM attained a low value on cluster5 for 60d-SPF. In this way, we can conclude that 60d-SPF performs better than 6d-SCPSF when applied to the SRSIO-FCM algorithm in terms of DBI values for the W05 protein dataset.

6.4 Summary

In this chapter, two approaches 60d-SPF and 6d-SCPSF have been proposed to extract numerical feature vectors from huge protein sequences using Apache Spark cluster. These preprocessed numerical feature vectors are applied to the developed scalable fuzzy clustering algorithm. Here, we have used the SRSIO-FCM algorithm to cluster huge soybean protein sequences. The preprocessed feature vectors of protein sequences obtained from 60d-SPF and 6d-SCPSF are used as an input to the SRSIO-FCM algorithm. Our proposed 6d-SCPSF and 60d-SPF approaches are used to efficiently extract feature vectors from protein sequences. Thus, both the algorithms are scalable and can handle a huge amount of protein sequences. We have directed the exact evaluation of both the algorithms applied to SRSIO-FCM using different soybean protein datasets, which exhibited potential advantages for utilizing our methodology for clustering protein sequences.

At this point in this thesis, we have developed preprocessing approach for huge SNP and protein data. Then, we have applied proposed scalable fuzzy clustering algorithms for the clustering of massive SNP and protein data. In the next chapter, we focus on preprocessing and clustering of massive SARS-CoV-2 protein sequences using proposed approaches.

Chapter 7

Investigation of Massive SARS-CoV-2 Protein Datasets on Developed Scalable Feature Extraction and Scalable Fuzzy Clustering Algorithms

In the previous chapters (Chapter 5 and 6), all proposed scalable algorithms were applied for clustering of SNPs and proteins from plant genome sequences. In this chapter, we propose to investigate the preprocessing and clustering of massive protein sequences of Coronavirus Disease-19 (COVID-19) caused by the Severe Acute Respiratory Syndrome Coronavirus-2 (SARS-CoV-2) virus using these scalable algorithms. We have used 60-dimensional Scalable Feature (60d-SPF) extraction method (discussed in Chapter 6) to preprocess SARS-CoV-2 protein sequences. Furthermore, we have applied the preprocessed SARS-CoV-2 protein data to the developed scalable kernelized clustering algorithms (discussed in Chapter 3). The preprocessing and clustering of massive SARS-CoV-2 protein data are discussed in detail in the subsequent section.

7.1 Introduction

COVID-19 is a disease caused by the SARS-CoV-2 virus which has been declared a pandemic by the World Health Organization on March 11, 2020. COVID-19 has placed immense stress on the world's healthcare system. The amount of COVID-19 data generated by next generation sequencing technologies are increasingly huge [247, 248, 249]. As the amount of techniques used in data collection keeps expanding, the amount of observations that could be used as training data also keeps increasing. The problem becomes more rigid and more significant when each of these data samples contains multiple features or attributes [250]. The innovative COVID-19 analysis technologies need to be developed, which can significantly reduce the time and cost of clustering of SARS-CoV-2 genomes [251]. Since April 2020, a lot of research [252, 190, 253] is going on and a wide range of applications and methods of machine learning has been identified to overcome medical challenges and to predict the outbreak of the COVID-19 pandemic.

The SARS-CoV-2 genome data is growing faster than the rate at which it can be analyzed [232, 231]. It is becoming increasingly popular to investigate valuable information from huge SARS-CoV-2 data to interpret data practically and timely. To extract useful information from huge genomics data such as protein, DNA, and RNA sequences, many machine learning approaches are widely applied, such as clustering, classification, and neural network, etc [254]. There are several approaches applied for clustering of SARS-CoV-2 data [255, 256, 257], but the application of SARS-CoV-2 sequencing using clustering is rare [258]. These techniques do not receive a great acknowledgment for huge SARS-CoV-2 genome data as they remained non-scalable while there have been rapid advancements in clustering algorithms. The non-scalability has led to the confinement usage of many clustering algorithms at enormous information scales. Hence, there is a great need to design scalable clustering algorithms to handle massive SARS-CoV-2 genome data continuously generated from various sources. We have developed scalable kernelized fuzzy clustering algorithms (discussed in Chapter 3), which can deal with massive SARS-CoV-2 genome data. The KSRSIO-FCM per-

forms better than KSLFCM for handling Big Data (discussed in Chapter 3). A good representation is necessary for the proper clustering of genome data using the developed clustering methods. Hence, we have developed feature extraction approaches (discussed in Chapters 5 and 6) to preprocess huge genome data. The 60d-SPF method exhibited potential advantages for preprocessing massive protein sequences presented in Chapter 6. This chapter presents the preprocessing of massive SARS-CoV-2 protein data using developed 60d-SPF approach (discussed in Section 6.1). Thereafter, clustering of preprocessed SARS-CoV-2 data is performed using developed scalable kernelized fuzzy clustering methods; KSRSIO-FCM and KSLFCM. Further, the effectiveness of KSRSIO-FCM and KSLFCM on the SARS-CoV-2 dataset is performed in terms of SI and DBI measures. The preprocessing of SARS-CoV-2 protein data is presented in the subsequent section.

7.2 Preprocessing of SARS-CoV-2 Protein Datasets

In this section, the preprocessing of SARS-CoV-2 protein data is performed using the 60d-SPF method. The developed scalable protein preprocessing approach is being implemented using the Apache Spark framework and thus produces 60-dimensional numeric feature vectors. The workflow of preprocessing of the SARS-CoV-2 protein sequence using the 60d-SPF extraction approach is shown in Figure 7.1. We can see in Figure 7.1 that first raw SARS-CoV-2 protein data is used as input to the 60d-SPF method for feature extraction. Then the 60d-SPF method produces 60-dimensional numeric feature vectors as output. The preprocessing steps of SARS-CoV-2 protein data using an illustrative example are presented next.

Illustrative Example:

We have taken five SARS-CoV-2 sequences from SARS dataset[1]. The detailed description is presented in Section 2.6.3. In Figure 7.2, we represent five sequences of SARS-CoV-2 protein data taken from the SARS dataset. To show the steps of

[1]https://www.ncbi.nlm.nih.gov/datasets/coronavirus/genomes/

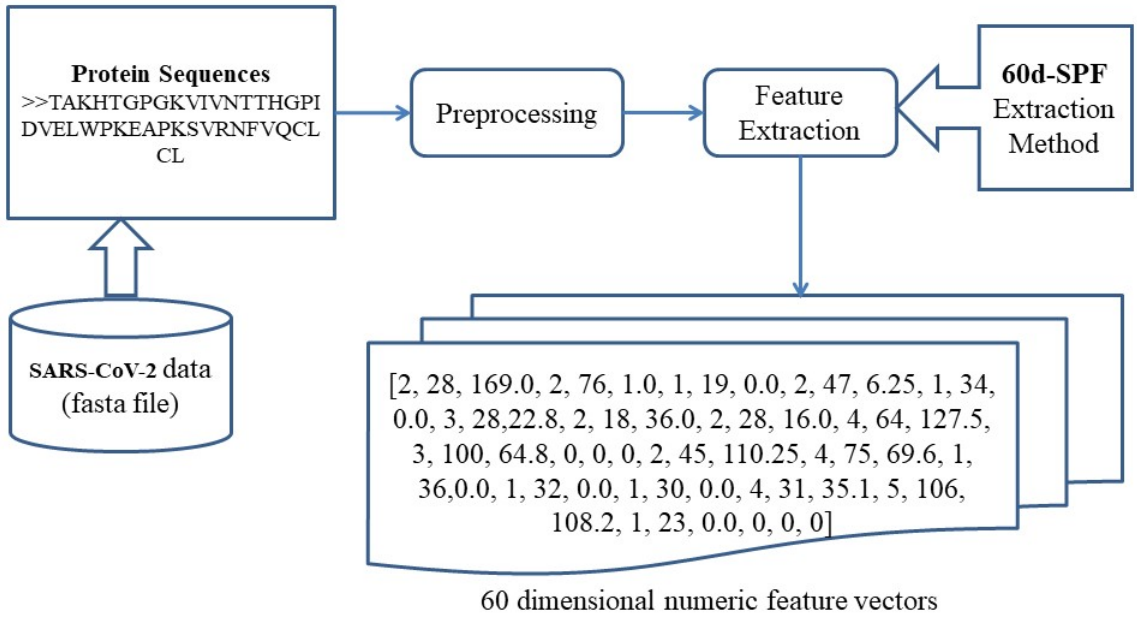


Figure 7.1: Workflow of the preprocessing of SARS-CoV-2 protein sequence.

```
>AAP41042.1:1-63 Orf7 [organism=SARS coronavirus Tor2] [isolate=Tor2]
MFHLVDFQVTIAEILIIIMRTFRIAIWNLDVIISSIVRQLFKPLTKKNYSELDDEEPMELDYP
>AAP41043.1:1-122 Orf8 [organism=SARS coronavirus Tor2] [isolate=Tor2]
MKIILFLTLIVFTSCELYHYQECVRGTTVLLKEPCPSGTYEGNSPFHPLADNKFALTCTSTHFAFACADG
TRHTYQLRARSVSPKLFIRQEEVQQELYSPLFLIVAALVFLILCFTIKRKTE
>AAP41044.1:1-44 Orf9 [organism=SARS coronavirus Tor2] [isolate=Tor2]
MNELTLIDFYLCFLAFLLFLVLIMLIIFWFSLEIQDLEEPCTKV
>AAP41045.1:1-39 Orf10 [organism=SARS coronavirus Tor2] [isolate=Tor2]
MKLLIVLTCISLCSCICTVVQRCASNKPHVLEDPCKVQH
>AAP41046.1:1-84 Orf11 [organism=SARS coronavirus Tor2] [isolate=Tor2]
MCLKILVRYNTRGNTYSTAWLCALGKVLPFHRWHTMVQTCTPNVTINCQDPAGGALIARCWYLHEGHQTA
AFRDVLVVLNKRTN
```

Figure 7.2: Five SARS-CoV-2 sequences from SARS dataset.

SARS-CoV-2 protein preprocessing, we have applied these sequences to the 60d-SPF extraction method. The 60d-SPF works in three stages; the first stage calculates the amino acid length for each sequence. The second stage counts the total distance of each amino acid to the first, and the third stage calculates the variance of the distance of each amino acid. For example, twenty amino acids will generate a feature vector consist of twenty numeric values after calculating the length of each amino acid. Likewise, we get twenty amino acids for the second and third stages. Hence, we have obtained 60-dimensional numeric feature vectors corresponding to SARS-

```
1, 2, 35, 42.25, 0, 0, 0, 5, 199, 411.76, 5, 229, 292.16, 4, 68, 230.5, 0, 0, 0, 1, 2, 0.0, 10, 217, 69.41, 3, 132, 4.666666666666667, 7, 237,
345.2653061224449, 3, 75, 566.0, 2, 74, 100.0, 3, 160, 70.22222222222223, 2, 45, 240.25, 3, 78, 62.0, 3, 116, 53.555555555555556, 3, 73, 213.55555555555557,
4, 78, 188.75, 1, 26, 0.0, 2, 109, 42.25

2, 8, 590, 426.6875, 6, 307, 1121.4722222222222, 2, 118, 81.0, 8, 509, 1468.734375, 10, 654, 1344.6399999999999, 4, 172, 260.0, 4, 198, 422.25, 7, 435,
 2550.693877551020 7, 6, 407, 1934.4722222222222, 16, 992, 1639.75, 1, 0, 0.0, 2, 93, 20.25, 6, 343, 651.1388888888889, 5, 375, 802.4000000000001, 6, 462,
792.0, 7, 414, 794.4081632653306, 12, 666, 1232.75, 7, 450, 1537.6326530612246, 0, 0, 0, 5, 248, 1019.8399999999999

3, 1, 14, 0.0, 2, 51, 210.25, 2, 42, 196.0, 4, 109, 217.6875, 6, 109, 57.80555555555556, 0, 0, 0, 0, 0, 5, 112, 80.24, 1, 42, 0.0, 11, 195,
93.28925619834712, 2, 23, 132.25, 1, 1, 0.0, 1, 39, 0.0, 1, 34, 0.0, 0, 0, 1, 30, 0.0, 2, 45, 342.25, 2, 63, 132.25, 1, 28, 0.0, 1, 9, 0.0

4, 1, 23, 0.0, 6, 106, 71.22222222222221, 1, 32, 0.0, 1, 31, 0.0, 0, 0, 0, 0, 0, 0, 2, 66, 25.0, 3, 28, 20.22222222222222, 3, 62, 206.88888888888889,
5, 52, 105.84000000000002, 1, 0, 0.0, 1, 25, 0.0, 2, 60, 9.0, 2, 57, 72.25, 1, 21, 0.0, 3, 47, 36.22222222222223, 2, 24, 25.0, 5, 107, 111.44, 0, 0, 0,
 0, 0, 0

5, 7, 342, 380.97959183673476, 5, 167, 415.0399999999996, 2, 123, 156.25, 1, 64, 0.0, 2, 101, 462.25, 5, 206, 394.16, 4, 192, 274.5, 3, 105,
 500.6666666666663, 3, 109, 1078.2222222222222, 9, 349, 782.1728395061729, 2, 35, 306.25, 6, 274, 845.5555555555555, 3, 119, 81.55555555555556, 3,
152, 153.55555555555554, 6, 262, 851.2222222222222, 1, 16, 0.0, 9, 348, 530.8888888888889, 7, 341, 697.6326530612243, 3, 111, 292.6666666666667, 3,
 84, 552.6666666666667
```

Figure 7.3: 60-dimensional numerical feature vectors of Figure 7.2.

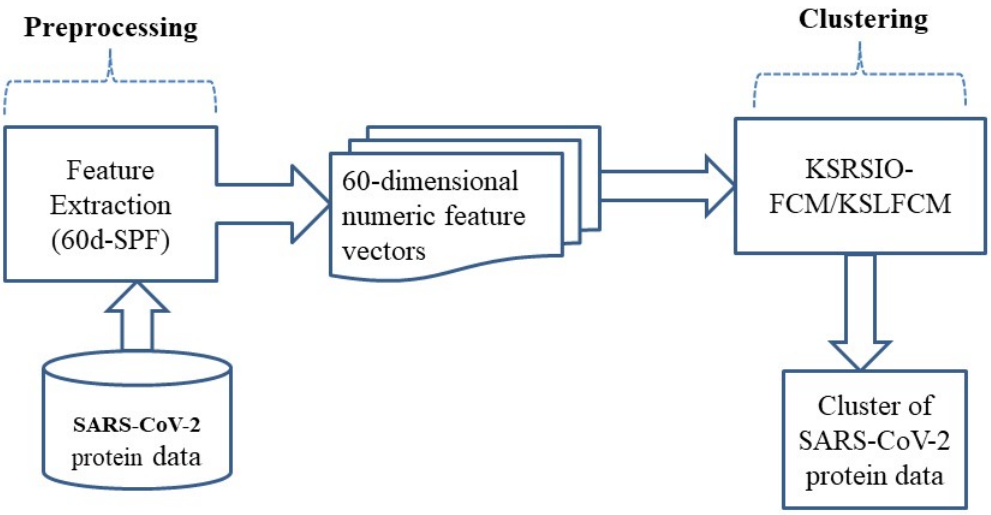


Figure 7.4: Workflow of the developed clustering algorithms applied to massive SARS-CoV-2 protein data.

CoV-2 protein sequences after all three stages of the 60d-SPF method. A detailed description of all three stages is presented in Section 6.1. Figure 7.3 represents 60-dimensional numerical feature vectors of Figure 7.2 obtained after applying the 60d-SPF method. The 60-dimensional numerical feature vectors are used as input to the developed scalable kernelized fuzzy clustering algorithm, which is presented next.

7.3 Clustering of SARS-CoV-2 Protein Datasets

In this section, we have used KSRSIO-FCM and KSLFCM methods to cluster SARS-CoV-2 protein datasets. The workflow of clustering of SARS-CoV-2 protein sequences using the KSRSIO-FCM and KSLFCM approaches is shown in Figure 7.4. First, we provide 60-dimensional numeric feature vectors as input to the KSRSIO-FCM and KSLFCM algorithms. The developed KSRSIO-FCM and KSLFCM approach starts by randomly partitioning the data into various subsets. Here, we have divided the SARS-CoV-2 protein sequences into three subsets. In this method, for clustering of the first subset, cluster centers are initialized randomly. After clustering the first subset, the cluster centers and membership values corresponding to the first subset

is obtained. After clustering the first subset, the final cluster centers are used as an input for clustering the second subset. After clustering of the second subset, the cluster centers, and membership value is obtained. After that, it combines the membership value of the first and second subset to compute the new cluster centers. These cluster centers are then fed as an input for clustering of the third subset. Then, after clustering of this subset, the final cluster centers, and final membership value is obtained. The developed KSRSIO-FCM and KSLFCM approach is being implemented using Apache Spark framework, which produces clusters consists of SARS-CoV-2 protein sequences. A detailed description of KSRSIO-FCM and KSLFCM algorithms is presented in Chapter 3. To evaluate the performance of KSRSIO-FCM and KSLFCM approaches on SARS-CoV-2 protein datasets, we fix the value of fuzzification parameter $p = 1.75$ and termination criteria $\epsilon = 0.01$ used in the experimental study. The performance analysis of scalable kernelized fuzzy clustering algorithms on SARS-CoV-2 protein datasets is presented next.

7.4 Experimental Analysis of SARS-CoV-2 Protein Datasets

To investigates the SARS-CoV-2 protein datasets, we have collected them from multiple sources[2][3]. In this section, we present the experimental results on various SARS-CoV-2 protein datasets which are applied to our developed scalable kernelized fuzzy clustering algorithms. The developed KSRSIO-FCM and KSLFCM algorithms take the preprocessed numeric feature vectors of SARS-CoV-2 protein sequences as input and produce output in the form of clusters. The efficacy of KSRSIO-FCM and KSLFCM algorithms is tested with various benchmark datasets, including Big Data. These are also tested with real-life SNP data, i.e., plant genome data. In this chapter, these proposed algorithms are tested with SARS-CoV-2 datasets. The experimental results were conducted to demonstrate the effectiveness of KSRSIO-FCM

[2]https://www.ebi.ac.uk/pdbe/covid-19
[3]https://www.ncbi.nlm.nih.gov/datasetscoronavirus/genomes/

in comparison with KSLFCM on SARS-CoV-2 protein datasets in terms of SI and DBI measures (discussed in Section 2.5.2). The detailed description of the SARS-CoV-2 datasets used in the experimental study is discussed in the subsequent section.

7.4.1 Datasets Description

The detailed description of the SARS-CoV-2 datasets are used in the experimentation is given in Section 2.6.3 and their characterization for the experimental analysis is presented in Table 7.1.

Table 7.1: Description of SARS-CoV-2 protein Datasets.

Parameters	**Datasets**		
	SARS	**Coronaviridae**	**P0DTD1**
sequences	93274	96596	1348
size	2.7GB	2.8 GB	367 KB

We have used two variants of Coronavirus genomes taken from NCBI and their clustering is performed using proposed kernelized clustering algorithms (KSRSIO-FCM and KSLFCM). The SARS and Coronaviridae are massive SARS-CoV-2 genome data (in GBs). We have performed preprocessing using developed 60d-SPF (discussed in Section 6.1) and then performed clustering using developed KSRSIO-FCM. The proposed 60d-SPF; a preprocessing and KSRSIO-FCM; a clustering algorithm is also tested on another SARS-CoV-2 P0DTD1 in Kilobytes (KBs), showing that proposed algorithms for preprocessing and clustering can be applicable for any size of SARS-CoV-2 protein data.

7.4.2 Clustering Performance on huge SARS-CoV-2 protein datasets

We perform clustering with KSRSIO-FCM by dividing the entire data into three subsets represented by subsets3 where KSLFCM is applied on the whole data (100%

data). The clustering is performed by taking the different number of clusters, i.e., 2, 3, 4, and 5 represented as cluster2, cluster3, cluster4, and cluster5, respectively.

Table 7.2: Results of KSRSIO-FCM and KSLFCM in terms of SI for SARS, Coronaviridae, and P0DTD1 protein dataset.

7.2(a) Results of SARS

Cluster	KSRSIO-FCM	KSLFCM
2	0.8175	-0.19304
3	0.0955	-0.2003
4	0.1038	-0.1976
5	0.3299	-0.2062

7.2(b) Results of Coronaviridae

Cluster	KSRSIO-FCM	KSLFCM
2	0.857	0.00651
3	0.0851	-0.0007
4	0.2187	-0.0232
5	0.0907	-0.0685

7.2(c) Results of P0DTD1

Cluster	KSRSIO-FCM	KSLFCM
2	0.3456	-0.19304
3	0.4348	-0.2003
4	0.3313	-0.1976
5	0.4198	-0.2062

Table 7.2 shows the results of KSRSIO-FCM and KSLFCM in terms of SI for SARS, Coronaviridae, and the P0DTD1 protein dataset. The SI demonstrates the quality of clustering. Subsequently, a superior clustering would bring about higher SI. Observing the SI values for SARS, Coronaviridae, and P0DTD1 datasets, respectively, presented in Table 7.2(a), 7.2(b), and 7.2(c), the KSRSIO-FCM algorithm has obtained positive values for almost all the clusters as compared to the KSLFCM algorithm. On the other hand, the KSRSIO-FCM achieved a higher value for cluster2 for SARS and Coronaviridae. For P0DTD1, KSRSIO-FCM achieved a higher value for cluster3. Along these lines, we can conclude that the proposed KSRSIO-FCM performs better than KSLFCM in terms of SI values for all three SARS-CoV-2 protein datasets.

Conversely to the SI, the DBI is not bounded within a given range. As a general rule, the lower the DBI value better the clustering result. Table 7.3 shows the result of KSRSIO-FCM and KSLFCM in terms of DBI for SARS, Coronaviridae, and the P0DTD1 protein dataset. The DBI demonstrates the quality of clustering. Sub-

Table 7.3: Results of KSRSIO-FCM and KSLFCM in terms of DBI for SARS, Coronaviridae, and P0DTD1 protein dataset.

7.3(a) Results of SARS

cluster	KSRSIO-FCM	KSLFCM
2	0.8809	8.9042
3	2.3923	15.964
4	1.5848	20.2402
5	3.1483	25.491

7.3(b) Results of Coronaviridae

cluster	KSRSIO-FCM	KSLFCM
2	0.9714	25.271
3	2.4082	46.831
4	3.5031	55.3994
5	2.7414	57.218

7.3(c) Results of P0DTD1

cluster	KSRSIO-FCM	KSLFCM
2	1.6699	8.9042
3	1.5233	15.964
4	2.8153	20.2402
5	1.5756	25.491

sequently, a superior clustering would bring about lower DBI. Observing the DBI values for SARS, Coronaviridae, and P0DTD1 datasets, respectively, presented in Table 7.3(a), 7.3(b), and 7.3(c), the KSRSIO-FCM algorithm has obtained lower values for almost all the clusters as compared to the KSLFCM algorithm. On the other hand, the KSRSIO-FCM achieved a lower value for cluster2 for SARS and Coronaviridae dataset. For P0DTD1, KSRSIO-FCM achieved a lower value for cluster3. Along these lines, we can conclude that the proposed KSRSIO-FCM performs better than KSLFCM in terms of DBI values for all three SARS-CoV-2 protein datasets.

7.5 Summary

In this chapter, massive protein sequences of COVID-19 caused by the SARS-CoV-2 virus are investigated using newly developed scalable algorithms. The scalable 60d-SPF, feature extraction technique and clustering; KSRSIO-FCM/KSLFCM are applied on massive SARS-CoV-2 protein datasets. First, a scalable feature extraction technique (60-SPF) is applied on massive SARS-CoV-2 protein datasets to extract the

60-dimensional numeric vectors for each protein sequence. After that, the preprocessed feature vectors of protein sequences obtained from the 60d-SPF method are used as input to the KSRSIO-FCM and KSLFCM algorithms. We have performed the investigation on various SARS-CoV-2 protein datasets using the 60d-SPF extraction method applied to the KSRSIO-FCM/KSLFCM algorithm in terms of SI and DBI measures. The efficacy of KSRSIO-FCM and KSLFCM algorithms is tested with various benchmark datasets, including Big Data. These are also tested with real-life SNP data, i.e., plant genome data. In this chapter, these proposed algorithms are tested with SARS-CoV-2 datasets. However, in this chapter, these developed algorithms are applied to massive SARS-CoV-2 protein datasets. It is found that KSRSIO-FCM performs better than KSLFCM. Hence, it is observed that our proposed algorithm can handle any kind of genome data of any size (Big Data).

Chapter 8

Conclusions and Future Work

This thesis primarily investigates the scalable fuzzy clustering algorithms for clustering of Big Data, and these developed algorithms are applied on the massive genome sequences. In particular, we have developed scalable kernelized fuzzy clustering algorithms and scalable incremental fuzzy consensus clustering algorithms using Apache Spark cluster. First, we have developed a scalable kernelized clustering algorithm named Kernelized Scalable Random Sampling with Iterative Optimization Fuzzy C-Means (KSRSIO-FCM). This is based on the Kernelized Scalable Literal Fuzzy C-Means (KSLFCM) clustering algorithm, in which kernel function (RBF) is used. Further, a Scalable Incremental Fuzzy Consensus Clustering (SIFCC) algorithm using Apache Spark cluster is developed to improve the quality of clusters for Big Data. This method combines SRSIO-FCM and Scalable Fuzzy Consensus Clustering (SFCC) algorithms, in which SRSIO-FCM is used to generate Basic Segments (BSs), and SFCC is an integral part of SIFCC. These proposed algorithms have been tested on various benchmark datasets and compared results with various scalable fuzzy clustering algorithms. Results analysis exhibits that the proposed algorithms have outperformed existing scalable fuzzy clustering algorithms in terms of NMI, ARI, and F-score, respectively. In order to test the proposed scalable fuzzy clustering algorithms on real-life, massive genome data, we have designed a feature extraction technique for the genome dataset. First, we have developed a scalable preprocessing feature extraction approach for huge SNP sequences, producing a 12-dimensional numeric feature vector

that is used as an input to the proposed KSRSIO-FCM and SIFCC algorithms. Then, scalable feature extraction techniques for huge protein sequences are developed, i.e., 60d-SPF and 6d-SCPSF, producing 60-dimensional and 6-dimensional numeric feature vectors. Finally, to tackle massive SARS-CoV-2 protein data challenges, we have applied developed scalable feature extraction and clustering algorithms on massive SARS-CoV-2 data. These proposed feature extraction methods have been tested on various genome datasets and compared results with various scalable fuzzy clustering algorithms. Results analysis exhibits that the proposed algorithms have outperformed existing scalable fuzzy clustering algorithms in terms of SI and DBI measures. The proposed scalable fuzzy clustering algorithms are general-purpose which can be applied to any problem. Also, the developed scalable feature extraction techniques are the generalized approaches that can be applied to any SNP (nucleotide form) and protein (amino acid form) sequences.

8.1 Summary of Research Achievements

The objectives specified in Section 1.2 have been successfully fulfilled by the following main contributions:

(i) **Scalable Kernelized Fuzzy Clustering Algorithms for Handling Big Data:**

We have proposed a Kernelized Scalable Random Sampling with Iterative Optimization Fuzzy C-Means (KSRSIO-FCM), which is based on the Kernelized Scalable Literal Fuzzy C-Means (KSLFCM) clustering algorithm. The scalable kernelized clustering algorithms are evolved to deal with the non-linear separable problems by applying a kernel Radial Basis Functions (RBF), which maps the input data space non-linearly into a high dimensional feature space. The proposed scalable kernelized fuzzy clustering algorithms for handling Big Data reduces the run-time and optimizes the storage space. Further, an analytical formulation for space and time complexity is developed for KSRSIO-FCM and KSLFCM.

The performance of KSRSIO-FCM is compared with KSLFCM, SRSIO-FCM, and SLFCM algorithms. We have performed experimentation on four replicated benchmark datasets to test the performance of the KSRSIO-FCM. The KSRSIO-FCM exhibited better performance than KSLFCM, SRSIO-FCM, and SLFCM algorithms in terms of NMI, ARI, and F-score, respectively. The scalable fuzzy clustering algorithm is further improved by combining consensus clustering using Apache Spark to enhance the quality of clusters.

(ii) **Scalable Incremental Fuzzy Consensus Clustering Algorithms for Handling Big Data:**

We have also designed a Scalable Incremental Fuzzy Consensus Clustering (SIFCC) algorithm using Apache Spark cluster, which is based on the Scalable Fuzzy Consensus Clustering (SFCC) algorithm. The scalable fuzzy consensus clustering algorithms consider the set of partitions as input to find the cluster. For this purpose, data is to be clustered using SRSIO-FCM r number of times to obtain r BSs by varying the number of clusters. Hence, the SIFCC improves the quality of clusters for handling Big Data. Further, an analytical formulation for space and time complexity is developed for SIFCC and SFCC algorithms. The performance of SIFCC is compared with SFCC and SRSIO-FCM. We have performed experimentation on three benchmark datasets and two Big Data to test the performance of the SIFCC. The SIFCC exhibited better performance than SFCC and SRSIO-FCM algorithms in terms of NMI, ARI, and F-score, respectively.

(iii) **Design of Novel Scalable Feature Extraction Algorithm for Huge SNP Sequences with Application of Scalable Fuzzy Clustering Algorithms:**

To handle huge SNP data, which is generally available in nucleotide form, we have preprocessed SNP sequences using the feature extraction technique with the Apache Spark cluster. For this purpose, we have developed a scalable feature extraction method for huge SNP sequences, which extracts a 12-dimensional numeric feature vector. Further, these 12-dimensional numeric feature vectors are

used to input the proposed scalable clustering algorithms to cluster massive SNP data. We have performed experimentation on four SNP datasets (1 soybean and 3 rice), and the same SNP data is tested with the scalable version of KSRSIO-FCM, KSLFCM, SRSIO-FCM, and SLFCM algorithms for the sake of comparison. We have also tested the results of SNP data on developed scalable fuzzy consensus-based SIFCC and SFCC algorithms in terms of SI and DBI measures.

(iv) **Design of Novel Scalable Feature Extraction Algorithms for Huge Protein Sequences with Application of Scalable Fuzzy Clustering Algorithm:**

To handle huge protein data, which is generally available in amino acid form, we have preprocessed protein sequences using 60-dimensions and 6-dimensions feature extraction techniques executed on the Apache Spark cluster. In this way, we have developed two novel scalable feature extraction algorithms named 60d-SPF and 6d-SCPSF for massive protein sequences, which extracts numeric feature vectors of 60-dimensions and 6-dimensions fixed-length. Further, this 60-dimensional and 6-dimensional numeric feature vectors are used as input to the SRSIO-FCM algorithm to cluster massive protein data. We have performed experimentation with four soybean datasets to test the performance of both algorithms using the SRSIO-FCM in terms of SI and DBI measures.

(v) **Investigation of Massive SARS-CoV-2 Protein Datasets on Developed Scalable Feature Extraction and Scalable Fuzzy Clustering Algorithms:**

The proposed scalable fuzzy clustering algorithms are tested with various benchmark datasets, including Big Data. These are also tested with real-life SNP data, i.e., plant genome data. Like plant genome sequences available in abundant amounts, SARS-CoV-2 genome sequences are also available massively for a clinical study. Therefore, we have investigated massive SARS-CoV-2 protein datasets using proposed scalable feature extraction and scalable kernelized fuzzy clustering algorithms. First, we have performed the preprocessing of various

SARS-CoV-2 protein datasets using our developed 60d-SPF extraction method (given in Section 6.1), which generates a 60-dimensional numeric feature vector. Then, these feature vectors are used as input to the developed KSRSIO-FCM and KSLFCM algorithms (given in Chapter 3) for clustering of SARS-CoV-2 protein datasets. We have performed experimentation on three SARS-CoV-2 protein datasets to test the efficacy of KSRSIO-FCM and KSLFCM applied to SARS-CoV-2 protein datasets. The experimental analysis shows that KSRSIO-FCM performs better than KSLFCM for massive SARS-CoV-2 datasets. Hence, we can conclude that our proposed scalable feature extraction techniques and scalable fuzzy clustering algorithms can handle massive genome data of any type. Thus these algorithms can be integrated to develop a complete tool for a specific bioinformatics problem.

8.2 Future Research Directions

Despite significant progress in the topic of scalable fuzzy clustering for massive genome data, it can be explored in several interesting future directions as follows:

(i) **Data Stream Online Fuzzy Clustering based on Kernelized Scalable Random Sampling Iterative Optimization for handling massive Genome Data:**

All developed fuzzy-based scalable clustering algorithms are only suitable for stationary datasets, unable to handle non-stationary and streaming datasets. These types of data can be handled by online learning. In recent years, analysis of the concept drift problem by using fuzzy clustering approaches are developed [259, 260, 261]. These methods are not suitable for Big Data due to the high time and memory requirements. Our proposed scalable KSRSIO-FCM algorithm can be extended for online learning for Big Data.

(ii) **Scalable Deep Fuzzy Clustering Algorithms for handling massive Genome Data:**

The developed fuzzy clustering can be improved by handling high-dimensional data with a complex latent distribution. A deep fuzzy clustering method is developed by Feng [262], which represents the data in a feature space produced by the deep neural network. Both concepts can be combined using Apache Spark cluster for improving the performance in terms of NMI, ARI, and F-score. This combined architecture can be further applied to huge genome data for DNA, RNA, SNP, and protein sequence analysis, which can result in better-integrated tools for bioinformaticians.

Bibliography

[1] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, 1999.

[2] Vladimir Estivill Castro. Why so many clustering algorithms: a position paper. *ACM SIGKDD Explorations Newsletter*, 4(1):65–75, 2002.

[3] Joshua Zhexue Huang, Michael K Ng, Hongqiang Rong, and Zichen Li. Automated variable weighting in k-means type clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(5):657–668, 2005.

[4] Jia Li, Dongsheng Li, and Yiming Zhang. Efficient distributed data clustering on spark. In *2015 IEEE International Conference on Cluster Computing*, pages 504–505. IEEE, 2015.

[5] Giovanna Castellano, Anna M Fanelli, and Corrado Mencar. A fuzzy clustering approach for mining diagnostic rules. In *Proc. of 2003 IEEE International Conference on Systems, Man and Cybernetics*, pages 2007–2012. IEEE, Washington, D.C., USA, October, 2003.

[6] Hong Bin Shen, Jie Yang, Xiao Jun Liu, and Kuo Chen Chou. Using supervised fuzzy clustering to predict protein structural classes. *Biochemical and Biophysical Research Communications*, 334(2):577–581, 2005.

[7] Liping Jing, Michael K Ng, and Joshua Zhexue Huang. An entropy weighting k-means algorithm for subspace clustering of high-dimensional sparse data. *IEEE Transactions on Knowledge and Data Engineering*, 19(8):1026–1041, 2007.

[8] Zhaohong Deng, Kup Sze Choi, Fu Lai Chung, and Shitong Wang. Enhanced soft subspace clustering integrating within-cluster and between-cluster information. *Pattern Recognition*, 43(3):767–781, 2010.

[9] Manju Sardana and RK Agrawal. A comparative study of clustering methods for relevant gene selection in microarray data. In *Proc. of Second International Conference on Computer Science, Engineering and Applications, Advances in Computer Science, Engineering and Applications*, volume 166 of *Advances in Intelligent and Soft Computing*, pages 789–797. Springer Berlin Heidelberg, 2012.

[10] Lei Tang, Huan Liu, and Jianping Zhang. Identifying evolving groups in dynamic multimode networks. *IEEE Transactions on Knowledge and Data Engineering*, 24(1):72–85, 2012.

[11] Hichem Frigui and Raghu Krishnapuram. A robust competitive clustering algorithm with applications in computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5):450–465, 1999.

[12] Yee Leung, Jiang She Zhang, and Zong Ben Xu. Clustering by scale-space filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1396–1410, 2000.

[13] Richard O Duda, Peter E Hart, et al. *Pattern classification and scene analysis*, volume 3. Wiley Interscience New York, 1973.

[14] Richard Bellman, Robert Kalaba, and L Zadeh. Abstraction and pattern classification. *Journal of Mathematical Analysis and Applications*, 13(1):1–7, 1966.

[15] KaiLe Zhou, Chao Fu, and ShanLin Yang. Fuzziness parameter selection in fuzzy c-means: the perspective of cluster validation. *Science China Information Sciences*, 57(11):1–8, 2014.

[16] Soumadip Ghosh, Sushanta Biswas, Debasree Sarkar, and Partha Pratim Sarkar. A novel neuro-fuzzy classification technique for data mining. *Egyptian Informatics Journal*, 15(3):129–147, 2014.

[17] Firat Tekiner and John A Keane. Big data framework. In *2013 IEEE international conference on systems, man, and cybernetics*, pages 1494–1499. IEEE, 2013.

[18] Azlinah Mohamed, Maryam Khanian Najafabadi, Yap Bee Wah, Ezzatul Akmal Kamaru Zaman, and Ruhaila Maskat. The state of the art and taxonomy of big data analytics: view from new big data framework. *Artificial Intelligence Review*, 53(2):989–1037, 2020.

[19] Yinan Xu, Hui Liu, and Zhihao Long. A distributed computing framework for wind speed big data forecasting on apache spark. *Sustainable Energy Technologies and Assessments*, 37:100582, 2020.

[20] Panos Louridas and Christof Ebert. Embedded analytics and statistics for big data. *IEEE software*, 30(6):33–39, 2013.

[21] Jeyachandran Sivakamavalli, Kiyun Park, and Ihn-Sil Kwak. Genome databases, types and applications: An overview. 2020.

[22] Peter J Huber. Massive data sets workshop: the morning after. In *Proc. of a Workshop Massive Data Sets*, pages 169–184. Washington, DC: National Academy Press, 1997.

[23] Peter J Huber. Huge data sets. In *Compstat*, pages 3–13. Springer, 1994.

[24] Timothy C Havens, James C Bezdek, Christopher Leckie, Lawrence O Hall, and Marimuthu Palaniswami. Fuzzy c-means algorithms for very large data. *IEEE Transactions on Fuzzy Systems*, 20(6):1130–1146, 2012.

[25] Tai Wai Cheng, Dmitry B Goldgof, and Lawrence O Hall. Fast fuzzy clustering. *Fuzzy Sets and Systems*, 93(1):49–56, 1998.

[26] Ming Chuan Hung and Don Lin Yang. An efficient fuzzy c-means clustering algorithm. In *Proc. of 2001 IEEE International Conference on Data Mining*, pages 225–232. IEEE, San Jose, California, USA, December, 2001.

[27] Nikhil R Pal and James C Bezdek. Complexity reduction for large image processing. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 32(5):598–611, 2002.

[28] Richard J Hathaway and James C Bezdek. Extending fuzzy and probabilistic clustering to very large data sets. *Computational Statistics and Data Analysis*, 51(1):215–234, 2006.

[29] Prodip Hore, Lawrence O Hall, and Dmitry B Goldgof. Single pass fuzzy c means. In *Proc. of 2007 IEEE International on Fuzzy Systems Conference*, pages 1–7. IEEE, London, United Kingdom, July, 2007.

[30] Liang Wang, James C Bezdek, Christopher Leckie, and Ramamohanarao Kotagiri. Selective sampling for approximate clustering of very large data sets. *International Journal of Intelligent Systems*, 23(3):313–331, 2008.

[31] Prodip Hore, Lawrence O Hall, Dmitry B Goldgof, Yuhua Gu, Andrew A Maudsley, and Ammar Darkazanli. A scalable framework for segmenting magnetic resonance images. *Journal of Signal Processing Systems*, 54(1-3):183–203, 2009.

[32] Timothy C Havens, James C Bezdek, Christopher Leckie, Lawrence O Hall, and Marimuthu Palaniswami. Fuzzy c-means algorithms for very large data. *IEEE Transactions on Fuzzy Systems*, 20(6):1130–1146, 2012.

[33] James C Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Springer Science and Business Media, 2013.

[34] Neha Bharill, Aruna Tiwari, Aayushi Malviya, Om Prakash Patel, Akahansh Gupta, Deepak Puthal, Amit Saxena, and Mukesh Prasad. Fuzzy knowledge based performance analysis on big data. *Neurocomputing*, 389:218–228, 2020.

[35] Yin-Ping Zhao, Long Chen, and CL Philip Chen. Multiple kernel shadowed clustering in approximated feature space. In *International Conference on Data Mining and Big Data*, pages 265–275. Springer, 2018.

[36] Long Chen and Lingning Kong. Fuzzy clustering in high-dimensional approximated feature space. In *2016 International Conference on Fuzzy Theory and Its Applications (iFuzzy)*, pages 1–6. IEEE, 2016.

[37] Dao-Qiang Zhang and Song-Can Chen. Clustering incomplete data using kernel-based fuzzy c-means algorithm. *Neural processing letters*, 18(3):155–162, 2003.

[38] Mark Girolami. Mercer kernel-based clustering in feature space. *IEEE Transactions on Neural Networks*, 13(3):780–784, 2002.

[39] Hongfu Liu, Tongliang Liu, Junjie Wu, Dacheng Tao, and Yun Fu. Spectral ensemble clustering. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 715–724. ACM, 2015.

[40] Nam Nguyen and Rich Caruana. Consensus clusterings. In *Seventh IEEE international conference on data mining (ICDM 2007)*, pages 607–612. IEEE, 2007.

[41] Yiteng Zhai, Yew Soon Ong, and Ivor W Tsang. The emerging big dimensionality. *IEEE Computational Intelligence Magazine*, 9(3):14–26, 2014.

[42] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: a new platform for distributed machine learning on big data. *IEEE Transactions on Big Data*, 1(2):49–67, 2015.

[43] Zachary D Stephens, Skylar Y Lee, Faraz Faghri, Roy H Campbell, Chengxiang Zhai, Miles J Efron, Ravishankar Iyer, Michael C Schatz, Saurabh Sinha, and Gene E Robinson. Big data: astronomical or genomical? *PLoS biology*, 13(7):e1002195, 2015.

[44] Behrooz Hosseini and Kourosh Kiani. A big data driven distributed density based hesitant fuzzy clustering using apache spark with application to gene expression microarray. *Engineering Applications of Artificial Intelligence*, 79:100–113, 2019.

[45] Behrooz Hosseini and Kourosh Kiani. A robust distributed big data clustering-based on adaptive density partitioning using apache spark. *Symmetry*, 10(8):342, 2018.

[46] Daxin Jiang, Chun Tang, and Aidong Zhang. Cluster analysis for gene expression data: a survey. *IEEE Transactions on Knowledge & Data Engineering*, (11):1370–1386, 2004.

[47] Ge Wang, Pengbo Pu, and Tingyan Shen. An efficient gene bigdata analysis using machine learning algorithms. *Multimedia Tools and Applications*, 79(15):9847–9870, 2020.

[48] Claire Nédellec. Machine learning for information extraction in genomics—state of the art and perspectives. In *Text Mining and its Applications*, pages 99–118. Springer, 2004.

[49] Razvan Bunescu, Ruifang Ge, Rohit J Kate, Edward M Marcotte, Raymond J Mooney, Arun K Ramani, and Yuk Wah Wong. Comparative experiments on learning information extractors for proteins and their interactions. *Artificial intelligence in medicine*, 33(2):139–155, 2005.

[50] Goo Jun, Mary Kate Wing, Gonçalo R Abecasis, and Hyun Min Kang. An efficient and scalable analysis framework for variant extraction and refinement from population-scale dna sequence data. *Genome research*, 25(6):918–925, 2015.

[51] Mehrdad J Gangeh, Hadi Zarkoob, and Ali Ghodsi. Fast and scalable feature selection for gene expression data using hilbert-schmidt independence criterion. *IEEE/ACM transactions on computational biology and bioinformatics*, 14(1):167–181, 2017.

[52] Neha Bharill, Aruna Tiwari, and Aayushi Malviya. Fuzzy based scalable clustering algorithms for handling big data using apache spark. *IEEE Transactions on Big Data*, 2(4):339–352, 2016.

[53] Junjie Wu, Zhiang Wu, Jie Cao, Hongfu Liu, Guoqing Chen, and Yanchun Zhang. Fuzzy consensus clustering with applications on big data. *IEEE Transactions on Fuzzy Systems*, 25(6):1430–1445, 2017.

[54] David Grant, Rex T Nelson, Steven B Cannon, and Randy C Shoemaker. Soybase, the usda-ars soybean genetics and genomics database. *Nucleic acids research*, 38(suppl_1):D843–D846, 2010.

[55] Christine Jade Dilla-Ermita, Erwin Tandayu, Venice Margarette Juanillas, Jeffrey Detras, Dennis Nicuh Lozada, Maria Stefanie Dwiyanti, Casiana Vera Cruz, Edwige Gaby Nkouaya Mbanjo, Edna Ardales, Maria Genaleen Diaz, et al. Genome-wide association analysis tracks bacterial leaf blight resistance loci in rice diverse germplasm. *Rice*, 10(1):1–17, 2017.

[56] 000 Rice Genomes Project 3. The 3,000 rice genomes project. *GigaScience*, 3(1):2047–217X, 2014.

[57] Locedie Mansueto, Roven Rommel Fuentes, Frances Nikki Borja, Jeffery Detras, Juan Miguel Abriol-Santos, Dmytro Chebotarov, Millicent Sanciangco, Kevin Palis, Dario Copetti, Alexandre Poliakov, et al. Rice snp-seek database update: new snps, indels, and queries. *Nucleic acids research*, 45(D1):D1075–D1081, 2017.

[58] Olfa Nasraoui and Chiheb-Eddine Ben N'Cir. *Clustering Methods for Big Data Analytics: Techniques, Toolboxes and Applications*. Springer, 2018.

[59] Amit Saxena, Mukesh Prasad, Akshansh Gupta, Neha Bharill, Om Prakash Patel, Aruna Tiwari, Meng Joo Er, Weiping Ding, and Chin-Teng Lin. A review of clustering techniques and developments. *Neurocomputing*, 267:664–681, 2017.

[60] Doug Fisher, Ling Xu, James R Carnes, Yoram Reich, J Fenves, Jason Chen, Richard Shiavi, Gautam Biswas, and Jerry Weinberg. Applying ai clustering to engineering tasks. *IEEE Expert*, 8(6):51–60, 1993.

[61] Yu Kun and Wang Ling. Construction of university network ideology management platform based on constrained clustering algorithm. *Design Engineering*, pages 269–278, 2021.

[62] Qusay Alsarhan, Bestoun S Ahmed, Miroslav Bures, and Kamal Zuhairi Zamli. Software module clustering: An in-depth literature analysis. *IEEE Transactions on Software Engineering*, 2020.

[63] Guo Pu, Lijuan Wang, Jun Shen, and Fang Dong. A hybrid unsupervised clustering-based anomaly detection method. *Tsinghua Science and Technology*, 26(2):146–153, 2020.

[64] Johanna Barzen and Frank Leymann. Quantum humanities: A first use case for quantum-ml in media. *Digitale Welt*, 4:102–103, 2020.

[65] Mehmet Efe Biresselioglu, Muhittin Hakan Demir, Berfu Solak, Altan Kayacan, and Sebnem Altinci. Investigating the trends in arctic research: the increasing role of social sciences and humanities. *Science of The Total Environment*, 729:139027, 2020.

[66] Quan Zou, Gang Lin, Xingpeng Jiang, Xiangrong Liu, and Xiangxiang Zeng. Sequence clustering in bioinformatics: an empirical study. *Briefings in bioinformatics*, 21(1):1–10, 2020.

[67] Aldo Faisal, Erwann Le Lannou, Benjamin Post, Shlomi Haar, Stephen Brett, and Balasundaram Kadirvelu. Clustering of patient comorbidities within electronic medical records enables high-precision covid-19 mortality prediction. 2021.

[68] Kaijian Xia, Xiaoqing Gu, and Yudong Zhang. Oriented grouping-constrained spectral clustering for medical imaging segmentation. *Multimedia Systems*, 26(1):27–36, 2020.

[69] Yu Tian, Ruiqing Zheng, Zhenlan Liang, Suning Li, Fang-Xiang Wu, and Min Li. A data-driven clustering recommendation method for single-cell rna-sequencing data. *Tsinghua Science and Technology*, 26(5):772–789, 2021.

[70] Salim Khan, Gang Situ, Keith Decker, and Carl J Schmidt. Gofigure: Automated gene ontology™ annotation. *Bioinformatics*, 19(18):2484–2485, 2003.

[71] Stephan Gunnemann, Hardy Kremer, Dominik Lenhard, and Thomas Seidl. Subspace clustering for indexing high dimensional data: a main memory index based on local reductions and individual multi-representations. In *Proc. of 14th International Conference on Extending Database Technology*, pages 237–248. ACM, Uppsala, Sweden, March, 2011.

[72] Aurélien Ducournau, Alain Bretto, Soufiane Rital, and Bernard Laget. A reductive approach to hypergraph clustering: an application to image segmentation. *Pattern Recognition*, 45(7):2788–2803, 2012.

[73] Theam Foo Ng, Tuan D Pham, and Xiuping Jia. Feature interaction in subspace clustering using the choquet integral. *Pattern Recognition*, 45(7):2645–2660, 2012.

[74] Jung Yi Jiang, Ren Jia Liou, and Shie Jue Lee. A fuzzy self-constructing feature clustering algorithm for text classification. *IEEE Transactions on Knowledge and Data Engineering*, 23(3):335–349, 2011.

[75] Jan Feyereisl and Uwe Aickelin. Privileged information for data clustering. *Information Sciences*, 194:4–23, 2012.

[76] Saeid Soheily-Khah, Ahlame Douzal-Chouakria, and Eric Gaussier. Generalized k-means-based clustering for temporal data under weighted and kernel time warp. *Pattern Recognition Letters*, 75:63–69, 2016.

[77] Stuart P Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[78] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley and Sons, 2009.

[79] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[80] Adil Fahad, Najlaa Alshatri, Zahir Tari, Atif Alamri, Issa Khalil, Albert Y Zomaya, Sebti Foufou, and Abdelaziz Bouras. A survey of clustering algorithms for big data: taxonomy and empirical analysis. *IEEE Transactions on Emerging Topics in Computing*, 2(3):267–279, 2014.

[81] Erzhou Zhu and Ruhui Ma. An effective partitional clustering algorithm based on new clustering validity index. *Applied soft computing*, 71:608–621, 2018.

[82] Ahmad Taher Azar, Shaimaa Ahmed El-Said, and Aboul Ella Hassanien. Fuzzy and hard clustering analysis for thyroid disease. *Computer methods and programs in biomedicine*, 111(1):1–16, 2013.

[83] Christopher D Manning, Prabhakar Raghavan, Hinrich Schutze, et al. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.

[84] Joseph C Dunn. *A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters*. Taylor and Francis, 1973.

[85] James C Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Kluwer Academic Publishers, 1981.

[86] Abraham Kandel and William J Byatt. Fuzzy sets, fuzzy algebra, and fuzzy statistics. *Proceedings of the IEEE*, 66(12):1619–1639, 1978.

[87] Jonathon K Parker, Lawrence O Hall, and Abraham Kandel. Scalable fuzzy neighborhood dbscan. In *Proc. of 2010 IEEE International Conference on Fuzzy Systems*, pages 1–8. IEEE, Barcelona, Spain, July, 2010.

[88] Jonathon K Parker, Lawrence O Hall, and James C Bezdek. Comparison of scalable fuzzy clustering methods. In *Proc. of 2012 IEEE International Conference on Fuzzy Systems*, pages 1–9. IEEE, Brisbane, Australia, June, 2012.

[89] Francisco De AT De Carvalho and Camilo P Tenorio. Fuzzy k-means clustering algorithms for interval-valued data based on adaptive quadratic distances. *Fuzzy Sets and Systems*, 161(23):2978–2999, 2010.

[90] Mika Sato-Ilic. Symbolic clustering with interval-valued data. *Procedia Computer Science*, 6:358–363, 2011.

[91] Hua Zhao, Zeshui Xu, Shousheng Liu, and Zhong Wang. Intuitionistic fuzzy mst clustering algorithms. *Computers and Industrial Engineering*, 62(4):1130–1140, 2012.

[92] Carlos WD De Almeida, Renata MCR De Souza, and Ana LB Candeias. Fuzzy kohonen clustering networks for interval data. *Neurocomputing*, 99:65–75, 2013.

[93] Jonathon K Parker and Lawrence O Hall. Accelerating fuzzy-c means using an estimated subsample size. *IEEE Transactions on Fuzzy Systems,*, 22(5):1229–1244, 2014.

[94] James C Bezdek, Robert Ehrlich, and William Full. Fcm: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2-3):191–203, 1984.

[95] Hsin-Chien Huang, Yung-Yu Chuang, and Chu-Song Chen. Multiple kernel fuzzy clustering. *IEEE Transactions on Fuzzy Systems*, 20(1):120–134, 2011.

[96] Naouel Baili and Hichem Frigui. Fuzzy clustering with multiple kernels in feature space. In *2012 IEEE International Conference on Fuzzy Systems*, pages 1–8. IEEE, 2012.

[97] Timothy C Havens, James C Bezdek, Christopher Leckie, Lawrence O Hall, and Marimuthu Palaniswami. Fuzzy c-means algorithms for very large data. *IEEE Transactions on Fuzzy Systems*, 20(6):1130–1146, 2012.

[98] Jingwei Liu and Meizhi Xu. Kernelized fuzzy attribute c-means clustering algorithm. *Fuzzy sets and systems*, 159(18):2428–2445, 2008.

[99] Daniel Graves and Witold Pedrycz. Kernel-based fuzzy clustering and fuzzy clustering: A comparative experimental study. *Fuzzy sets and systems*, 161(4):522–543, 2010.

[100] Du-Ming Tsai and Chung-Chan Lin. Fuzzy c-means based clustering for linearly and nonlinearly separable data. *Pattern recognition*, 44(8):1750–1760, 2011.

[101] Tianhao Li, Liyong Zhang, Wei Lu, Hui Hou, Xiaodong Liu, Witold Pedrycz, and Chongquan Zhong. Interval kernel fuzzy c-means clustering of incomplete data. *Neurocomputing*, 237:316–331, 2017.

[102] Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, (3):326–334, 1965.

[103] Weiling Cai, Songcan Chen, and Daoqiang Zhang. Robust fuzzy relational classifier incorporating the soft class labels. *Pattern Recognition Letters*, 28(16):2250–2263, 2007.

[104] Chandan Gautam, Aruna Tiwari, Sundaram Suresh, and Kapil Ahuja. Adaptive online learning with regularized kernel for one-class classification. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.

[105] Junjie Wu, Hongfu Liu, Hui Xiong, Jie Cao, and Jian Chen. K-means-based consensus clustering: A unified view. *IEEE transactions on knowledge and data engineering*, 27(1):155–169, 2014.

[106] Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617, 2002.

[107] Witold Pedrycz and Partab Rai. Collaborative clustering with the use of fuzzy c-means and its quantification. *Fuzzy Sets and Systems*, 159(18):2399–2427, 2008.

[108] Kunal Punera and Joydeep Ghosh. Consensus-based ensembles of soft clusterings. *Applied Artificial Intelligence*, 22(7-8):780–810, 2008.

[109] Xue Li and Hongfu Liu. Greedy optimization for k-means-based consensus clustering. *Tsinghua Science and Technology*, 23(2):184–194, 2018.

[110] Hongfu Liu, Junjie Wu, Tongliang Liu, Dacheng Tao, and Yun Fu. Spectral ensemble clustering via weighted k-means: Theoretical and practical evidence. *IEEE transactions on knowledge and data engineering*, 29(5):1129–1143, 2017.

[111] Ana LN Fred and Anil K Jain. Combining multiple clusterings using evidence accumulation. *IEEE transactions on pattern analysis and machine intelligence*, 27(6):835–850, 2005.

[112] Zhiwu Lu, Yuxin Peng, and Jianguo Xiao. From comparing clusterings to combining clusterings. In *AAAI*, pages 665–670, 2008.

[113] Hanan G Ayad and Mohamed S Kamel. On voting-based consensus of cluster ensembles. *Pattern Recognition*, 43(5):1943–1953, 2010.

[114] Sandro Vega-Pons, Jyrko Correa-Morris, and José Ruiz-Shulcloper. Weighted partition consensus via kernels. *Pattern Recognition*, 43(8):2712–2724, 2010.

[115] Hye-Sung Yoon, Sun-Young Ahn, Sang-Ho Lee, Sung-Bum Cho, and Ju Han Kim. Heterogeneous clustering ensemble method for combining different cluster results. In *International Workshop on Data Mining for Biomedical Applications*, pages 82–92. Springer, 2006.

[116] Alexander Topchy, Anil K Jain, and William Punch. Combining multiple weak clusterings. In *Third IEEE International Conference on Data Mining*, pages 331–338. IEEE, 2003.

[117] Alexander Topchy, Anil K Jain, and William Punch. A mixture model for clustering ensembles. In *Proceedings of the 2004 SIAM international conference on data mining*, pages 379–390. SIAM, 2004.

[118] Sihong Xie, Jing Gao, Wei Fan, Deepak Turaga, and Philip S Yu. Class-distribution regularized consensus maximization for alleviating overfitting in model combination. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 303–312. ACM, 2014.

[119] Hongfu Liu, Zhiqiang Tao, and Zhengming Ding. Consensus clustering: An embedding perspective, extension and beyond. *arXiv preprint arXiv:1906.00120*, 2019.

[120] Dong Huang, Jian-Huang Lai, and Chang-Dong Wang. Robust ensemble clustering using probability trajectories. *IEEE transactions on knowledge and data engineering*, 28(5):1312–1326, 2015.

[121] Boris Mirkin. Reinterpreting the category utility function. *Machine Learning*, 45(2):219–228, 2001.

[122] Musa Mojarad, Samad Nejatian, Hamid Parvin, and Majid Mohammadpoor. A fuzzy clustering ensemble based on cluster clustering and iterative fusion of base clusters. *Applied Intelligence*, 49(7):2567–2581, 2019.

[123] Mohamed Ali Zoghlami, Minyar Sassi Hidri, and Rahma Ben Ayed. A merging-based consensus-driven fuzzy clustering of distributed data. In *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6. IEEE, 2015.

[124] Witold Pedrycz and Kaoru Hirota. A consensus-driven fuzzy clustering. *Pattern Recognition Letters*, 29(9):1333–1343, 2008.

[125] Minyar Sassi Hidri, Mohamed Ali Zoghlami, and Rahma Ben Ayed. Speeding up the large-scale consensus fuzzy clustering for handling big data. *Fuzzy Sets and Systems*, 348:50–74, 2018.

[126] Vernon Turner, John F Gantz, David Reinsel, and Stephen Minton. The digital universe of opportunities: Rich data and the increasing value of the internet of things. *IDC Analyze the Future*, 16:13–19, 2014.

[127] William Yu Chung Wang and Yichuan Wang. Analytics in the era of big data: the digital transformations and value creation in industrial marketing, 2020.

[128] Apache Hadoop. Apache hadoop. *URL http://hadoop. apache. org*, 2011.

[129] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[130] Peter Mika. Flink: Semantic web technology for the extraction and analysis of social networks. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):211–223, 2005.

[131] A. M. Team. Apache mahout: Scalable machine-learning and data-mining library, 2011.

[132] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D Ernst. Haloop: efficient iterative data processing on large clusters. *Proceedings of the VLDB Endowment*, 3(1-2):285–296, 2010.

[133] Jielong Xu, Zhenhua Chen, Jian Tang, and Sen Su. T-storm: traffic-aware online scheduling in storm. In *Proc. of IEEE 34th International Conference on Distributed Computing Systems*, pages 535–544. IEEE, Madrid, Spain, June 30-July 3, 2014.

[134] Thomas A Runkler and Helmut Krcmar. Stream processing on demand for lambda architectures. In *Proc. of 12th European Workshop on Computer Performance Engineering, Computer Performance Engineering*, volume 9272 of *Lecture Notes in Computer Science*, pages 243–257. 2015.

[135] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient

distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proc. of 9th USENIX Conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, San Jose, CA, April, 2012.

[136] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. *HotCloud*, 10:10–10, 2010.

[137] Apache Spark. Apache spark. *Retrieved January*, 17:2018, 2018.

[138] Jorge Veiga, Roberto R Expósito, Xoán C Pardo, Guillermo L Taboada, and Juan Tourifio. Performance evaluation of big data frameworks for large-scale data analytics. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 424–431. IEEE, 2016.

[139] Ren Li, Haibo Hu, Heng Li, Yunsong Wu, and Jianxi Yang. Mapreduce parallel programming model: a state-of-the-art survey. *International Journal of Parallel Programming*, 44(4):832–866, 2016.

[140] Dhruba Borthakur et al. Hdfs architecture guide. *Hadoop Apache Project*, 53(1-13):2, 2008.

[141] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. Big data analytics on apache spark. *International Journal of Data Science and Analytics*, 1(3-4):145–164, 2016.

[142] Yanfeng Zhang, Shimin Chen, Qiang Wang, and Ge Yu. i^2mapreduce: Incremental mapreduce for mining evolving big data. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1906–1919, 2015.

[143] Yanfeng Zhang, Shimin Cheen, Qiang Wang, and Ge Yu. i^2mapreduce: Incremental mapreduce for mining evolving big data. *arXiv preprint arXiv:1501.04854*.

[144] Behrooz Hosseini and Kourosh Kiani. A big data driven distributed density based hesitant fuzzy clustering using apache spark with application to gene expression microarray. *Engineering Applications of Artificial Intelligence*, 79:100–113, 2019.

[145] Minyar Sassi Hidri, Mohamed Ali Zoghlami, and Rahma Ben Ayed. Speeding up the large-scale consensus fuzzy clustering for handling big data. *Fuzzy Sets and Systems*, 348:50–74, 2018.

[146] Terence Kwok, Kate Smith, Sebastian Lozano, and David Taniar. Parallel fuzzy c-means clustering for large data sets. In *Proc. of European Conference on Parallel Processing, Euro-Par 2002 Parallel Processing*, Lecture Notes in Computer Science, pages 365–374. Springer Berlin Heidelberg, 2002.

[147] Jurgen Beringer and Eyke Hullermeier. Fuzzy clustering of parallel data streams. *Advances in Fuzzy Clustering and Its Application*, pages 333–352, 2007.

[148] Weizhong Zhao, Huifang Ma, and Qing He. Parallel k-means clustering based on mapreduce. In *Proc. of IEEE International Conference on Cloud Computing, Cloud Computing*, volume 5931 of *Lecture Notes in Computer Science*, pages 674–679. Springer Berlin Heidelberg, 2009.

[149] Raghavendra K Chunduri and Aswani Kumar Cherukuri. Scalable formal concept analysis algorithms for large datasets using spark. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–21, 2018.

[150] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih. Big data technologies: A survey. *Journal of King Saud University-Computer and Information Sciences*, 30(4):431–448, 2018.

[151] Neha Bharill, Aruna Tiwari, and Aayushi Malviya. Fuzzy based scalable clustering algorithms for handling big data using apache spark. *IEEE Transactions on Big Data*, 2(4):339–352, 2016.

[152] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[153] Timothy C Havens, James C Bezdek, and Marimuthu Palaniswami. Incremental kernel fuzzy c-means. In *Computational Intelligence*, pages 3–18. Springer, 2012.

[154] Neha Bharill and Aruna Tiwari. Handling big data with fuzzy based classification approach. In *Advance Trends in Soft Computing*, pages 219–227. Springer, 2014.

[155] Nicholas M Luscombe, Dov Greenbaum, Mark Gerstein, et al. What is bioinformatics? a proposed definition and overview of the field. *Methods of Information in Medicine*, 40(4):346–358, 2001.

[156] José A Castellanos-GarzóN, Carlos Armando GarcíA, Paulo Novais, and Fernando DíAz. A visual analytics framework for cluster analysis of dna microarray data. *Expert Systems with Applications*, 40(2):758–774, 2013.

[157] Ka-Chun Wong. *Computational biology and bioinformatics: Gene regulation*. CRC Press, 2016.

[158] Grainne Kerr, Heather J Ruskin, Martin Crane, and Padraig Doolan. Techniques for clustering gene expression data. *Computers in biology and medicine*, 38(3):283–293, 2008.

[159] Xiuwen Zheng, David Levine, Jess Shen, Stephanie M Gogarten, Cathy Laurie, and Bruce S Weir. A high-performance computing toolset for relatedness and principal component analysis of snp data. *Bioinformatics*, 28(24):3326–3328, 2012.

[160] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. Recent advances and emerging challenges of feature selection in the context of big data. *Knowledge-Based Systems*, 86:33–45, 2015.

[161] Guoguang Zhao, Cheng Ling, and Donghong Sun. Sparksw: scalable distributed computing system for large-scale biological sequence alignment. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 845–852. IEEE, 2015.

[162] Yanfeng Zhang, Shimin Chen, Qiang Wang, and Ge Yu. i ̂2 mapreduce: Incremental mapreduce for mining evolving big data. *IEEE transactions on knowledge and data engineering*, 27(7):1906–1919, 2015.

[163] Shanjiang Tang, Bingsheng He, Ce Yu, Yusen Li, and Kun Li. A survey on spark ecosystem for big data processing. *arXiv preprint arXiv:1811.08834*, 2018.

[164] Runxin Guo, Yi Zhao, Quan Zou, Xiaodong Fang, and Shaoliang Peng. Bioinformatics applications on apache spark. *GigaScience*, 7(8):giy098, 2018.

[165] Swati Vipsita and Santanu Ku Rath. Two-stage approach for protein superfamily classification. *Computational Biology Journal*, 2013, 2013.

[166] Nickolai Alexandrov, Shuaishuai Tai, Wensheng Wang, Locedie Mansueto, Kevin Palis, Roven Rommel Fuentes, Victor Jun Ulat, Dmytro Chebotarov, Gengyun Zhang, Zhikang Li, et al. Snp-seek database of snps derived from 3000 rice genomes. *Nucleic acids research*, 43(D1):D1023–D1027, 2015.

[167] Tae-Ho Lee, Hui Guo, Xiyin Wang, Changsoo Kim, and Andrew H Paterson. Snphylo: a pipeline to construct a phylogenetic tree from huge snp data. *BMC genomics*, 15(1):162, 2014.

[168] Ann-Christine Syvänen. Toward genome-wide snp genotyping. *Nature genetics*, 37(6):S5–S10, 2005.

[169] Robert J Henry. *Plant genotyping II: SNP technology*. CABI, 2008.

[170] Libin Liu, Yee-kin Ho, and Stephen Yau. Clustering dna sequences by feature vectors. *Molecular phylogenetics and evolution*, 41(1):64–69, 2006.

[171] Kohbalan Moorthy, Mohd Saberi Mohamad, and Safaai Deris. A review on missing value imputation algorithms for microarray gene expression data. *Current Bioinformatics*, 9(1):18–22, 2014.

[172] Nomin Batnyam, Ariundelger Gantulga, and Sejong Oh. An efficient classification for single nucleotide polymorphism (snp) dataset. In *Computer and Information Science*, pages 171–185. Springer, 2013.

[173] Qijian Song, David L Hyten, Gaofeng Jia, Charles V Quigley, Edward W Fickus, Randall L Nelson, and Perry B Cregan. Fingerprinting soybean germplasm and its utility in genomic research. *G3: Genes, Genomes, Genetics*, 5(10):1999–2006, 2015.

[174] Cathy Wu, George Whitson, Jerry McLarty, Adisorn Ermongkonchai, and Tzu Chung Chang. Protein classification artificial neural system. *Protein Science*, 1(5):667–677, 1992.

[175] Limin Fu and Enzo Medico. Flame, a novel fuzzy clustering method for the analysis of dna microarray data. *BMC bioinformatics*, 8(1):3, 2007.

[176] Mihail Popescu, James C Bezdek, and James M Keller. eccv: A new fuzzy cluster validity measure for large relational bioinformatics datasets. In *2009 IEEE International Conference on Fuzzy Systems*, pages 1003–1008. IEEE, 2009.

[177] Zhong-dong Wu, Wei-xin Xie, and Jian-ping Yu. Fuzzy c-means clustering algorithm based on kernel method. In *Proceedings Fifth International Conference on Computational Intelligence and Multimedia Applications. ICCIMA 2003*, pages 49–54. IEEE, 2003.

[178] Tae-Ho Lee, Hui Guo, Xiyin Wang, Changsoo Kim, and Andrew H Paterson. Snphylo: a pipeline to construct a phylogenetic tree from huge snp data. *BMC genomics*, 15(1):162, 2014.

[179] Alessandro G Di Nuovo and Vincenzo Catania. An evolutionary fuzzy c-means approach for clustering of bio-informatics databases. In *2008 IEEE Interna-*

tional Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence), pages 2077–2082. IEEE, 2008.

[180] Clare M O'Connor, Jill U Adams, and Jennifer Fairman. Essentials of cell biology. *Cambridge, MA: NPG Education*, 1:54, 2010.

[181] Eghbal G Mansoori, Mansoor J Zolghadri, and Seraj D Katebi. Protein superfamily classification using fuzzy rule-based classifier. *IEEE Transactions on NanoBioscience*, 8(1):92–99, 2009.

[182] Jason Tsong Li Wang, Qicheng Ma, Dennis Shasha, and Cathy H. Wu. New techniques for extracting features from protein sequences. *IBM Systems Journal*, 40(2):426–441, 2001.

[183] MO Dayhoff, RM Schwartz, and BC Orcutt. 22 a model of evolutionary change in proteins. In *Atlas of Protein Sequence and Structure*, volume 5, pages 345–352. National Biomedical Research Foundation Silver Spring, MD, 1978.

[184] Sanghamitra Bandyopadhyay. An efficient technique for superfamily classification of amino acid sequences: feature extraction, fuzzy clustering and prototype selection. *Fuzzy Sets and Systems*, 152(1):5–16, 2005.

[185] Eghbal G Mansoori, Mansoori J Zolghadri, Seraj D Katebi, Hassan Mohabatkar, Reza Boostani, and Mohammad H Sadreddini. Generating fuzzy rules for protein classification. *Iranian Journal of Fuzzy Systems*, 5(2):21–33, 2008.

[186] Neha Bharill, Aruna Tiwari, and Anshul Rawat. A novel technique of feature extraction with dual similarity measures for protein sequence classification. *Procedia Computer Science*, 48:795–801, 2015.

[187] Kuo-Chen Chou. Some remarks on protein attribute prediction and pseudo amino acid composition. *Journal of theoretical biology*, 273(1):236–247, 2011.

[188] Kuo-Chen Chou. Using amphiphilic pseudo amino acid composition to predict enzyme subfamily classes. *Bioinformatics*, 21(1):10–19, 2005.

[189] MK Gupta, R Niyogi, and M Misra. An alignment-free method to find similarity among protein sequences via the general form of chou's pseudo amino acid composition. *SAR and QSAR in Environmental Research*, 24(7):597–609, 2013.

[190] Martin Steinegger and Johannes Söding. Clustering huge protein sequence sets in linear time. *Nature communications*, 9(1):1–8, 2018.

[191] Karen F Han and David Baker. Recurring local sequence motifs in proteins. *Journal of molecular biology*, 251(1):176–187, 1995.

[192] Christopher Bystroff, Vesteinn Thorsson, and David Baker. Hmmstr: a hidden markov model for local sequence-structure correlations in proteins. *Journal of molecular biology*, 301(1):173–190, 2000.

[193] Hui Xiong, Junjie Wu, and Jian Chen. K-means clustering versus validation measures: a data-distribution perspective. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):318–331, 2008.

[194] Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300. ACM, 2004.

[195] Chun-Ting Zhang, Kuo-Chen Chou, and GM Maggiora. Predicting protein structural classes from amino acid composition: application of fuzzy clustering. *Protein Engineering, Design and Selection*, 8(5):425–435, 1995.

[196] Tao Lu, Yongchao Dou, and Chi Zhang. Fuzzy clustering of cpp family in plants with evolution and interaction analyses. *BMC bioinformatics*, 14(S13):S10, 2013.

[197] Erfan Farhangi, Nasser Ghadiri, Mahsa Asadi, Mohammad Amin Nikbakht, and Sylvain Pitre. Fast and scalable protein motif sequence clustering based on hadoop framework. In *2017 3th International Conference on Web Research (ICWR)*, pages 24–31. IEEE, 2017.

[198] Quang-Thinh Bui, Bay Vo, Vaclav Snasel, Witold Pedrycz, Tzung-Pei Hong, Ngoc-Thanh Nguyen, and Mu-Yen Chen. Sfcm: A fuzzy clustering algorithm of extracting the shape information of data. *IEEE Transactions on Fuzzy Systems*, 2020.

[199] Eréndira Rendón, Itzel Abundez, Alejandra Arizmendi, and Elvia M Quiroz. Internal versus external cluster validation indexes. *International Journal of computers and communications*, 5(1):27–34, 2011.

[200] Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617, 2002.

[201] Ka Yee Yeung and Walter 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. *Bioinformatics*, 17(9):763–774, 2001.

[202] Nadia Bolshakova and Francisco Azuaje. Cluster validation techniques for genome expression data. *Signal processing*, 83(4):825–833, 2003.

[203] Guilherme P Coelho, Celso C Barbante, Levy Boccato, Romis RF Attux, José R Oliveira, and Fernando J Von Zuben. Automatic feature selection for bci: an analysis using the davies-bouldin index and extreme learning machines. In *The 2012 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2012.

[204] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures. In *2010 IEEE international conference on data mining*, pages 911–916. IEEE, 2010.

[205] José María Luna-Romera, María Martínez-Ballesteros, Jorge García-Gutiérrez, and José C Riquelme. External clustering validity index based on chi-squared statistical test. *Information Sciences*, 487:1–17, 2019.

[206] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[207] Verena Zuber, A Pedro Duarte Silva, and Korbinian Strimmer. A novel algorithm for simultaneous snp selection in high-dimensional genome-wide association studies. *BMC bioinformatics*, 13(1):1–8, 2012.

[208] Jiming Jiang, Cong Li, Debashis Paul, Can Yang, Hongyu Zhao, et al. On high-dimensional misspecified mixed model analysis in genome-wide association study. *The Annals of Statistics*, 44(5):2127–2160, 2016.

[209] Mingshun Yuan, Zijiang Yang, Guangzao Huang, and Guoli Ji. Feature selection by maximizing correlation information for integrated high-dimensional protein data. *Pattern Recognition Letters*, 92:17–24, 2017.

[210] Chandra Shekhar Pareek, Rafal Smoczynski, and Andrzej Tretyn. Sequencing technologies and genome sequencing. *Journal of applied genetics*, 52(4):413–435, 2011.

[211] W Lathe, J Williams, M Mangan, and D Karolchik. Genomic data resources: challenges and promises. *Nature Education*, 1(3):2, 2008.

[212] David Grant, Marce Imsande, and Randy Shoemaker. Soybase, a soybean genome database. Technical report, 1996.

[213] Tanya Barrett, Dennis B Troup, Stephen E Wilhite, Pierre Ledoux, Dmitry Rudnev, Carlos Evangelista, Irene F Kim, Alexandra Soboleva, Maxim Tomashevsky, and Ron Edgar. Ncbi geo: mining tens of millions of expression profiles—database and tools update. *Nucleic acids research*, 35(suppl_1):D760–D765, 2007.

[214] UniProt Consortium. Uniprot: a hub for protein information. *Nucleic acids research*, 43(D1):D204–D212, 2015.

[215] Helen M Berman, John Westbrook, Zukang Feng, Gary Gilliland, Talapady N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. The protein data bank. *Nucleic acids research*, 28(1):235–242, 2000.

[216] Howard S Bilofsky and Burks Christian. The genbank® genetic sequence data bank. *Nucleic acids research*, 16(5):1861–1863, 1988.

[217] Nayanah Siva. 1000 genomes project, 2008.

[218] Michael Y Galperin. The molecular biology database collection: 2008 update. *Nucleic acids research*, 36(suppl_1):D2–D4, 2008.

[219] Eric W Sayers, Richa Agarwala, Evan E Bolton, J Rodney Brister, Kathi Canese, Karen Clark, Ryan Connor, Nicolas Fiorini, Kathryn Funk, Timothy Hefferon, et al. Database resources of the national center for biotechnology information. *Nucleic acids research*, 47(Database issue):D23, 2019.

[220] Stephen T Sherry, M-H Ward, M Kholodov, J Baker, Lon Phan, Elizabeth M Smigielski, and Karl Sirotkin. dbsnp: the ncbi database of genetic variation. *Nucleic acids research*, 29(1):308–311, 2001.

[221] Chuming Chen, Hongzhan Huang, and Cathy H Wu. Protein bioinformatics databases and resources. *Protein Bioinformatics*, pages 3–39, 2017.

[222] Qijian Song, David L Hyten, Gaofeng Jia, Charles V Quigley, Edward W Fickus, Randall L Nelson, and Perry B Cregan. Development and evaluation of soysnp50k, a high-density genotyping array for soybean. *PloS one*, 8(1):e54985, 2013.

[223] Nonoy Bandillo, Chitra Raghavan, Pauline Andrea Muyco, Ma Anna Lynn Sevilla, Irish T Lobina, Christine Jade Dilla-Ermita, Chih-Wei Tung, Susan McCouch, Michael Thomson, Ramil Mauleon, et al. Multi-parent advanced generation inter-cross (magic) populations in rice: progress and potential for genetics research and breeding. *Rice*, 6(1):11, 2013.

[224] Philip Traldi Wysmierski and Natal Antonio Vello. The genetic base of brazilian soybean cultivars: evolution over time and breeding implications. *Genetics and molecular Biology*, 36(4):547–555, 2013.

[225] Eric J Sedivy, Faqiang Wu, and Yoshie Hanzawa. Soybean domestication: the origin, genetic architecture and molecular bases. *New Phytologist*, 214(2):539–553, 2017.

[226] Jeong-Dong Lee, J Grover Shannon, Tri D Vuong, and Henry T Nguyen. Inheritance of salt tolerance in wild soybean (glycine soja sieb. and zucc.) accession pi483463. *Journal of Heredity*, 100(6):798–801, 2009.

[227] Min Xie, Claire Yik-Lok Chung, Man-Wah Li, Fuk-Ling Wong, Xin Wang, Ailin Liu, Zhili Wang, Alden King-Yung Leung, Tin-Hang Wong, Suk-Wah Tong, et al. A reference-grade wild soybean genome. *Nature communications*, 10(1):1–12, 2019.

[228] Fan Wu, Su Zhao, Bin Yu, Yan-Mei Chen, Wen Wang, Zhi-Gang Song, Yi Hu, Zhao-Wu Tao, Jun-Hua Tian, Yuan-Yuan Pei, et al. A new coronavirus associated with human respiratory disease in china. *Nature*, 579(7798):265–269, 2020.

[229] Chaolin Huang, Yeming Wang, Xingwang Li, Lili Ren, Jianping Zhao, Yi Hu, Li Zhang, Guohui Fan, Jiuyang Xu, Xiaoying Gu, et al. Clinical features of patients infected with 2019 novel coronavirus in wuhan, china. *The lancet*, 395(10223):497–506, 2020.

[230] A Mishal, R Saravanan, S Sakthi Atchitha, K Santhiya, M Rithika, S Sanju Menaka, and T Thiruvalluvan. A review of corona virus disease-2019. *History*, 4:07, 2020.

[231] Zi Yue Zu, Meng Di Jiang, Peng Peng Xu, Wen Chen, Qian Qian Ni, Guang Ming Lu, and Long Jiang Zhang. Coronavirus disease 2019 (covid-19): a perspective from china. *Radiology*, 296(2):E15–E25, 2020.

[232] Heng Li, Shang-Ming Liu, Xiao-Hua Yu, Shi-Lin Tang, and Chao-Ke Tang. Coronavirus disease 2019 (covid-19): current status and future perspectives. *International journal of antimicrobial agents*, 55(5):105951, 2020.

[233] John F Kolen and Tim Hutcheson. Reducing the time complexity of the fuzzy c-means algorithm. *IEEE Transactions on Fuzzy Systems*, 10(2):263–267, 2002.

[234] Moshe Lichman et al. Uci machine learning repository, 2013.

[235] Gereon Frahling and Christian Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 209–217. ACM, 2005.

[236] Gaëlle Loosli, Stéphane Canu, and Léon Bottou. Training invariant support vector machines using selective sampling. *Large scale kernel machines*, 2, 2007.

[237] P. Fränti, O. Virmajoki, and V. Hautamäki. Fast agglomerative clustering using a k-nearest neighbor graph. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28(11):1875–1881, 2006.

[238] Yangtao Wang, Lihui Chen, and Jian Ping Mei. Incremental fuzzy clustering with multiple medoids for large data. *IEEE Transactions on Fuzzy Systems*, 22(6):1557–1568, 2014.

[239] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proc. of 4th Annual Symposium on Cloud Computing*, page 5. ACM, Santa Clara, CA, USA, October, 2013.

[240] Enrique H Ruspini, James C Bezdek, and James M Keller. Fuzzy clustering: A historical perspective. *IEEE Computational Intelligence Magazine*, 14(1):45–55, 2019.

[241] Neha Bharill, Aruna Tiwari, and Aayushi Malviya. Fuzzy based scalable clustering algorithms for handling big data using apache spark. *IEEE Transactions on Big Data*, 2(4):339–352, 2016.

[242] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[243] Olvi L Mangasarian and William H Wolberg. Cancer diagnosis via linear programming. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1990.

[244] P. Fränti R. Mariescu-Istodor and C. Zhong. Xnn graph. LNCS 10029:207–217, 2016.

[245] Veit Schwämmle and Ole Nørregaard Jensen. A simple and fast method to determine the parameters for fuzzy c–means cluster analysis. *Bioinformatics*, 26(22):2841–2848, 2010.

[246] Hon-Ming Lam, Xun Xu, Xin Liu, Wenbin Chen, Guohua Yang, Fuk-Ling Wong, Man-Wah Li, Weiming He, Nan Qin, Bo Wang, et al. Resequencing of 31 wild and cultivated soybean genomes identifies patterns of genetic diversity and selection. *Nature genetics*, 42(12):1053, 2010.

[247] Andrea Remuzzi and Giuseppe Remuzzi. Covid-19 and italy: what next? *The lancet*, 395(10231):1225–1228, 2020.

[248] Carlos Padilla-Rojas, Priscila Lope-Pari, Karolyn Vega-Chozo, Johanna Balbuena-Torres, Omar Caceres-Rey, Henri Bailon-Calderon, Maribel Huaringa-Nuñez, and Nancy Rojas-Serrano. Near-complete genome sequence of a 2019 novel coronavirus (sars-cov-2) strain causing a covid-19 case in perú. *Microbiology resource announcements*, 9(19), 2020.

[249] Marco Ciotti, Silvia Angeletti, Marilena Minieri, Marta Giovannetti, Domenico Benvenuto, Stefano Pascarella, Caterina Sagnelli, Martina Bianchi, Sergio Bernardini, and Massimo Ciccozzi. Covid-19 outbreak: an overview. *Chemotherapy*, 64(5-6):215–223, 2019.

[250] Siham Tabik, Anabel Gómez-Ríos, José Luis Martín-Rodríguez, Iván Sevillano-García, Manuel Rey-Area, David Charte, Emilio Guirado, Juan Luis Suárez, Julián Luengo, MA Valero-González, et al. Covidgr dataset and covid-sdnet methodology for predicting covid-19 based on chest x-ray images. *IEEE journal of biomedical and health informatics*, 24(12):3595–3605, 2020.

[251] AS Albahri, Rula A Hamid, Jwan K Alwan, ZT Al-Qays, AA Zaidan, BB Zaidan, AOS Albahri, AH AlAmoodi, Jamal Mawlood Khlaf, EM Almahdi, et al. Role of biological data mining and machine learning techniques in detecting and diagnosing the novel coronavirus (covid-19): a systematic review. *Journal of medical systems*, 44:1–11, 2020.

[252] Antje Krause, Jens Stoye, and Martin Vingron. Large scale hierarchical clustering of protein sequences. *BMC bioinformatics*, 6(1):15, 2005.

[253] Thanh Thi Nguyen. Artificial intelligence in the battle against coronavirus (covid-19): a survey and future research directions. *arXiv preprint arXiv:2008.07343*, 2020.

[254] Mohammad Reza Mahmoudi, Dumitru Baleanu, Zulkefli Mansor, Bui Anh Tuan, and Kim-Hung Pho. Fuzzy clustering method to compare the spread rate of covid-19 in the high risks countries. *Chaos, Solitons & Fractals*, 140:110230, 2020.

[255] Aymeric Silvin, Nicolas Chapuis, Garett Dunsmore, Anne-Gaëlle Goubet, Agathe Dubuisson, Lisa Derosa, Carole Almire, Clémence Hénon, Olivier Kosmider, Nathalie Droin, et al. Elevated calprotectin and abnormal myeloid cell subsets discriminate severe from mild covid-19. *Cell*, 182(6):1401–1418, 2020.

[256] Untari N Wisesty and Tati Rajab Mengko. Comparison of dimensionality reduction and clustering methods for sars-cov-2 genome. *Bulletin of Electrical Engineering and Informatics*, 10(4):2170–2180, 2021.

[257] Halat Ahmed Hussein and Adnan Mohsin Abdulazeez. Covid-19 pandemic datasets based on machine learning clustering algorithms: A review. *PalArch's Journal of Archaeology of Egypt/Egyptology*, 18(4):2672–2700, 2021.

[258] Bo Wang and Lin Jiang. Principal component analysis applications in covid-19 genome sequence studies. *Cognitive computation*, pages 1–12, 2021.

[259] Yiliao Song, Jie Lu, Haiyan Lu, and Guangquan Zhang. Fuzzy clustering-based adaptive regression for drifting data streams. *IEEE Transactions on Fuzzy Systems*, 28(3):544–557, 2019.

[260] Anjin Liu, Jie Lu, and Guangquan Zhang. Concept drift detection via equal intensity k-means space partitioning. *IEEE transactions on cybernetics*, 2020.

[261] Anjin Liu, Jie Lu, and Guangquan Zhang. Concept drift detection: Dealing with missing values via fuzzy distance estimations. *IEEE Transactions on Fuzzy Systems*, 2020.

[262] Qiying Feng, Long Chen, CL Philip Chen, and Li Guo. Deep fuzzy clustering—a representation learning approach. *IEEE Transactions on Fuzzy Systems*, 28(7):1420–1433, 2020.