B.TECH. PROJECT REPORT

On

GPU-accelerated Scalable Feature Extraction Techniques with Scalable Kernelized Fuzzy Clustering Algorithms and its Application to Real-life Genomics Data for Gene Identification.

BY

Namani Sreeharsh, 180001032
&
Saloni Sawarkar, 180001048

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE

May, 2022

GPU-accelerated Scalable Feature Extraction Techniques with Scalable Kernelized Fuzzy Clustering Algorithms and its Application to Real-life Genomics Data for Gene Identification.

PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degrees

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Namani Sreeharsh, 180001032
&
Saloni Sawarkar, 180001048,

Discipline of Computer Science and Engineering,

Indian Institute of Technology, Indore

Guided by:

Prof. Aruna Tiwari,

Professor,

Computer Science and Engineering,

IIT Indore

CANDIDATES' DECLARATION

We hereby declare that the project entitled **"GPU-accelerated Scalable Feature Extraction Techniques with Scalable Kernelized Fuzzy Clustering Algorithms and its Application to Real-life Genomics Data for Gene Identification."** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in 'Computer Science and Engineering' completed under the supervision of **Prof. Aruna Tiwari, Professor, Computer Science and Engineering**, IIT Indore is an authentic work.

Further, we declare that we have not submitted this work for the award of any other degree elsewhere.

26/05/2022
Namani Sreeharsh

26/05/22
Saloni Sawarkar

CERTIFICATE by BTP Guide

It is certified that the above statement made by the student is correct to the best of my knowledge.

26/5/2022
Prof. Aruna Tiwari,
Professor,
Discipline of Computer Science and Engineering,
IIT Indore

PREFACE

This report on "GPU-accelerated Scalable Feature Extraction Techniques with Scalable Kernelized Fuzzy Clustering Algorithms and its Application to Real-life Genomics Data for Gene Identification." is prepared under the guidance of Prof. Aruna Tiwari, Professor, Computer Science and Engineering, IIT Indore.

Through this report, we have tried to provide a detailed description of our approach, design, and implementation of an innovative method to perform Analysis of Scalable Kernelized fuzzy Clustering for Genome Data. We tried to analyze the datasets and perform testing through various measures and performing gene identification of real-life genome data.

ACKNOWLEDGEMENTS

We would like to express our gratitude to **Prof. Aruna Tiwari**, our B.Tech Project Supervisor, for her direction and ongoing assistance in organising the project and offering helpful input throughout the project. We appreciate her time and work.

This project would not have been achieved without the help of **Dr. Preeti Jha**(Research Associate, IIT Indore). She gave us invaluable advice on how to deal with the project's complexities and showed us how to produce a scientific article. **Dr. Milind Ratnaparkhe** (Senior Scientist at ICAR-Indian Institute of Soybean Research Indore) provided genomic data, which we gratefully acknowledge. **Dr. Neha Bharill**(Assistant Professor, Mahindra University) has been of great assistance, support, and constructive comments to us.

We are grateful to the Institute for giving us with the chance to learn about systems research, particularly **Prof. Aruna Tiwari's** Lab, which provided us with the essential hardware utilities to finish the project. I would like to express my gratitude to the National Supercomputing Mission, HPC Applications Development Funded Research Project by SERC, IISC Bangalore in collaboration with the Ministry of Electronics and Information Technology (MeiTY), Government of India, for allowing access to supercomputers from IIT Kharagpur and CDAC-Pune (PARAM SHAKTI and PARAM SIDDHI) respectively.

Finally, we want to express our gratitude to everyone who assisted us in completing this project, especially those whose names we may have overlooked.

26/05/2022 26/05/22

Namani Sreeharsh & Saloni Sawarkar,
B.Tech. 4^{th} **Year**
Discipline of Computer Science and Engineering,
IIT Indore

ABSTRACT

Bioinformatics is the study of gaining knowledge from biological data. It encompasses data collection, storage, retrieval, manipulation, modelling, and prediction using algorithms and software. When it comes to genomics, the role of technology is primarily focused on the tremendous rise in genome sequencing, which is developing at a rate that is faster than projected by Moore's law. The size of the data set is increasing exponentially, necessitating the use of massive data processing technology. 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.

This thesis mainly investigates to design and develop the fuzzy based scalable kernelized clustering algorithms and feature extraction techniques for handling huge soybean RNA and SNP data using Apache Spark cluster on High Performance Supercomputing (HPC). To handle Big Data, we proposed an Apache Spark cluster-based Log kernelized clustering algorithm named Kernelized Scalable Random Sampling with Iterative Optimization Fuzzy c-Means (LKSRSIO-FCM). This is based on the Log Kernelized Scalable Literal Fuzzy c-Means (LKSLFCM) clustering algorithm, in which log kernel function is used. Additionally, we proposed an Apache Spark cluster-based Cauchy kernelized clustering algorithm named Cauchy Kernelized Scalable Random Sampling with Iterative Optimization Fuzzy c-Means (CKSRSIO-FCM). This is based on the Cauchy Kernelized Scalable Literal Fuzzy c-Means (CKSLFCM) clustering algorithm, in which cauchy kernel function is used.This proposed work is inspired by a Kernelized Scalable Random Sampling with Iterative Optimization Fuzzy c-Means (KSRSIO-FCM) algorithm. The kernel function is applied to achieve better mapping for non-linearly separable datasets. 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. The effectiveness of the proposed scalable kernelized fuzzy clustering algorithms are tested on large benchmark datasets.

To handle huge real-life soybean SNP sequences, we have proposed novel scalable feature extraction techniques for preprocessing huge SNP/RNA data that extract fixed-length numerical feature vectors. The extracted numerical feature vectors are then fed as an input to the proposed scalable kernelized fuzzy clustering algorithms to cluster huge real-life SNP datasets. The algorithms are intended for detecting disease by grouping samples (individuals) with comparable gene expression patterns, as well as identifying groupings of genes with similar profiles across samples. However, few practical research have been undertaken to test the efficacy of suggested scalable kernelized fuzzy clustering algorithms for issues aimed at identifying new illness utilising gene identification. For the SoySNP50K iSelect BeadChip, we have created a new version of the SNP dataset. The complete data set for 20,087 *G. max* and *G. soja* accessions genotyped with 42,509 SNPs is generated for **Wm82.a3**.

Contents

List of Tables

List of Figures

List of Abbreviations

KSRSIO-FCM	**K**ernelized **S**calable **R**andom **S**ampling with **I**terative **O**ptimization **F**uzzy **C**-Means
CKSRSIO-FCM	**C**auchy **K**ernelized **S**calable **R**andom **S**ampling with **I**terative **O**ptimization **F**uzzy **C**-Means
LKSRSIO-FCM	**L**ogarithmic **K**ernelized **S**calable **R**andom **S**ampling with **I**terative **O**ptimization **F**uzzy **C**-Means
KSLFCM	**K**ernelized **S**calable **L**iteral **F**uzzy **C**-Means
CKSLFCM	**C**auchy **K**ernelized **S**calable **L**iteral **F**uzzy **C**-Means
LKSLFCM	**L**ogarithmic **K**ernelized **S**calable **L**iteral **F**uzzy **C**-Means
FCM	**F**uzzy **C**-Means
RBF	**R**adial **B**asis **F**unctions
NMI	**N**ormalized **M**utual **I**nformation
ARI	**A**djusted **R**and **I**ndex
HPC	**H**igh **P**erformance **C**omputing
GPU	**G**raphical **P**rocessing **U**nit
GB	**G**iga **B**ytes

Chapter 1

Introduction

1.1 Background

The size of everyday data sets is outpacing the capability of computational hardware to analyze these datasets. The datasphere will rise by 175 zettabytes (ZB) by 2025, according to a report released by IDC on the ever-expanding datasphere [1]. In the field of bioinformatics, data is being generated at an alarming rate as well. Particle physics experiments, search engine logs and indexes, and other big data sources are no longer the exclusive sources of big data. Data volume is increasing everywhere, including in the bioinformatics area, as a result of the digitization of all operations and the availability of low-cost, high-throughput technologies. By 2030, nine out of every ten persons aged six and above would be engaged in some form of digital activity [2]. In today's world, a limitless amount of advanced information is being obtained at a rising rate in a variety of disciplines [3, 4, 5, 6]. Because of the increasing volume of Big Data from various sources, there is a demand for authentic research that necessitates thorough examination in Big Data analytic in order to gain important insights from the rich information included in Big Data. Data is created rapidly in bioinformatics as well. Not only are search engine logs and indexes used as big data sources. Data volume is increasing due to digitization of processes and low-cost high-throughput equipment, notably in bioinformatics. A human genome, for example, is around 200 terabytes in size. To find a new disease bio-marker, biologists no longer rely on traditional laboratories, but on the vast amounts of genomic data generated by numerous research organisations. Automated genome sequencers, clustering of genome sequencing, and other bioinformatics technologies are enabling this new era of big data.

The genome datasets comprise of list of strings and strings contain characters. But, the machine learning algorithms can process only numerals. So we have to convert the dataset into series of numbers and extract meaningful information from this series of numbers so that the resultant feature vector can be used in clustering. This process of extracting features from the list of numbers is very tedious and significant step of the entire pipeline. And selection of appropriate feature extraction method is a subjective issue and each kind of dataset may have to be dealt with differently. If inappropriate feature extraction method is used for extracting feature vectors from the given crude dataset, then the whole process of soft clustering the dataset falls apart. Since the size and complexity of dataset is highly unpredictable, we have to come up with scalable algorithms to extract feature vectors, which can process data sets of huge size with ease. 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. 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 genome data, but the efficiency is poor for massive data [7]. Many researchers used machine learning methods to extract relevant information from various genome datasets [8, 9]. Researchers also addressed this issue by developing scalable feature extraction methods for genome data [10, 11]. 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 RNA/SNP data and clustering of huge RNA data by proposing various scalable fuzzy clustering models. However, the proposed scalable clustering models are general purpose which can be applied to any problem. The dataset is getting bigger, requiring bigger data processing technology. So, to manage huge genomic datasets, improvements are required. A scalable fuzzy clustering methodology for huge RNA/SNP sequences was investigated in this thesis.

Clustering is an unsupervised learning method which is widely considered a data mining strategy to extract important insights from unlabeled data. It tries to transform data into clusters such that the data patterns in a cluster share similarity, which leads to finding out the patterns of the dataset. In fuzzy clustering, a data sample with different membership levels can belong to different clusters. The FCM clustering algorithm uses iterative optimization to minimize an objective function using a measure of feature spatial similarity.

In most cases, FCM is suitable for grouping data with linear data distribution in feature space. For real-world clustering tasks, the input data is typically not easily separable due to the highly complex data structure or when clusters vary in size, density and shape. Kernel functions must be continuous, symmetric, and most preferably should have a positive (semi) definite Gram matrix. Kernels which are said to satisfy Mercer's theorem are positive semi-definite, meaning their kernel matrices have only non-negative Eigenvalues. Choosing the most suitable kernel depends heavily on the problem to be solved and tweaking its parameters can easily become a tedious task [12].

We make use of KSLFCM and KSRSIO-FCM [13] algorithms to propose Analysis of big data by applying different kernel functions on Apache spark framework. We have used three kernels to compare the results. These kernels are Cauchy Kernel, Logarithmic Kernel and Radial Basis Kernel. There exists lot of popular kernels like Fisher Kernel, Graph Kernel, Polynomial Kernel, Radial Basis Function Kernel, Sigmoid Kernel, HyperBolic Tangent Kernel, Cauchy Kernel, Quadratic Kernel, Logarithmic Kernel, Multiquadratic Kernel and several others [12]. From these kernel functions we chose our kernel functions selectively on the basis of these two factors 1) Part of these 3 kernel function equations contains euclidean distance. In analyzing past research work, the researcher demonstrated that the adaptation of kernel functions could improve Euclidean distance measure customary clustering algorithms.Kernel Functions should contain square of equilidean distance for the sake of mathmatical derivation. 2) Kernels that don't give NaN values and give better results. Since the feature vectors size is unpredictable, we have to use scalable algorithms to implement Kernelized Fuzzy c means. Scalable feature extraction algorithms and scalable kernelized Fuzzy c means

algorithms are discussed in Chapter 4 in detail.

1.2 Motivation

This thesis is a study of design and analysis of scalable online fuzzy based clustering algorithms for handling Big Data and feature extraction techniques for huge RNA Data.

Many mathematical models of feature extraction have been investigated in the literature, including Genomic Signal Processing (GSP), DNA Numerical Representation (DNR) [14, 15], and Complex Networks [16]. Since the feature extraction pipeline algorithms usually have high computational costs, scaling the framework to handle real-life Big Data is another challenging issue. In the light of this, we propose the systematic pipeline for feature extraction of RNA sequence and perform analysis of their clustering outcomes. We propose two scalable feature extraction algorithms for massive RNA sequences utilizing the Apache Spark framework on high-performance computing (HPC) infrastructure: a 13-dimensional Scalable Zcurve (13d-SZcurve) and a 13-dimensional Scalable EIIP (13d-SEIIP). Using these approaches, fixed-length numeric feature vectors are extracted from large RNA sequences, and the numeric feature vectors are then passed as input to scalable clustering algorithms.

This thesis investigates Kernelized Scalable Fuzzy c-means algorithm for clustering the genome sequences. We propose Cauchy Kernelized Scalable Random Sampling with Iterative Optimization (CKSRSIO) algorithm which uses Cauchy Kernelized Literal Fuzzy c means algorithm (CKSLFCM). This proposed algorithm uses Cauchy function as a kernel and Apache Spark framework for distributing the data points in the dataset among the worker nodes and parallelizing the parallelizable computations in the CKSRSIO algorithm. Apache Spark and its internal details are explained in detail, in chapter 4. Similarly we also propose Logarithm Kernelized Scalable Random Sampling with Iterative Optimization (LKSRSIO) algorithm, which uses Logarithm Kernelized Scalable Literal Fuzzy c means algorithm (LKSLFCM). This proposed algorithm uses Logarithm function as a kernel and Apache Spark framework for the same purpose as above. We also compare the clustering efficiencies of these 2 algorithms with an already proposed algorithm Kernelized Scalable Random Sampling with Iterative Optimization (KSRSIO) which uses Gaussian radial basis function as a kernel, by running these 3 algorithms on benchmark datasets and use the best kernelized SRSIO algorithm to cluster the genome data. To test the scalable clustering technique on real-life genome data, vast RNA datasets of soybean plant genomes were acquired from ICAR-IISR, Indore for grouping and categorising huge RNA sequences.

1.3 Objectives

The main objectives of this project are:

- To develop a novel scalable feature extraction algorithm for huge real-life SNP/RNA sequences,which extracts 13-dimensional numeric feature vector and validate its superiority.
- GPU accelerated feature extraction techniques for genome sequences on HPC using Apache Spark cluster for improving run time.

- Analysis of kernel mercer functions.
- To develop scalable kernelized fuzzy clustering algorithms, which use the analyzed mercer kernel functions, for handling Big Data by reducing the run-time and optimizing the storage space and validation of their superiority.
- Gene identification of new soybean sequences using the proposed scalable feature extraction and scalable kernelized fuzzy clustering algorithms.

The rest of the report is organized as follows. Chapter 2 Shows Related Work, Chapter 3 briefs about experimental setup, apache Spark and GPU, Chapter 4 describes the proposed methods for both preprocessing and clustering part. The results are shown in Chapter 5 and report concludes with Chapter 6, which describes our planned future work.

Chapter 2

Literature Survey

The following chapter discusses literature pertaining to previously known methods of feature extraction and scalable clustering , providing a short description of the methodology and the dataset used, filtering, and performance measures used for the testing.

2.1 Feature Extraction

This section shows how to extract features for Zcurve and Integer approaches using the Discrete Fourier Transform (DFT) and numerical mappings. It is critical to clarify that we are referring to a biological sequence $s = s[0], s[1], ..., s[N-1]$ such that $s \in \{A, C, G, T\}^N$ [17]. When time domain data is transformed into frequency domain space, it is possible to extract features using the Discrete Fourier Transform (DFT), which is commonly used in digital image and signal processing and can reveal latent periodicities. The DFT of a signal with length N, $x \in R^N$, at frequency k, can be represented as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}} k = 0, 1....N-1. \tag{2.1}$$

In bioinformatics, this approach has been extensively investigated, mostly for analysing periodicities and repetitive elements in DNA sequences and protein structures. Figure 2.1 depicts this method. We'll utilise the Fast Fourier Transform (FFT) to calculate DFT, which is a fast method for computing DFT for a time series. To apply GSP approaches, however, genomic data needs be transformed or mapped using a numeric representation. These representations may be split into three groups, according to Mendizabal Ruiz et al. [18]: single-value mapping, multidimensional sequence mapping, and cumulative sequence mapping. As a result, we investigate the Zcurve numerical mapping approach [19] and Integer single-value mapping.

2.1.1 Zcurve Representation

Zhang [19] developed the Z-curve method as a three-dimensional curve for encoding DNA sequences. In essence, we can check a given sequence $s[n]$ of length N by considering the n^{th} element of the series $(n = 1, 2, ..., N)$. Then, for each base A, C, G, and T, indicates the cumulative occurrence of numbers A_n, C_n, G_n, and T_n as the number of times that base occurred up until $s[1]$ and $s[n]$.

$$A_n + C_n + G_n + T_n = n. \tag{2.2}$$

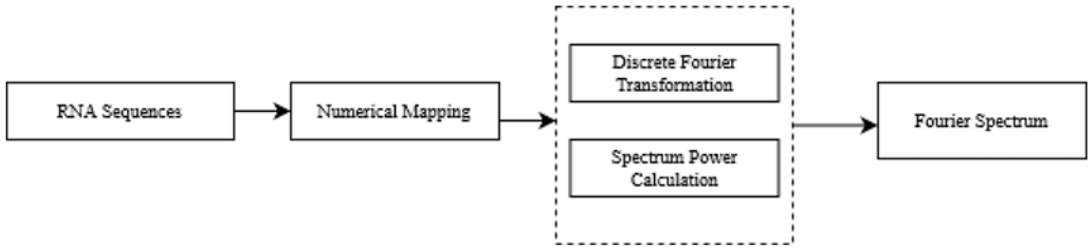

FIGURE 2.1: Fourier Transform and Numerical Mapping Pipeline of Biological Sequence.

As illustrated in Eq.2.3, the Z-curve is composed of a set of nodes designated as $P_1, P_2, ..., P_N$, each of whose coordinates $x[n], y[n]$, and $z[n]$ $(n = 1, 2, ...)$ are uniquely determined by the Z-transform.

$$P[n] = \begin{cases} x[n] &= (A_n + G_n) - (C_n + T_n) \\ y[n] &= (A_n + C_n) - (G_n + T_n) \\ z[n] &= (A_n + T_n) - (C_n + G_n) \end{cases} \tag{2.3}$$

Where, $x[n], y[n], z[n] \in [-n, n]$ $(n = 1, 2, ..., N)$. An entire sequence may be described by the three independent distributions that are represented by the coordinates $x[n], y[n]$, and $z[n]$ [20]. As a result, $x[n]$ = purine/pyrimidine, $y[n]$ = amino/keto, and $z[n]$ = weak hydrogen bonds/strong hydrogen bonds are the three distributions of biological significance [19]. Let s = (G, A, G, A, G, T, G, A, C, C, A), so, x= (1, 2, 3, 4, 5, 4, 5, 6, 5, 4, 5), y = (-1, 0, -1, 0, -1, -2, -3, -2, -1, 0, 1), z = (-1, 0, -1, 0, -1, 0, -1, 0, -1, 2, -1). At each n^{th} index, there is either 1 or -1 difference between the dimensions compared to the previous $(n-1)$ index [19]. Each dimension of an array is updated using the following set of equations, where $x[-1] = y[-1] = z[-1] = 0$, where $n = 1, 2, ...N$.

$$x[n] = \begin{cases} x[n] &= x[n-1] + 1, \quad s[n] = A \quad or \quad G \\ x[n] &= x[n-1] - 1, \quad s[n] = C \quad or \quad T \end{cases} \tag{2.4}$$

$$y[n] = \begin{cases} y[n] &= y[n-1] + 1, \quad s[n] = A \quad or \quad C \\ y[n] &= y[n-1] - 1, \quad s[n] = G \quad or \quad T \end{cases} \tag{2.5}$$

$$z[n] = \begin{cases} z[n] &= z[n-1] + 1, \quad s[n] = A \quad or \quad T \\ z[n] &= z[n-1] - 1, \quad s[n] = G \quad or \quad C \end{cases} \tag{2.6}$$

Finally, the DFT (X, Y, and Z) and power spectrum (P) of the Z-Curve representation may be defined as:

$$\begin{aligned} X[k] &= \textstyle\sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}} \\ Y[k] &= \textstyle\sum_{n=0}^{N-1} y[n] e^{\frac{-j2\pi kn}{N}} \\ Z[k] &= \textstyle\sum_{n=0}^{N-1} z[n] e^{\frac{-j2\pi kn}{N}} \end{aligned} \tag{2.7}$$

$$P[k] = |X[k]|^2 + |Y[k]|^2 + |Z[k]|^2, k = 1, 2, ...N. \tag{2.8}$$

2.1.2 Integer Representation

This representation is one-dimensional [18, 21]. This mapping can be obtained by substituting the four nucleotides (T, C, A, G) of a biological sequence for integers (0, 1, 2, 3), respectively. Following are the steps to perform feature extraction using Integer Representation:

- Takes biological sequence as input and apply numeric mapping

$$b[n] = \begin{cases} 0, s[n] = G \\ 1, s[n] = A \\ 2, s[n] = C \\ 3, s[n] = T \end{cases} \tag{2.9}$$

- Perform Discrete Fourier Transform on numerical sequence

$$B[k] = \sum_{n=0}^{N-1} b[n] e^{\frac{-j2\pi kn}{N}} \tag{2.10}$$

- Perform Power Spectrum for the biological sequence

$$P[k] = |B[k]|^2 \tag{2.11}$$

- Apply feature extraction on the resulted data.

2.2 Fuzzy Clustering

In general hard clustering algorithms, each data point belongs to one cluster center. Whereas in soft clustering or fuzzy clustering algorithm, the extent to which one data point belongs to another cluster center is defined and cluster centers are found out depending on that.Membership degree represents the extent to which one data point belongs to another cluster center. Each data-point x_i is an object and has a set of membership degrees $u_{ij} : j \epsilon (1, c)$ associated with it. Group of all membership degrees of all data-points make the membership matrix U. Since Bezdek's[22] initial fuzzy clustering technique, Fuzzy c-Means(FCM), the clustering approach has progressed significantly. Iterative optimization is used in the FCM clustering technique to minimise an objective function on feature space using a similarity measure.

$$J_m(U,\ V) = \sum_{i=1}^{n} \sum_{j=1}^{c} u_{ij}^m \|x_i - v_j\|^2, \quad m > 1 \tag{1}$$

2.2.1 Kernel Functions

The kernel trick is an implicit non-linear map (ϕ) from the input space X to a high dimensional feature space R [23].

$$\phi : x \to \phi(x) \in R^d$$

where the data samples $\{x_1, x_2, ..., x_s\} \subseteq X$. In this, an input data space with lower dimension is mapped to potentially much higher dimensional feature space (R) or inner product [24]. A mercer kernel described as a function can express the inner product operation in kernel space K below:

$$K(x_i, v_j) = \phi(x_i)^T \phi(v_j)$$

where $x_i, v_j \in R^d$ such that $i = 1, .., s$ and $j = 1, ..., c$.
By the kernel substitution, we have the following equation.

$$\begin{aligned} &\|\phi(x_i) - \phi(v_j)\|^2 = (\phi(x_i) - \phi(v_j))^T (\phi(x_i) - \phi(v_j)) \\ &= \phi(x_i)^T \phi(x_i) - \phi(v_j)^T \phi(x_i) - \phi(x_i)^T \phi(v_j) + \phi(v_j)^T \phi(v_j) \\ &= K(x_i, x_i) + K(v_j, v_j) - 2K(x_i, v_j) \end{aligned} \tag{2.12}$$

This results in the creation of a new class of non-Euclidean distance measures in the original input space (along with a Euclidean distance in feature space). As a result, the original space will be measured differently by various kernels.
For example $K(x_i, v_j)$ is the Radial Basis Function (RBF) kernel [25], which is a well-known kernel function represented as follows:

$$K(x_i, v_j) = \exp(-\|x_i - v_j\|^2 / \sigma^2) \tag{2.13}$$

The kernel parameter is indicated by the symbol *sigma*. The selection of kernel parameters is the most important task, according to [25]. The kernel parameter is chosen in this study in the following manner:

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

like this we have explored many other kernel functions in our work which will get introduced in the next chapters.

2.2.2 KSLFCM

SLFCM can work with linear relationships. The kernel approach is used to expand SLFCM in order to accommodate non-linear relations. The KSLFCM method is a kernelized variant of the SLFCM algorithm that employs several kernel functions. Data samples and cluster centre values are used to compute the membership degree. As a result, the membership degree of specific data samples may be calculated in simultaneously on many slave nodes.
Algorithmic Steps:

1. KSLFCM algorithm calculates the membership degree separately for each data sample. Eq. (6).

2. In Line 2 of KSLFCM algorithm, it parallely calculates membership values for all data samples on apache spark by making use of the Map and ReduceByKey functions.

3. The cluster centre values are updated from membership degrees of all data samples in line 3 of the method. As a result, this line is run after all membership degrees have been computed. The membership degrees of all data samples are integrated and kept as a membership knowledge I' at the master node, which is necessary to update the cluster centre v_j by Eq (7).

4. The difference between the previous initialised cluster centre values and the newly computed cluster centre values is calculated on line 4. This technique is repeated until no change in the values of cluster centres is detected. Following that, all iterations are run in order since the updated cluster centres are needed as input for the following iteration.

PsuedoCode of KSLFCM:

Algorithm 2: *KSLFCM to Iteratively Minimize* $J_p(M, V')$

Input: X, c, p, ϵ, (initial V); X is an array of data samples such that $X = \{x_1, x_2, ...x_s\}$.
Output: I', V'
1: If V is not initialized, randomly initialize $V = \{v_1, v_2, ...v_c\}$.
2: **Compute** membership knowledge by using Eq. (6).
$I' = X.Map(V).ReduceByKey()$
3: **Compute** the set of final cluster centers $V' = \{v'_1, v'_2, ...v'_c\}$ by using Eq. (7).
4: If $\| V' - V \| < \epsilon$ then stop, otherwise go to step 2.
5: **Return** I', V'.

2.2.3 KSRSIO-FCM

We propose three Kernelized Scalable Random Sampling With Iterative Optimization Fuzzy c-Means (KSRSIO-FCM) algorithms implemented on the Apache Spark framework with different three different kernels as Cauchy Kernel, Logarithmic Kernel, Gaussian Radial-Basis Kernel. The objective function for these algorithms would be the same as that of FCC. The KSRSIO-FCM algorithms work by partitioning data across slave nodes into equal sized subsets by selecting 100% of the data without any replacement for Big Data Algorithm summarizes the steps of the Base algorithm (KSRSIO-FCM).

The dataset is divided into a number of subsets or chunks, and the cluster centres are initialised with a few randomly selected data points in the first chunk. The base algorithm then calculates the cluster centres and membership values for the first chunk, let's say $X1$, represented by V and I, respectively, using the second base algorithm (KSLFCM), whose description is explained in the next section. The estimated cluster centres V are then used as an input for clustering the second subset $X2$. Now, using the second base algorithm (KSLFCM), the base algorithm (KSRSIO-FCM) clusters $X2$ and determines the cluster centres and membership knowledge represented by I and V. However,for the clustering of the third subset, KSRSIO-FCM does not use V as an input. This is because KSRSIO-FCM takes into account the fact that random partitioning might result in two continuous subsets with data samples from different classes. As a result, the cluster centres of these two subgroups will differ dramatically. As a result, for the clustering

of the current subset, KSRSIO-FCM avoids using cluster centres as the original cluster centres from the prior iteration. Rather, it combines the membership values of all the processed subsets, i.e. it combines I and I', and uses Eq.(7) to update cluster centres. Because they are computed with the combined membership knowledge of a greater number of data samples that spans a bigger sample region, this method of determining cluster centres is the most prominent approach of estimating real cluster centres.

The I numerator and denominator of V are computed using the KSLFCM technique. Combining membership matrices is identical to union of the very first subset $I1$ and the second subset $I2$ because membership values of one data sample are independent on membership values of other data samples. As a result, rather than allocating a large amount of storage space for I, we may integrate $I1$ and $I2$ without losing any information. This aids in space optimization; this optimization analogy also applies to the remaining subsets, i.e. all $s \in [3, s]$, where s is the number of subsets. Because operations on one subset are performed serially, KSLFCM will effectively consume only $(\frac{1}{s})^{th}$ times of the space. We save a large amount of space and processing time as a result of this.
The workflow of KSRSIO-FCM is shown in Fig. 2.1, which uses KSLFCM to compute membership knowledge and cluster centres for all subsets. It shows how the dataset is randomly divided into subgroups and how the underlying cluster centres for clustering the initial subset are picked at random. [13]

PseudoCode of KSRSIO :

Algorithm 1: *KSRSIO-FCM to Iteratively Minimize* $J_p(M, V')$

Input: X, c, p, ϵ; X is an array of data samples such that $X = \{x_1, x_2, ...x_s\}$.
Output: $I^{'}, V^{'}$
1: **Partition** set X into n subsets such that $X = \{X_1, X_2, ...X_n\}$.
2: **Randomly** select X_1 from X without replacement where X_1 represents the first subset consist of random s/n samples.
3: $I^{'}, V^{'} = KSLFCM(X_1, c, p, \epsilon)$
4: **for** $t = 2$ to n do
  4.1: $I, V' = KSLFCM(X_t, c, p, \epsilon, V')$
  4.2: Merge the partition of all blocks of processed subsets
  for $j = 1$ to c **do**
    $I_j^{'} =< j, < (sum\_d_j x)_{I_j}, +(sum\_d_j x)_{I_j^{'}}, (sum\_d_j)_{I_j}, +$
    $(sum\_d_j)_{I_j^{'}} >>$
  end for
  4.3: Compute updated cluster center v_j' using:
  $< j, < sum\_d_j x, sum\_d_j >>$ in I' by Eq. (7) $\forall \in [1, c]$
  end for
5: **Compute** the objective function using Eq. (5).
6: **Return** I', V'

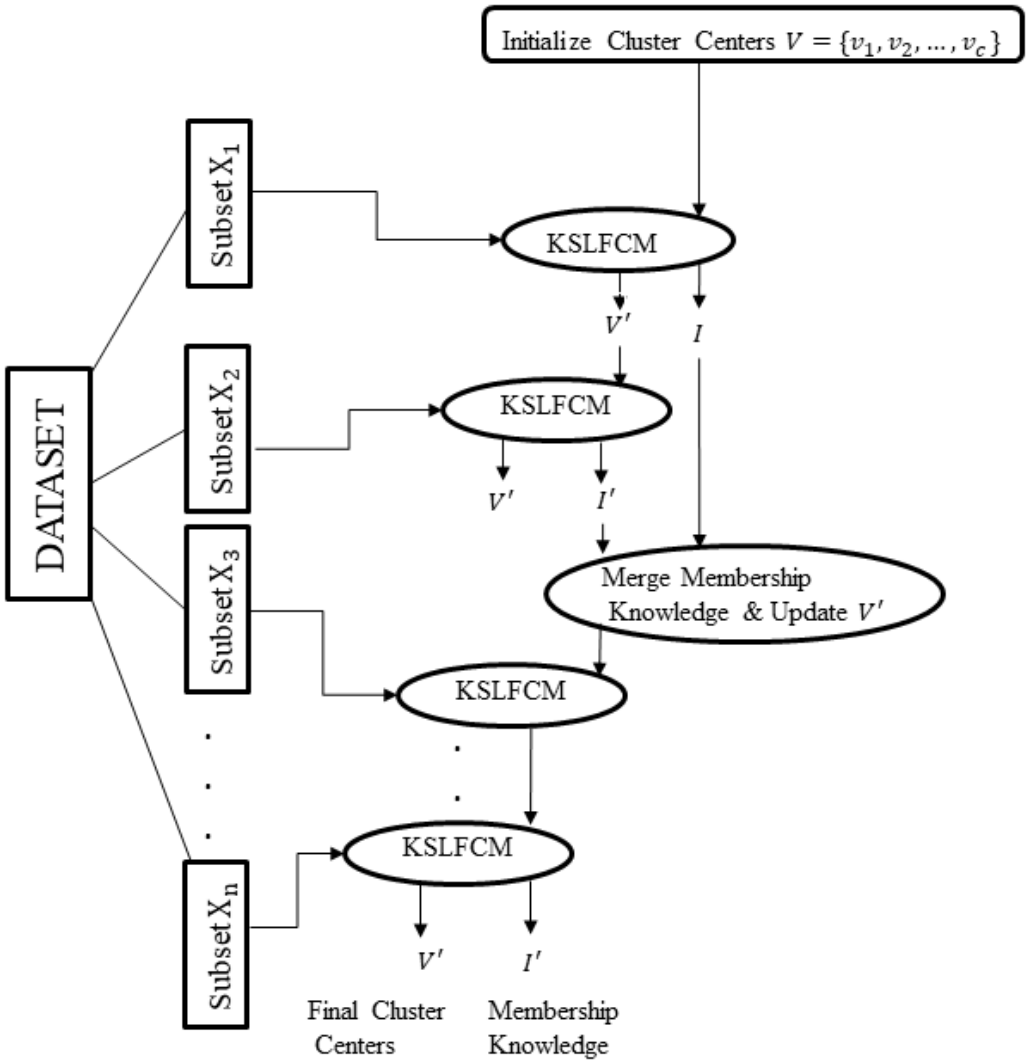

FIGURE 2.2: Workflow of KSRSIO-FCM algorithm.

Chapter 3

Tools and Frameworks used for computations

3.1 Introduction

The frameworks and resources utilised to handle Big Data are discussed in the next chapter. To do calculations for our BTech research, we used a **Apache Spark** cluster in Prof. Aruna Tiwari's Big Data lab. We also utilised two **HPCs**: **Paramshakti** from IIT Kharagpur and **Param Siddhi-AI** from CDAC Pune. In the next parts, we utilised **Hadoop Distributed File System (HDFS)**, which will be detailed in detail. **Graphics Processing Units (GPUs)** on the Param Siddhi-AI supercomputer were used to parallelize our feature extraction techniques.

3.2 Apache Spark

3.2.1 Spark Introduction

At the University of California, Berkeley, Apache Spark was established in 2009. It's an open-source data processing engine for in-depth research. It has swiftly risen to prominence among consumer electronics, offering a severe challenge to Hadoop MapReduce.

3.2.2 Purpose

Apache Spark is a framework for parallelizing parallelizable calculations via Application Programming Interfaces APIs without worrying about the complexity of parallel algorithm implementation. The framework offers a wide range of APIs, making it simple to connect their use to the parallelizable phases of the algorithm. Apache Spark is a scalable in-memory computing platform for big data processing. It allows for the analysis of subsets of the dataset in parallel across a cluster. Apache Spark is a high-performance cluster computing solution with easy and efficient programming APIs that enable the slave node to retrieve and execute the dataset repeatedly. By executing a spark job on the Hadoop framework to share a cluster and dataset while ensuring continuous service and response levels, Spark's in-memory cluster computing technology improves application processing performance.

3.2.3 Internal working of Apache Spark clusters

The Apache Spark clusters are depicted in Figure 3.1. One master node and a number of worker nodes comprise the Apache Spark cluster. A driver is a master that is used to schedule tasks. With jobs, steps, and tasks, Spark starts a hierarchical scheduling process. The map and reduce phases are matched using a subset of tasks partitioned from collective jobs. DAG Scheduler and Task Scheduler are two components of the Apache Spark framework. For each job, the DAG Scheduler generates a directed acyclic graph (DAG). It also keeps track of the RDD record for each step's outputs, whereas the Task Schedule sends tasks to the cluster from each step. By allowing the master to connect to a cluster manager, standalone in our example, Apache Spark offers several cluster modes to the user for the execution of their Apache Spark programme. Each worker node has its own executor.The executor is responsible for running the tasks and caching the data in memory or disk.

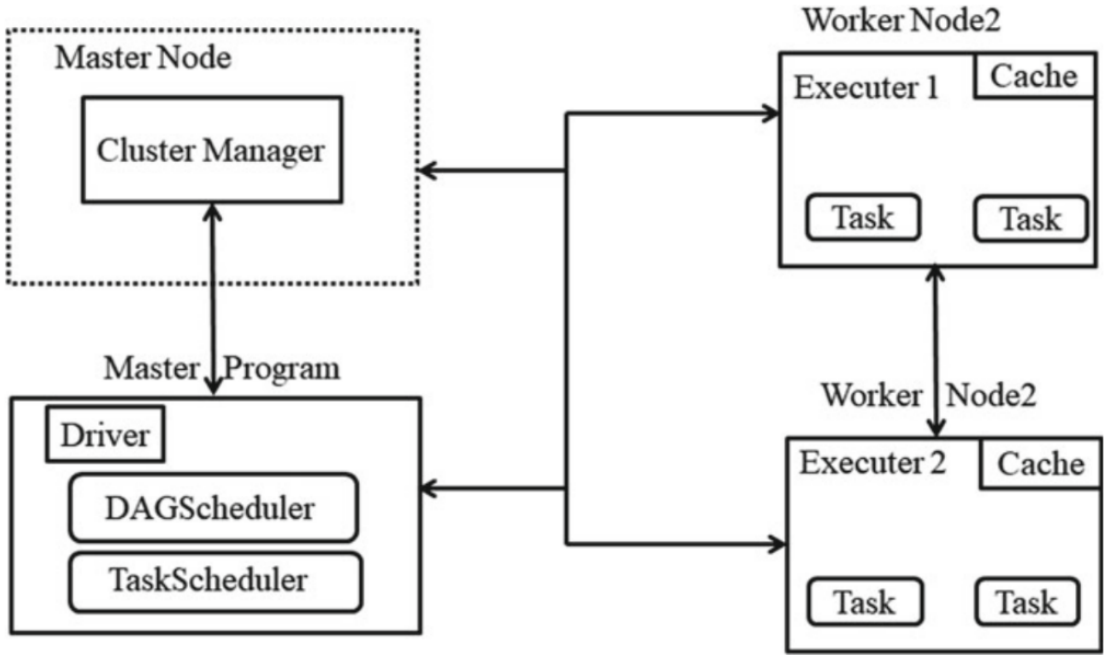

FIGURE 3.1: Overview of Apache Spark Cluster

3.2.4 Apache Spark APIs used for the project

We implemented our algorithms in Python and we incorporated Spark APIs in Python through Pyspark. The APIs used in the project are map, reduceByKey, union.

3.2.4.1 map(function)

map() API implements the Map fuction described in the previous chapter. It takes a function as an argument and applies that function to each data-point in the dataset. This API, splits the dataset among the worker nodes and apply the function on each data-point, on each worker node, thus parallelizing the process and reducing the time

3.2.4.2 reduceByKey(aggregate function)

reduceByKey() API implements the ReduceByKey function described in the previous chapter. It takes aggregate function as an argument, divides the whole RDD into buckets, based on the key value and applies the aggregate function on each bucket and returns a new RDD containing the output of the aggregate function on each bucket.

3.2.4.3 union(list of RDDs)

union() API coalesces two RDDs into a single RDD.

3.2.5 Hardware Description of Apache Spark clusters

There are 1 Server, 1 master, 5 slave nodes in the lab

3.2.5.1 Server specifications

Total number of cores is 32, total available memory is 187 GB and total disk space is 12 TB.

3.2.5.2 Master specifications

It is a Dell precision Tower 5810 workstation, it has 4 cores and its RAM is 32 GB

3.2.5.3 Slave Nodes specifications

They have Intel(R) Core(TM) i7-77000 CPU with 3.60 GHZ frequency and each slave node has 1 TB storage space.

3.3 High Performance Computing through supercomputers

For the project, we used 2 High Performance Computing (HPC) machines i.e supercomputers of India, Paramshakti and Param Siddhi-AI. The hardware descriptions of these supercomputers are given below.

3.3.1 Paramshakti

PARAM Shakti is IIT Kharagpur's supercomputer. PARAM Shakti systems are based on Intel Xeon SKL G-6148, NVIDIA Tesla V100 with total peak performance of 1.6 PFLOPS.

3.3.2 Param Siddhi-AI

PARAM Siddhi-AI is CDAC- Pune's supercomputer. Param Siddhi-AI, a machine with 210 AI Petaflops (6.5 Petaflops Peak DP), is based on the NVIDIA DGX SuperPOD reference architecture comprising of 42 NVIDIA DGX A100 systems.

3.4 Hadoop Distributed File System

Hadoop File System is based on a distributed file system architecture. It runs on standard hardware. HDFS, unlike other distributed systems, is designed with low-cost hardware and is highly fault-tolerant. HDFS can store a lot of data and makes it easy to access it. The files are spread over numerous machines to accommodate such large amounts of data. These files are duplicated to protect the system from data loss in the event of a system failure. HDFS also enables parallel processing of applications.

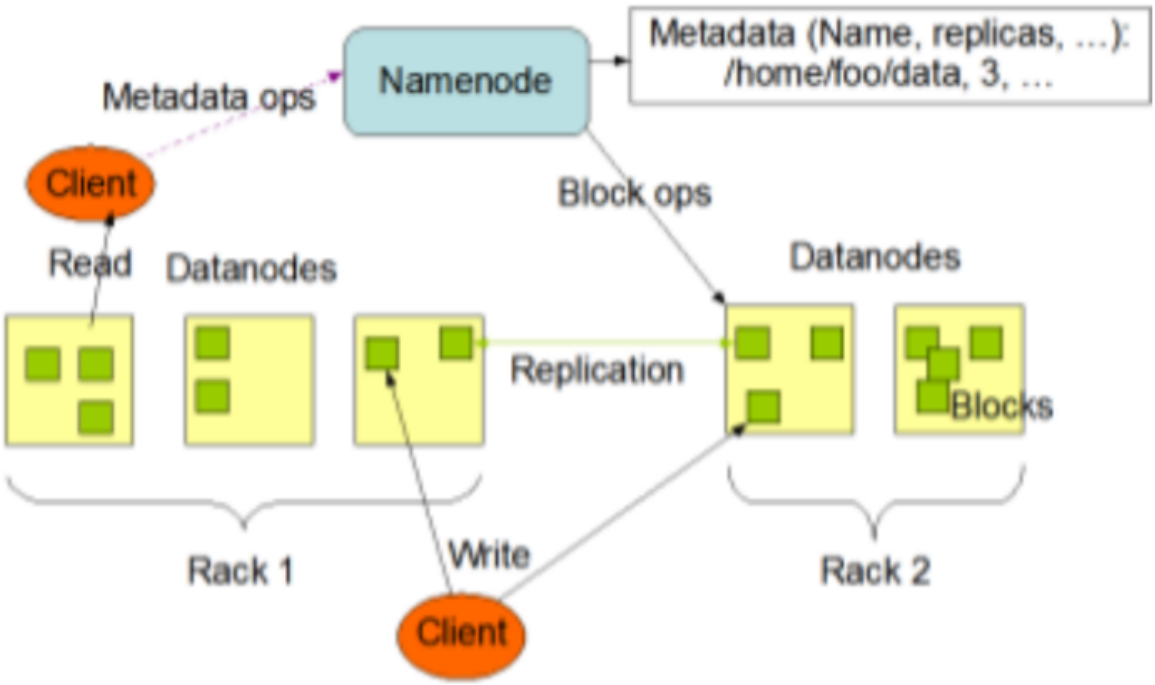

FIGURE 3.2: HDFS Architecture[26]

3.4.1 Features of HDFS

- It is suitable for distributed storage and processing.
- To communicate with HDFS, Hadoop provides a command interface.
- The namenode and datanode built-in servers make it simple for users to examine the cluster's status.
- Access to file system data in real time.
- File permissions and authentication are provided by HDFS.

3.4.2 Namenode

The namenode is a piece of commodity hardware that houses the GNU/Linux operating system as well as the namenode software. It's the heart of the HDFS file system. It

maintains the directory tree of all files in the file system and records where the file data is stored across the cluster. It does not save the contents of these files. The master server is the machine that contains the namenode and is responsible for the following tasks:

- Controls the namespace of the file system.
- Client access to files is regulated.
- File system activities including renaming, renaming, shutting, closing, and opening files and directories are performed.

3.4.3 Datanode

The data node is made up of commodity hardware that has been installed with the GNU/Linux operating system and data node software. Each node (common hardware/system) in a cluster will have a data node. These nodes are in charge of the data storage for the system. Client requests are handled by datanodes, which conduct read-write operations on file systems. In accordance with the name node's orders, they also perform block creation, creation, deletion, deletion, and replication.

3.4.4 Block

The user data is typically saved in HDFS files. HDFS files are divided into data blocks, which are block-sized chunks. These blocks are kept as separate units. In other terms, a Block is the smallest quantity of data that HDFS can read or write. The default block size is 64MB, although it can be extended if the HDFS setup requires it.

3.5 Graphics Processing Units

Unlike CPUs, GPUs have multiple Arithmetic Logic Units (ALUs), which speeds up the parallelizable portions of the process. In this project, we employed GPUs to speed up the process of extracting Featured vectors from snp data. To programmatically add GPUs, we used the Python modules Numba and CuPy, which both use CUDA APIs to communicate with the hardware GPUs.

3.5.1 CUDA

CUDA is a parallel computing platform and programming paradigm developed by NVIDIA. They provide APIs that deal with the GPU's low-level complexity. Many libraries, packages, and frameworks leverage CUDA APIs to implement their logic.

3.5.2 Numba

It's a free Just-In-Time (JIT) compiler that converts a portion of Python and NumPy code into machine code. By converting a chunk of Python code into CUDA kernels, this package facilitates CUDA GPU development.

3.5.3 CuPy

CuPy is a NumPy compatible array library for Python GPU acceleration[27].

Chapter 4

Proposed Methods

4.1 Proposed Scalable Feature Extraction Technique

we propose two pipelines to perform clustering on various RNA datasets, extracted from different parts of a soybean plant, and we compare the clustering efficiencies of these two pipelines on different RNA datasets. Both of the pipelines have two parts:

1. Convert the raw RNA datasets into a workable dataset (compatible with clustering, regression, neural networks, and so on) which is also known as the Feature Extraction sub-pipeline.

2. Using numeric features to cluster the obtained dataset.

In the first stage of the pipeline, it takes genomic data (DNA and RNA) and converts them into a dataset containing a particular number of numerical columns. Figure 4.1 shows the workflow of proposed methods. In this paper, we constructed two such pipelines, called 13d-SZcurve pipeline and the 13d-SEIIP pipeline, which are elucidated in the subsequent sections, and we compared the clustering efficiencies between the pipelines for different RNA datasets pertaining to different parts of the soybean plant. Both of these pipelines differ only in the feature extraction sub-pipeline. The scalable RNA feature extraction method is discussed in Algorithm 1.

Algorithm 1 Scalable RNA Feature Extraction Method

Require: X (Input data), Representation (feature extraction method(13d-SZcurve or 13d-SEIIP)).
Ensure: X' (features of corresponding data)
1: Compute power spectrum from input data $power\_spectrum = empty\_RDD$
2: **if** $Representation = 13d - SEIIP$ **then**
3: $power\_spectrum = X.Map$(13d-SEIIP)
4: Compute feature vectors.
5: $X' = power\_spectrum.Map(Feature\_Extract)$
6: Function $Feature\_Extract$ produce 13 features as discussed in point 7.

The Algorithm 1 presents sub-pipeline task in 7 stages. The feature extraction sub-pipeline takes the string "representation" as input, which decides the pipeline to be run. Stages of the feature extraction pipeline that are common between the two pipelines are:

1. Dataset parallelization: The given dataset must be converted into a Resilient Distributed Dataset (RDD) using spark APIs, where RDD is the fundamental data

structure of Spark. By using Sparks Map Application Programming Interface (API), for each row of the dataset, we perform the subsequent steps of the pipeline.

2. Biological Sequence: The biological sequence is a series of nucleotides in DNA and RNA, and it cannot be computed; it has to be converted into a series of numbers by the feature extraction pipeline, and it is the input to the whole clustering pipeline.

3. Numerical mapping: In this stage, the nucleotides A, C, G, and T are replaced by integers or the biological sequence is converted to a sequence of numbers of the same length. From this stage of the feature extraction pipeline, the two pipelines that we propose in this paper differ. In one pipeline, we used the 13-dimensional Scalable Z-curve (13d-SZcurve) numerical mapping scheme, and in the other pipeline, we used the 13-dimensional Scalable EIIP (13d-SEIIP) numerical mapping scheme. This step serves as a basis to get the final featured vector.

4. Numerical Sequence: In this stage, after applying the numerical mapping scheme to each nucleotide in every row in the raw RNA dataset, a numeric sequence is generated pertaining to each initial row in the raw RNA dataset. This sequence is then used in the next stage.

5. Discrete Fourier Transform (DFT) and Power Spectrum: In this stage, the numerical sequence obtained from the previous step is then transformed into its DFT, and the power spectrum is calculated from the DFT.

6. Fourier Spectrum: The fourier spectrum or power spectrum is extracted from the Previous stage.

   Steps from 2 to 6 are done in function Map($Dataset\_RDD$, 13d-SEIIP) and Map($Dataset\_RDD$, 13d-SZcurve) algorithm procedural blocks, and the Step 6 output is stored in a RDD called $power\_spectrum\_RDD$. Then, on this RDD, by applying Spark Map API, on each row of the RDD, we apply the $Feature\_Extract$ transformation, which is elucidated in the subsequent step.
   This is mentioned in the $Map(power\_spectrum\_RDD, Feature\_Extract)$.

7. Feature Extraction : In this stage, the power spectrum RDD that we got in the previous stage is taken as an input and 13 features are extracted from power spectrum like average, minimum, maximum, etc. as shown in Figure 4.1. This stage is denoted by $Map(power\_spectrum\_RDD, Feature\_Extract)$ in Figure 4.1. Here, $Feature\_Extract$ is the function for calculating 13 feature vectors. And performs feature extraction on $power\_spectrum\_RDD$ by using Spark Map API, for each row by calculating minimum, maximum, mean, median, peak, standard deviation, standard deviation population, variance, amplitude, percentile15, percentile25, percentile50, percentile75 of the row and return this final array and insert this results array in a new RDD called $Final\_dataset\_RDD$, which is the final output of the feature extraction pipeline. The detailed description of proposed scalable feature extraction algorithms (13d-SZcurve and 13d-SEIIP) are presented next.

4.1.1 Proposed 13d-SZcurve

As mentioned before, based on the numerical mapping stage, there are 2 pipelines. The 13d-SZcurve pipeline and the 13d-SEIIP pipeline. The steps mentioned in the below 13d-SZcurve pipeline cover only the sequence of actions which are performed in Map(*Dataset_RDD*, 13d-SZcurve) and 13d-SEIIP pipeline covers only the sequence of actions which are performed in Map(*Dataset_RDD*, 13d-SEIIP). Algorithm 2 discusses 13d-SZcurve method. It takes the RNA sequences as input. In Line 1, length of sequences is initialized. Line 2 generates the numeric mapping from the raw RNA sequences and return three numerical sequences x, y and z and each sequence is of the same length as the corresponding RNA sequence. Using Line 3-16, We generate 3 separate numerical sequences x, y and z for the RNA sequence. Then perform DFT of x, y and z sequences from Line 17-18 for the RNA sequence where, X is DFT of x sequence, Y is DFT of y sequence, and Z is DFT of z sequence. Thus, returns three sequences of complex numbers X, Y and Z. Line 19-20 performs power spectrum for the RNA sequences by using the corresponding X, Y and Z and return a sequence of real numbers P.

Algorithm 2 13d-SZcurve

Require: s
Ensure: P
1: Initialization of arrays $N = length(s)$, X[N], Y[N], Z[N], P[N].
2: Generating three separate numerical sequences x, y, z for each row of the raw RNA dataset.
3: **if** s[0] = A or G **then**
4: x[0] = 1
5: **else**
6: x[0] = -1
7: **if** $s[0]$ = A or C **then**
8: $y[0]$ = 1
9: **else**
10: $y[0]$ = -1
11: **if** $s[0]$ = A or T **then**
12: $z[0] = 1$
13: **else**
14: $z[0]$ = -1
15: **for** k = 1 to N -1 **do**
16: as per Eq. 2.4-2.6.
17: **for** k = 0 to N -1 **do**
18: Eq. 2.7
19: **for** k = 0 to N -1 **do**
20: Eq. 2.8

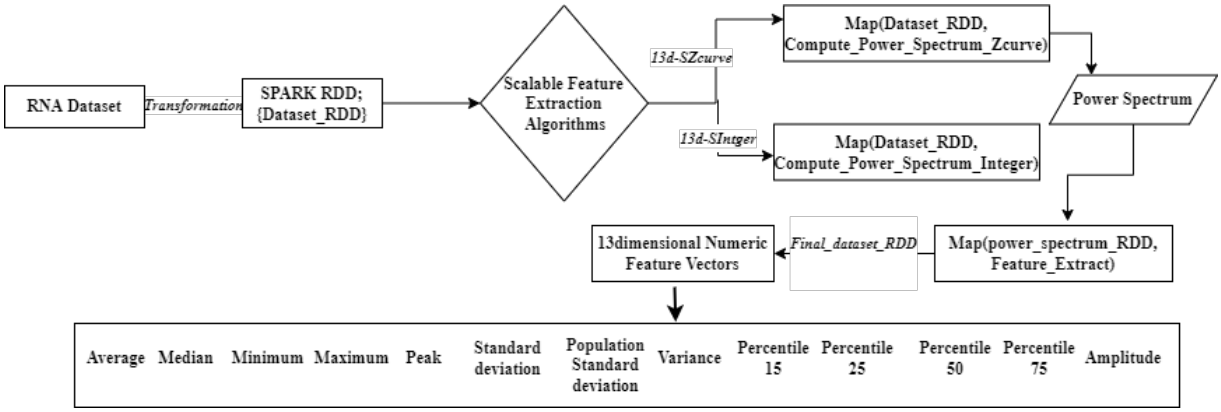

FIGURE 4.1: Workflow of the scalable feature extraction sub pipeline

4.1.2 Proposed 13d-SInteger

Similarly we proposed 13d-SInteger method, basic pipeline of steps is same as 13d-SZcurve method. Algorithm 3 discusses the proposed method. It takes the RNA sequences as input. In Line 1, length of sequences is initialized. Line 2 generates the numeric mapping from the raw RNA sequences and return one numerical sequence b and length of the sequence is same as of the corresponding RNA sequence. Then perform DFT of b in Line 33-37 for the RNA sequence where, B is DFT of b sequence. Line 38-39 performs power spectrum for the RNA sequences by using the corresponding B and return a sequence of real numbers P.

Algorithm 3 13d-SInteger

Require: s
Ensure: P
1: Initialization of arrays $N = length(s)$, B[N], P[N].
2: Generating numerical sequence b for each row of the raw RNA dataset.
3: **for** k = 0 to N -1 **do**
4: as per Eq. 2.9
5: **for** k = 0 to N -1 **do**
6: Eq. 2.10
7: **for** k = 0 to N -1 **do**
8: Eq. 2.11

4.1.3 GPU accelerated 13d-SZcurve

Algorithm 1, discusses 13d-SZcurve. First, it initializes the length of the sequence in Line 1. In Line 2, empty RDD is created for the computation of power spectrum. In Line 3, a Map function is used to distribute *Compute_Power_Spectrum_Zcurve* method onto the worker nodes for parallel processing. Line 4-8 describes the working of *Compute_Power_Spectrum_Zcurve* function. Line 5 generates the numeric mapping from the raw RNA sequences and return three numerical sequences x, y, and z and each sequence is of the same length as the corresponding RNA sequence. Line 6-7, we updated the values of each dimension of an array. Then perform DFT of x, y and z sequences from Line

8-9 for the RNA sequence. Thus, returns three sequences of complex numbers of DFT. Line 10-11 performs the power spectrum for the RNA sequences by using the corresponding complex numbers obtained in previous steps. In Line 11, kernel function of power spectrum has been executed, which runs on GPU. In Line 12, feature vectors are computed from power spectrum RDD. Line 13 returns *Final_dataset_RDD*, containing numerical feature vectors of 13 dimensions.

Algorithm 1: *GPU accelerated 13d-SZcurve*

Input: s (Raw RNA data)
Output: X' (13 feature vectors)
1: **Initialize** N = length(s).
2: **Compute** power spectrum from input data $power\_spectrum = empty\_RDD$.
3: $power\_spectrum = X.Map(Dataset\_RDD\ , Compute\_Power\_Spectrum\_Zcurve)$
4: $Function\ \ Compute\_Power\_Spectrum\_Zcurve\{$
5: Generating three separate numerical sequences x, y, z for the given RNA sequence.
6: **For** n = 1 to N -1
7: **Update** the values of each dimension using Eq. 2.4-2.6.
8: **For** k = 0 to N -1
9: **Calculate** the DFT using Eq. 2.7.
10: **For** k = 0 to N -1
11: Kernel function executed to compute power spectrum using Eq. 2.8 $\}$.
12: $X' = power\_spectrum.Map(Feature\_Extract)$.
13: **Return** $Final\_dataset\_RDD$.

4.2 Proposed Scalable Kernelized Fuzzy Clustering Algorithms

4.2.1 Proposed CKSRSIO-FCM Algorithm

In this subsection, we elucidated the mercer kernel used in the proposed CKSRSIO-FCM algorithm, then derived the formulae for cluster centers and membership degrees of each datapoint in each cluster using Lagrange multipliers optimization method and then list those formulae at the end of this subsection.

4.2.1.1 Kernel Description

Cauchy kernel is derived from Cauchy distribution (Basak, 2008). It's a long-tailed kernel that may be utilised to offer high-dimensional space long-range effect and sensitivity. The following equation represents this kernel:

$$K(x_i, v_j) = \frac{1}{(1 + \frac{\|x_i - v_j\|^2}{\sigma^2})} \tag{4.1}$$

4.2.1.2 Numerical Derivation

1. Objective function

$$J(U,\ V) = \sum_{i=1}^{n}\sum_{k=1}^{c} u_{ik}^{m}\|\phi(x_i) - \phi(v_k)\|^2, \quad m > 1 \tag{4.2}$$

where ϕ stands as map.
And distance function can be expressed using in a product space as

$$\|\phi(x_i) - \phi(v_j)\|^2 = \langle\phi(x_i), \phi(x_i)\rangle + \langle\phi(v_k), \phi(v_k)\rangle - 2\langle\phi(x_i), \phi(v_k)\rangle \tag{4.3}$$

2. Kernel Induced Measure from Eq. (4.2) we can express kernel function as

$$K(x_i, v_k) = \langle\phi(x_i), \phi(v_k)\rangle \tag{4.4}$$

based on the explanation of similarity measure on feature space.
Therefore we have the new Kernel induced distance function as

$$\|\phi(x_i) - \phi(v_j)\|^2 = K(x_i, x_i) + K(v_k, v_k) - 2K(x_i, v_k) \tag{4.5}$$

where i=1,2,3 ...n and k=1,2,3,...c

3. Recalculation of Effective Objective Function
Cauchy Kernel can be defined as

$$K(x_i, v_j) = 1 - log(1 + \|x_i - v_j\|^2/\sigma^2) \tag{4.6}$$

Now $K(x_i, x_i)$ = 1 and $K(v_k, v_k) = 1$
Therefore ,

$$\|\phi(x_i) - \phi(v_j)\|^2 = 2(1 - K(x_i, v_k)) \tag{4.7}$$

$$J(U,\ V) = 2\sum_{i=1}^{n}\sum_{k=1}^{c} u_{ik}^{m}(1 - K(x_i, v_k)), \quad m > 1 \tag{4.8}$$

4. Obtaining Membership

   4.1 To Obtain equation for calculating membership we minimize the objective function

$$J(U,\ V) = 2\sum_{i=1}^{n}\sum_{k=1}^{c} u_{ik}^{m}(1 - K(x_i, v_k)) \ \ subject\ to \ \ \sum_{k=1}^{c} u_{ik} = 1 \tag{4.9}$$

   4.2 Objective Function with lagrange multiplier and constraints

$$J(U, V, \lambda) = 2\sum_{i=1}^{n}\sum_{k=1}^{c} u_{ik}^{m}(1 - K(x_i, v_k)) - \sum_{i=1}^{n}\lambda_i\left(\sum_{k=1}^{c} u_{ik} - 1\right) \tag{4.10}$$

$$\begin{aligned} &\textit{where } \lambda = (\lambda_1, \lambda_2, \lambda_3, ...\lambda_n) \textit{ subject to constraints} \\ &\sum_{k=1}^{c} u_{1k} = 1 \\ &\sum_{k=1}^{c} u_{2k} = 1 \\ &\vdots \\ &\sum_{k=1}^{c} u_{nk} = 1 \end{aligned} \tag{4.11}$$

4.3 Write final function before calculating derivative

$$J(U,V,\lambda) = 2\left[\sum_{i=1}^{n} u_{i1}^{m}\left(1-k(x_i,v_1)\right) + u_{i2}^{m}\left(1-k(x_i,v_2)\right) + \cdots + u_{ic}^{m}\left(1-k(x_i,v_c)\right)\right] - \sum_{i=1}^{n}\lambda_i\left((u_{i1}+u_{i2}+u_{i3}+\cdots+u_{ic})-1\right) \tag{4.12}$$

Expanding above equation we get

$$\begin{aligned} J(U,V,\lambda) = &\left[(u_{11}^{m}\left(1-K(x_1,v_1)\right) + u_{21}^{m}\left(1-K(x_2,v_1)\right) + \cdots + u_{n1}^{m}\left(1-K(x_n,v_1)\right))\right] \\ &+ \left[(u_{12}^{m}\left(1-K(x_1,v_2)\right) + u_{22}^{m}\left(1-K(x_2,v_2)\right) + \cdots + u_{n2}^{m}\left(1-K(x_n,v_2)\right))\right] \\ &+ \cdots + \left[(u_{1c}^{m}\left(1-K(x_1,v_c)\right) + u_{2c}^{m}\left(1-K(x_2,v_c)\right)\cdots + u_{nc}^{m}\left(1-K(x_n,v_c)\right))\right] - \\ &\left[(\lambda_1 u_{11} + \lambda_2 u_{21} + \cdots + \lambda_n u_{n1}) + \right. \\ &(\lambda_1 u_{12} + \lambda_2 u_{22} + \cdots + \lambda_n u_{n2}) + \\ &\cdots + (\lambda_1 u_{1c} + \lambda_2 u_{2c} + \cdots + \lambda_n u_{1c}) - \\ &\left.(\lambda_1 + \lambda_2 + \cdots + \lambda_n)\right. \end{aligned} \tag{4.13}$$

4.4 Using the needed condition Lagrangian technique, we may take the objective function's derivative with respect to u and set the first derivative to zero.

$$\begin{aligned}
\frac{\partial J}{\partial u_{11}} &= 2mu_{11}^{m-1}(1-K(x_1,v_1)) - \lambda_1 = 0 \\
&\Rightarrow 2mu_{11}^{m-1}(1-K(x_1,v_1)) = \lambda_1 \\
&\Rightarrow u_{11}^{m-1}(1-K(x_1,v_1)) = \frac{\lambda_1}{2m} \\
&\Rightarrow u_{11}^{m-1} = \frac{\lambda_1}{2m(1-K(x_1,v_1))} \\
\Rightarrow u_{11} &= \left(\frac{\lambda_1}{2m(1-K(x_1,v_1))}\right)^{\frac{1}{m-1}} \\
\frac{\partial J}{\partial u_{12}} &= 2mu_{12}^{m-1}(1-K(x1,v1)) - \lambda_1 = 0 \\
\Rightarrow u_{12} &= \left(\frac{\lambda_1}{2m(1-K(x1,v2))}\right)^{1/m-1} \\
&\vdots \\
\Rightarrow u_{1c} &= \left(\frac{\lambda_1}{2m(1-K(x1,vc))}\right)^{1/m-1}
\end{aligned} \tag{4.14}$$

4.5 Summing up all u_{1c},u_{2c},u_{3k} ...and final calculation of u_{nc}

$$\begin{aligned}
u_{11}+u_{12}+\cdots+u_{1c} = &\left(\frac{\lambda_1}{2m(1-K(x_1,v_1))}\right)^{1/m-1} + \\
&\left(\frac{\lambda_1}{2m(1-K(x_1,v_2))}\right)^{1/m-1} + \\
&\cdots \\
&\left(\frac{\lambda_1}{2m(1-K(x_1,v_c))}\right)^{1/m-1} \\
= &\left(\frac{\lambda_1}{2m}\right)^{1/m-1}\left[\sum_{j=1}^{c}\left(\frac{1}{1-K(x_1,v_j)}\right)^{1/m-1}\right]
\end{aligned} \tag{4.15}$$

Since our proposed partition matrix in an objective function satisfies the following conditions

$$\sum_{k=1}^{c} u_{ik} = 1 \quad i = 1,2,3\ldots n$$

We have

$$\left(\frac{\lambda_1}{2m}\right)^{1/m-1}\left[\sum_{j=1}^{c}\left(\frac{1}{1-K(x_1,v_j)}\right)^{1/m-1}\right] = 1$$

We obtain

$$\left(\frac{\lambda_1}{2m}\right)^{1/m-1} = \frac{1}{\left[\sum_{j=1}^{c}\left(\frac{1}{1-K(x_1,v_j)}\right)^{1/m-1}\right]} \tag{4.16}$$

substituting eq. (4.15) in u_{11} ,we have

$$u_{11} = \frac{\left(\frac{1}{1-K(x_1,v_1)}\right)^{\frac{1}{m-1}}}{\sum_{j=1}^{c}\left(\frac{1}{1-K(x_1,v_j)}\right)^{\frac{1}{m-1}}}$$

Similarly we have

$$u_{12} = \frac{\left(\frac{1}{1-K(x_1,v_2)}\right)^{\frac{1}{m-1}}}{\sum_{j=1}^{c}\left(\frac{1}{1-K(x_1,v_j)}\right)^{\frac{1}{m-1}}}$$

$$\vdots$$

$$u_{1c} = \frac{\left(\frac{1}{1-K(x_1,v_c)}\right)^{\frac{1}{m-1}}}{\sum_{j=1}^{c}\left(\frac{1}{1-K(x_1,v_j)}\right)^{\frac{1}{m-1}}}$$

In similar way we obtain

$$u_{2c} = \left(\frac{\lambda_1}{2m(1-K(x_2,v_c))}\right)^{1/m-1}$$

$$u_{2c} = \frac{\left(\frac{1}{1-K(x_2,v_c)}\right)^{\frac{1}{m-1}}}{\sum_{j=1}^{c}\left(\frac{1}{1-K(x_2,v_j)}\right)^{\frac{1}{m-1}}}$$

$$\vdots$$

$$u_{nc} = \left(\frac{\lambda_1}{2m(1-K(x_n,v_c))}\right)^{1/m-1}$$

$$u_{nc} = \frac{\left(\frac{1}{1-K(x_n,v_c)}\right)^{\frac{1}{m-1}}}{\sum_{j=1}^{c}\left(\frac{1}{1-K(x_n,v_j)}\right)^{\frac{1}{m-1}}}$$

In general we have

$$u_{ik} = \frac{\left(\frac{1}{1-K(x_i,v_k)}\right)^{\frac{1}{m-1}}}{\sum_{j=1}^{c}\left(\frac{1}{1-K(x_i,v_j)}\right)^{\frac{1}{m-1}}} \tag{4.17}$$

The general equation is used to obtain membership grades for objects in data for finding meaningful groups and we use this expression to compute the expression for Cluster Centers

5. Obtaining Cluster Center

5.1 Taking the derivative of proposed objective function

$$J\left(U,V,\lambda\right)=2\sum_{i=1}^{n}\sum_{k=1}^{c}u_{ik}^{m}(1-K\left(x_i,v_k\right))-\sum_{i=1}^{n}\lambda_i\left(\sum_{k=1}^{c}u_{ik}-1\right)$$

with respect to v, The objective function can be written as

$$J\left(U,V,\lambda\right)=2\sum_{i=1}^{n}\sum_{k=1}^{c}u_{ik}^{m}\left(\frac{1}{1+\frac{\|x_i-v_j\|^2}{\sigma^2}}\right)\left(\frac{\|x_i-v_j\|^2}{\sigma^2}\right)-\sum_{i=1}^{n}\lambda_i\left(\sum_{k=1}^{c}u_{ik}-1\right) \tag{4.18}$$

since

$$K(x_i,v_j)=1/(1+\|x_i-v_j\|^2/\sigma^2)$$

$$\frac{1}{K(x_i,v_j)}=1+\frac{\|x_i-v_i\|^2}{\sigma^2}$$

$$\frac{\|x_i-v_i\|^2}{\sigma^2}=\frac{1-K(x_i,v_j)}{K(x_i,v_j)}$$

$$1-K(x_i,v_j)=\left(\frac{1}{1+\frac{\|x_i-v_j\|^2}{\sigma^2}}\right)\left(\frac{\|x_i-v_j\|^2}{\sigma^2}\right) \tag{4.19}$$

5.2 Expanding

$$\begin{aligned}J(U,V,\lambda)=2\sum_{i=1}^{n}\Bigg[&u_{i1}^{m}\left(\frac{1}{1+\frac{\|x_i-v_1\|^2}{\sigma^2}}\right)\left(\frac{\|x_i-v_1\|^2}{\sigma^2}\right)+\\&u_{i2}^{m}\left(\frac{1}{1+\frac{\|x_i-v_2\|^2}{\sigma^2}}\right)\left(\frac{\|x_i-v_2\|^2}{\sigma^2}\right)+\cdots+\\&u_{ic}^{m}\left(\frac{1}{1+\frac{\|x_i-v_c\|^2}{\sigma^2}}\right)\left(\frac{\|x_i-v_c\|^2}{\sigma^2}\right)\\&-\sum_{i=1}^{n}\lambda_i\left(\sum_{k=1}^{c}u_{ik}-1\right)\end{aligned}$$

5.3 Taking derivative
Taking the objective function's derivative with respect to v_1 and setting the first derivative to zero using the Lagrangian method's necessary condition,

we get

$$\frac{\partial J}{\partial v_1} = 2\sum_{i=1}^{n} u_{i1}^m \left(\frac{1}{1+\frac{\|x_i-v_1\|^2}{\sigma^2}}\right)^2 \frac{2\cdot\|x_i-v_1\|}{\sigma^2} = 0$$

$$4\sum_{i=1}^{n} u_{i1}^m K(x_i,v_1)^2 \|x_i-v_1\| = 0$$

since

$$K(x_i,v_1) = \frac{1}{1+\frac{\|x_i-v_1\|^2}{\sigma^2}}$$

Expanding

$$\sum_{i=1}^{n} u_{i1}^m K(x_i,v_1)^2 x_i - \sum_{i=1}^{n} u_{i1}^m K(x_i,v_1)^2 v_1 = 0$$

$$v_1 = \frac{\sum_{i=1}^{n} u_{i1}^m K(x_i,v_1)^2 x_i}{\sum_{i=1}^{n} u_{i1}^m K(x_i,v_1)^2}$$

Similarly taking derivative with respect to v_2

$$\frac{\partial J}{\partial v_2} = 2\sum_{i=1}^{n} u_{i2}^m \left(\frac{1}{1+\frac{\|x_i-v_2\|^2}{\sigma^2}}\right)^2 \frac{2\cdot\|x_i-v_2\|}{\sigma^2} = 0$$

$$4\sum_{i=1}^{n} u_{i2}^m K(x_i,v_2)^2 \|x_i-v_2\| = 0$$

$$\sum_{i=1}^{n} u_{i2}^m K(x_i,v_2)^2 x_i - \sum_{i=1}^{n} u_{i2}^m K(x_i,v_2)^2 v_2 = 0$$

$$v_2 = \frac{\sum_{i=1}^{n} u_{i2}^m K(x_i,v_2)^2 x_i}{\sum_{i=1}^{n} u_{i2}^m K(x_i,v_2)^2}$$

$\vdots$

$$\frac{\partial J}{\partial v_c} = 2\sum_{i=1}^{n} u_{ic}^m \left(\frac{1}{1+\frac{\|x_i-v_1\|^2}{\sigma^2}}\right)^2 \frac{2\cdot\|x_i-v_1\|}{\sigma^2} = 0$$

$$4\sum_{i=1}^{n} u_{ic}^m K(x_i,v_c)^2 \|x_i-v_c\| = 0$$

since

$$K(x_i,v_c) = \frac{1}{1+\frac{\|x_i-v_c\|^2}{\sigma^2}}$$

Expanding

$$\sum_{i=1}^{n} u_{ic}^m K(x_i,v_c)^2 x_i - \sum_{i=1}^{n} u_{ic}^m K(x_i,v_c)^2 v_c = 0$$

$$v_c = \frac{\sum_{i=1}^{n} u_{ic}^m K(x_i, v_c)^2 x_i}{\sum_{i=1}^{n} u_{ic}^m K(x_i, v_c)^2}$$

5.4 In general, we have

$$v_k = \frac{\sum_{i=1}^{n} u_{ik}^m K(x_i, v_k)^2 x_i}{\sum_{i=1}^{n} u_{ik}^m K(x_i, v_k)^2} \tag{4.20}$$

4.2.1.3 Membership Degree and Cluster Center Equation

$$U_{ij} = \frac{(1 - K(x_i, v_j))^{1/(m-1)}}{\sum_{j=1}^{c} (1 - K(x_i, v_j))^{1/(m-1)}} \tag{4.21}$$

$$V_j = \frac{\sum_{i=1}^{n} u_{ij}^m K(x_i, v_j)^2 x_i}{\sum_{i=1}^{n} u_{ij}^m K(x_i, v_j)^2} \tag{4.22}$$

4.2.2 Proposed LKSRSIO-FCM Algorithm

In this subsection, we elucidated the mercer kernel used in the proposed LKSRSIO-FCM algorithm, then derived the formulae for cluster centers and membership degrees of each datapoint in each cluster using Lagrange multipliers optimization method and then list those formulae at the end of this subsection.

4.2.2.1 Kernel Description

$$K(x_i, v_j) = 1 - log(1 + \frac{\|x_i - v_j\|^2}{\sigma^2}) \tag{4.23}$$

4.2.2.2 Numerical Derivation Steps

1. Change in step 3
   Log Kernel can be defined as

$$K(x_i, v_j) = 1 - log(1 + \|x_i - v_j\|^2/\sigma^2) \tag{4.24}$$

   Now $K(x_i, x_i) = 1$ and $K(v_k, v_k) = 1$
   Therefore ,

$$\|\phi(x_i) - \phi(v_j)\|^2 = 2(1 - K(x_i, v_k)) \tag{4.25}$$

$$J(U, V) = 2\sum_{i=1}^{n}\sum_{k=1}^{c} u_{ik}^m (1 - K(x_i, v_k)), \quad m > 1 \tag{4.26}$$

2. Change in step 5.1
   Taking the derivative of proposed objective function

$$J(U, V, \lambda) = 2\sum_{i=1}^{n}\sum_{k=1}^{c} u_{ik}^m (1 - K(x_i, v_k)) - \sum_{i=1}^{n} \lambda_i \left(\sum_{k=1}^{c} u_{ik} - 1 \right)$$

with respect to v, The objective function can be written as

$$J(U,V,\lambda) = 2\sum_{i=1}^{n}\sum_{k=1}^{c} u_{ik}^{m} \log\left(1+\frac{\|x_i-v_j\|^2}{\sigma^2}\right) - \sum_{i=1}^{n}\lambda_i\left(\sum_{k=1}^{c} u_{ik}-1\right) \tag{4.27}$$

since

$$K(x_i,v_j) = 1 - log\left(1+\frac{\|x_i-v_j\|^2}{\sigma^2}\right)$$

$$1-K(x_i,v_j) = log\left(1+\frac{\|x_i-v_j\|^2}{\sigma^2}\right)$$

3. Change in step 5.2

$$\begin{aligned} J(U,V,\lambda) = 2\sum_{i=1}^{n}\Bigg[& u_{i1}^{m}\, log\left(1+\frac{\|x_i-v_j\|^2}{\sigma^2}\right) + \\ & u_{i2}^{m}\, log\left(1+\frac{\|x_i-v_j\|^2}{\sigma^2}\right) + \cdots + \\ & u_{ic}^{m}\, log\left(1+\frac{\|x_i-v_j\|^2}{\sigma^2}\right) \\ & -\sum_{i=1}^{n}\lambda_i\left(\sum_{k=1}^{c} u_{ik}-1\right) \end{aligned}$$

4. Change in step 5.3

$$\frac{\partial J}{\partial v_1} = 2\sum_{i=1}^{n} u_{i1}^{m}\left(\frac{1}{1+\frac{\|x_i-v_1\|^2}{\sigma^2}}\right)\frac{2\cdot\|x_i-v_1\|}{\sigma^2} = 0$$

$$4\sum_{i=1}^{n} u_{i1}^{m}\left(\frac{1}{1+\frac{\|x_i-v_1\|^2}{\sigma^2}}\right)\|x_i-v_1\| = 0$$

Therefore by following same steps from numerical derivation we get,

$$v_1 = \frac{\sum_{i=1}^{n} u_{i1}^{m}\left(\frac{1}{1+\frac{\|x_i-v_1\|^2}{\sigma^2}}\right) x_i}{\sum_{i=1}^{n} u_{i1}^{m}\left(\frac{1}{1+\frac{\|x_i-v_1\|^2}{\sigma^2}}\right)}$$

5. In general, we have

$$v_k = \frac{\sum_{i=1}^{n} u_{ik}^m \left(\frac{1}{1+\frac{\|x_i - v_j\|^2}{\sigma^2}} \right) x_i}{\sum_{i=1}^{n} u_{ik}^m \left(\frac{1}{1+\frac{\|x_i - v_j\|^2}{\sigma^2}} \right)} \tag{4.28}$$

4.2.2.3 Membership Degree and Cluster Center Equation

$$U_{ij} = \frac{(1 - K(x_i, v_j))^{1/(m-1)}}{\sum_{j=1}^{c} (1 - K(x_i, v_j))^{1/(m-1)}} \tag{4.29}$$

$$V_j = \frac{\sum_{i=1}^{n} u_{ij}^m \left(\frac{1}{1+\frac{\|x_i - v_j\|^2}{\sigma^2}} \right) x_i}{\sum_{i=1}^{n} u_{ij}^m \left(\frac{1}{1+\frac{\|x_i - v_j\|^2}{\sigma^2}} \right)} \tag{4.30}$$

4.2.2.3.1 KSRSIO Algorithm Kernel Description

$$K(x_i, v_j) = \exp(-\|x_i - v_j\|^2/\sigma^2) \tag{4.31}$$

4.2.2.3.2 Membership Degree and Cluster Center Equation

$$U_{ij} = \frac{(1 - K(x_i, v_j))^{1/(m-1)}}{\sum_{j=1}^{c} (1 - K(x_i, v_j))^{1/(m-1)}} \tag{4.32}$$

$$V_j = \frac{\sum_{i=1}^{n} u_{ij}^m K(x_i, v_j) x_i}{\sum_{i=1}^{n} u_{ij}^m K(x_i, v_j)}$$

After analysing number of mercer kernel functions like HyperTangent Kernel, Sigmoid Kernel, Cauchy Kernel, Logarithmic Kernel , Quadratic Kernel , Inverse Quadratic Kernel etc. We finally chose two of them Cauchy and Logarithmic Kernels and have derived their Membership Values and Cluster Center Equations. We propose ZKSRSIO-FCM an optimisation of kernalized fuzzy clustering, here Z represents any kernel function that we would want to encorpate and ZKSLFCM which is an kernelized version of SLFCM. The next subsections go through both of these topics.

4.2.3 Z-Kernelized Scalable Literal Fuzzy c-Means (ZKSLFCM), Z= Cauchy and Logarithmic

As discussed in the introduction, SLFCM can work with linear relationships. The kernel approach is used to expand SLFCM in order to accommodate non-linear relations. The ZKSLFCM method is a kernelized variant of the SLFCM algorithm that employs several kernel functions. Data samples and cluster centre values are used to compute the membership degree.
Algorithmic Steps:

1. ZKSLFCM algorithm calculates the membership degree separately for each data sample. Eq. (6).

2. In Line 2 of ZKSLFCM algorithm, it parallely calculates membership values for all data samples on apache spark by making use of the Map and ReduceByKey functions.

3. The cluster centre values are updated from membership degrees of all data samples in line 3 of the method. As a result, this line runs after all membership degrees have been computed. The membership degrees of all data samples are integrated and kept as a membership knowledge I' at the master node, which is necessary to update the cluster centre v_j by Eq (7).

4. The difference between the previous initialised cluster centre values and the newly computed cluster centre values is calculated on line 4. This technique is repeated until no change in the values of cluster centres is detected. Following that, all iterations are run in order since the updated cluster centres are needed as input for the following iteration.

Algorithm 4: *ZKSLFCM to Iteratively Minimize $J_p(U, V')$*

Input: X, c, p, ϵ, (initial V); X is an array of data samples such that $X = \{x_1, x_2, ...x_s\}$.
Output: I', V'
1: If V is not initialized, randomly initialize $V' = \{V_1, V_2, ...V_c\}$.
2: **Compute** membership knowledge.
$I' = X.Map(V).ReduceByKey()$
3: **Compute** cluster centers.
3.1: **If** Z= Cauchy then:
3.2: $$V'_j = \frac{\sum_{i=1}^{s} u_{ij}^p K(x_i,v_j)^2 x_i}{\sum_{i=1}^{s} u_{ij}^p K(x_i,v_j)^2} \forall j.$$
3.3: **Else if** Z= Logarithmic then:
3.4: $$V'_j = \frac{\sum_{i=1}^{s} u_{ij}^p \left(\frac{1}{1+\|x_i-v_j\|^2/\sigma^2}\right) x_i}{\sum_{i=1}^{s} u_{ij}^p \left(\frac{1}{1+\|x_i-v_j\|^2/\sigma^2}\right)} \forall j.$$
3.5: **end if**
4: If $\| V' - V \| < \epsilon$ then stop, otherwise $V = V'$ and go to step 2.

We avoid aggregating the membership matrix of data samples to lower the space need in the proposed ZKSLFCM method. The membership matrix M is required in Algorithm 4 to compute the cluster centres (V')We avoid aggregating the membership matrix of data samples to lower the space need in the proposed ZKSLFCM method. The membership matrix M is required in Algorithm 4 to compute the cluster centres (V'). Rather than saving the huge membership matrix, we use mapper and reducer technique such that we calculate numerator contribution and denominator contribution for calculating cluster centers. So $m_{ij}^p K(x_i, v_j)x_i$, and $m_{ij}^p Z(x_i, v_j)$, where m_{ij}^p is denoted as d_{ij} and $Z(x_i, v_j)$ denoted as z_{ij}. Numerator contribution is denoted as $d_{ij}z_{ij}x_i$ and denominator contribution as $d_{ij}z_{ij}$. This eliminates the need to store the massive membership matrix M. After mapping values for all data samples we do the summation of all the $d_{ij}z_{ij}x_i$ values and all the $d_{ij}z_{ij}$ values belonging to the particular cluster center v_j and store it as membership knowledge in variable I'. After that, we access these values in from line 3-4 of Algorithm 4 to calculate the updated cluster centers. Fig. 4.2 shows the whole methodology of space improvement.

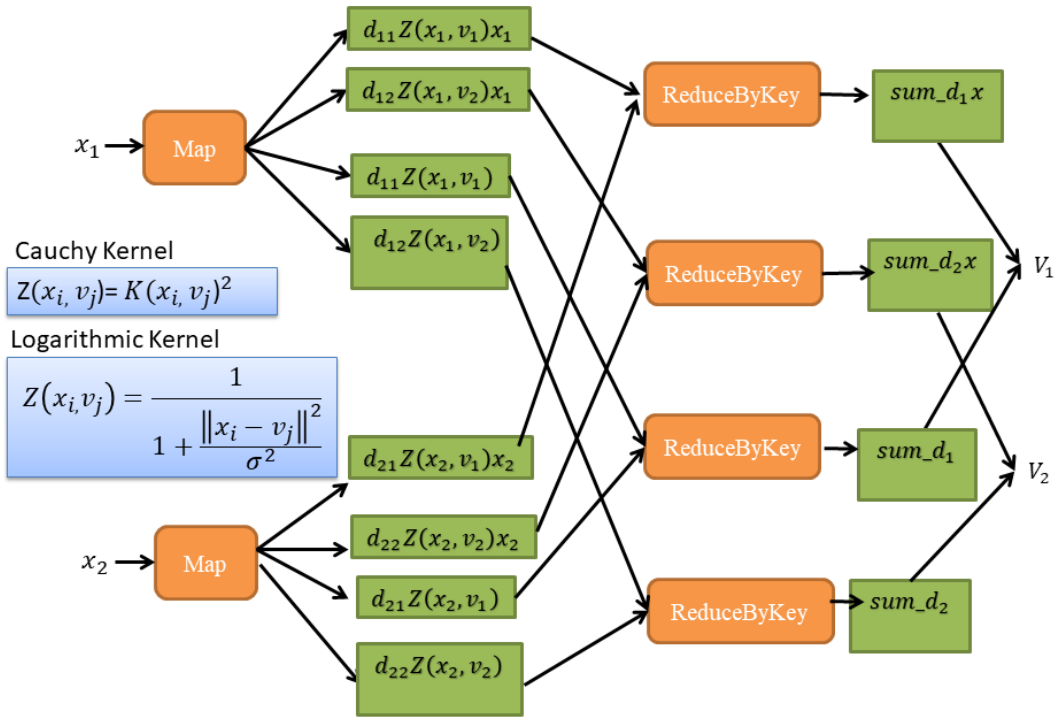

FIGURE 4.2: The figure describes repository space improvement by avoiding the storage of membership matrix of subsets.

4.2.3.1 Map Function

The master node divides the data logically, while the slave nodes deal with sampled data pieces. To determine data point's membership degree, the data sample itself and cluster center values are employed. As a consequence, the membership degrees of two

data points are computed separately. As a result, we run this procedure on slave node for each data sample individually before merging the results on master node. As a result of reduced processing time, ZKSLFCM method is significantly quicker.

Algorithm 5: *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: j = index of cluster center v.
3: $d_{ij}k_{ij} = d_{ij}$(membership degree of x_i concerning v_j,
k_{ij} (kernel value for an i^{th} data sample in the j^{th} cluster).
4: $d_{ij}k_{ij}x_i = d_{ij}k_{ij} * x_i$
5: **yield** $< j, < d_{ij}k_{ij}x_i, d_{ij}k_{ij} >>$
6: **end for**

4.2.3.2 ReduceByKey Function

Map function in the first stage returns multiple key-value pairs have the same key value. Now ReduceByKey performs summation on those pairs belonging to same key. On two different key-value pairs, Spark deals with similar tasks on all the key-value pairs. we have to find the numerator and denominator to update the cluster center attributes. ReduceByKey gives the numerator and denominator of each cluster center v_j as sum_d_jx and sum_d_j, respectively. In the algorithm 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 numerator part Map algoritm 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 denominator part Map algorithm outputs a and b respectively.

Algorithm 6: *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**:$< j, < sum\_d_jx, sum\_d_j >>$

4.2.4 Z-Kernelized Scalable Random Sampling with Iterative Optimization Fuzzy c-Means (ZKSRSIO-FCM), Z= Cauchy and Logarithmic

The dataset is divided into a number of subsets or chunks, and the cluster centres are initialised with a few randomly selected data points in the first chunk. The base algorithm

then calculates the cluster centres and membership values for the first chunk, let's say $X1$, represented by V and I, respectively, using the second base algorithm (ZKSLFCM), whose description is explained in the next section. The estimated cluster centres V are then used as an input for clustering the second subset $X2$. Now, using the second base algorithm (ZKSLFCM), the base algorithm (ZKSRSIO-FCM) clusters $X2$ and determines the cluster centres and membership knowledge represented by I and V. However,for the clustering of the third subset, ZKSRSIO-FCM does not use V as an input. This is because ZKSRSIO-FCM takes into account the fact that random partitioning might result in two continuous subsets with data samples from different classes. As a result, the cluster centres of these two subgroups will differ dramatically. As a result, for the clustering of the current subset, ZKSRSIO-FCM avoids using cluster centres as the original cluster centres from the prior iteration. Rather, it combines the membership values of all the processed subsets, i.e. it combines I and I', and uses Eq.(7) to update cluster centres. Because they are computed with the combined membership knowledge of a greater number of data samples that spans a bigger sample region, this method of determining cluster centres is the most prominent approach of estimating real cluster centres.

The I numerator and denominator of V are computed using the ZKSLFCM technique. Combining membership matrices is identical to union of the very first subset $I1$ and the second subset $I2$ because membership values of one data sample are independent on membership values of other data samples. As a result, rather than allocating a large amount of storage space for I, we may integrate $I1$ and $I2$ without losing any information. This aids in space optimization; this optimization analogy also applies to the remaining subsets, i.e. all $s \in [3, s]$, where s is the number of subsets. Because operations on one subset are performed serially, ZKSLFCM will effectively consume only $(\frac{1}{s})^{th}$ times of the space. We save a large amount of space and processing time as a result of this.
The workflow of ZKSRSIO-FCM is shown in 4.3 figure, which uses KSLFCM to compute membership knowledge and cluster centres for all subsets.

Algorithm 7: *ZKSRSIO-FCM to Iteratively Minimize* $J_p(U, V')$

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: $I^{'}, V^{'} = ZKSLFCM(X_1, c, p, \epsilon)$
4: **for** $t = 2$ to s do
4.1: $I, V' = ZKSLFCM(X_t, c, p, \epsilon, V')$
4.2: Merge the partition of all blocks of processed subsets
for $j = 1$ to c **do**
$I_j^{'} =< j, < (sum\_d_j x)_{I_j}, +(sum\_d_j x)_{I_j'}, (sum\_d_j)_{I_j}, +$
$(sum\_d_j)_{I_j'} >>$

end for
4.3: Compute updated cluster center v'_j using:
$< j, < sum\_d_j x, sum\_d_j >>$ in $I' \forall \in [1, c]$
end for
5: **Return** I', V'

Fig. 4.3 demonstrates the workflow of ZKSRSIO-FCM, which makes use of ZKSLFCM 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.

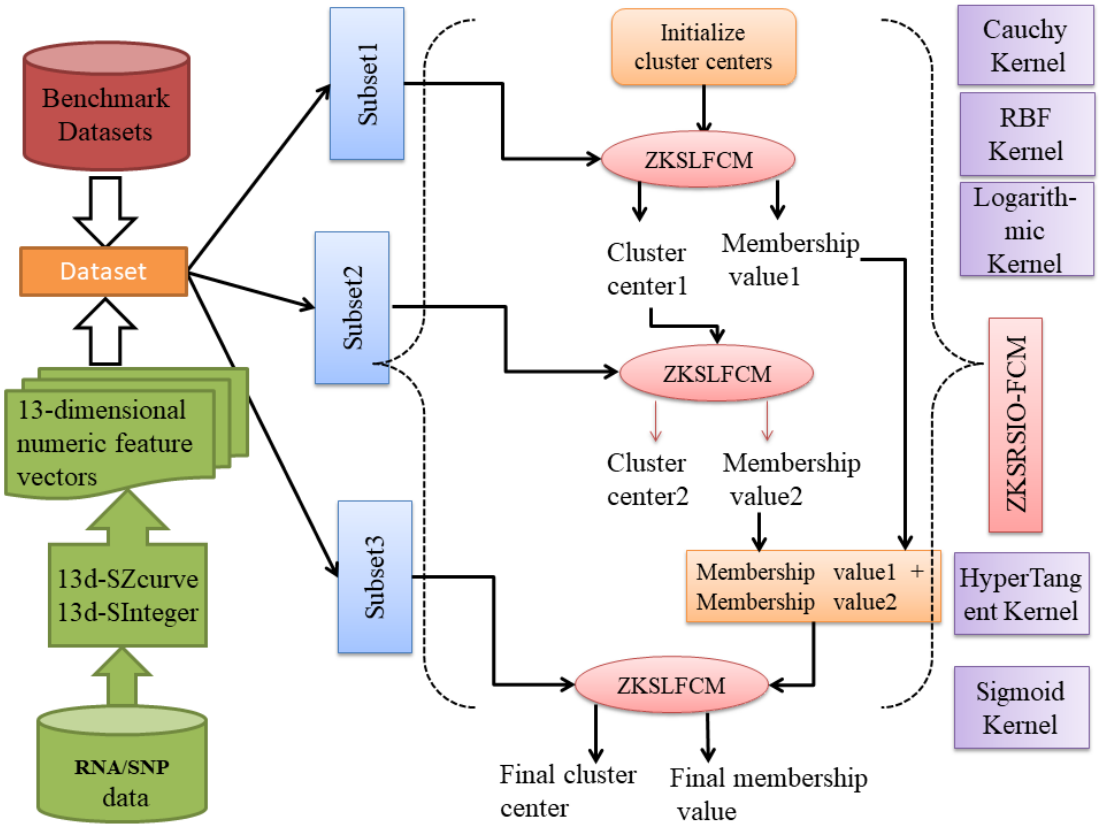

FIGURE 4.3: Workflow of ZKSRSIO-FCM algorithm.

4.2.5 Complexity Analysis of ZKSRSIO-FCM and ZKSLFCM

With large amounts of data and quadratic distance equations, several researchers adopted kernel-based clustering techniques. Our proposed ZKSLFCM and ZKSRSIO-FCM algorithms have linear complexity in terms of the input data sample, according to the complexity study. The space complexity refers to the quantity of data kept in RAM during the calculation. X refers to the dataset that contains s data points in d high dimensional space such that $X = \{x1, ..., xs\}$, $xi \in R^d$, where c is the number of clusters,

w is the number of slave nodes in the Spark cluster, and X is randomly distributed into n number of partitions, each subset containing random s/n samples. t is the number of iterations required to complete the termination. ZKSLFCM is run on the entire dataset.Every iteration of the Map stage takes $O(scdt/w)$ time and $O(scd)$ space across all slave nodes, as each slave node works on (s/w) data, assuming ZKSLFCM runs for t iterations. Because map results are only retained in memory once each iteration, the overall space complexity of the Map stage of ZKSLFCM is $O(scd)$. On each slave node, the ReduceByKey action adds s/w Map results corresponding to a specific cluster centre in parallel. Each slave node produces c outputs, which are aggregated and added on the master node. It takes $O(cd/w)$ time and $O(cd/w)$ space to accomplish this. In this vein, the ReduceByKey step has a total time complexity of $O(scdt/w)$ and a space complexity of $O(cd/w)$. As a result, ZKSLFCM has a time complexity of $O(scdt/w)$ and a space complexity of $O(scd)$), where $s >> w$ and c. The ZKSRSIO-FCM algorithm divides the entire dataset X into n equal subsets. ZKSLFCM is run consecutively over each of these subsets. ZKSLFCM across each subset has time and space complexity of $O(scdt/nw)$ and $O(scd/n)$, respectively. The time complexity is $O(scdt/w)$ since each subset is treated sequentially, while the space complexity is $O(scd/n)$ because data relating to one subset is not retained in memory while processing the next subset.[13]

In Table 4.1, Clustering performed by ZKSRSIO-FCM on each subset converges by taking the the less number of iterations (t) for each subset. Hence, ZKSRSIO-FCM has lesser run-time as it performs clustering on a small chunk of data in each subset in comparison with ZKSLFCM that performs clustering of the whole data.

TABLE 4.1: Complexity Analysis of Kernelized Algorithm.

Algorithm	**Time Complexity**	**Space Complexity**
ZKSRSIO-FCM	$O(scdt/w)$	$O(scd/n)$
ZKSLFCM	$O(scdt/w)$	$O(scd)$

Chapter 5

Experiments and Results

In this chapter, from section 5.1 to section 5.3, we discuss the datasets we used, which includes benchmark datasets and Real life data such as, SNP and RNA, various performance measures used to evaluate the efficiencies of clustering algorithms, all the results of various components of experimentation and then, in section 5.4, we discuss the way we prepared our Real Life Dataset and at the end, in section 5.5 we discuss how we performed gene identification on the new dataset.

5.1 Datasets Description

5.1.1 Benchmark Datasets

We use these benchmark datasets to examine the efficiency of our proposed Scalable Kernelized Fuzzy clustering algorithms. The benchmark datasets we used are:

5.1.1.1 Avila:

Avila Dataset is a 12 class dataset, with 10 features, which contains description of the images extracted from Avila Bible and each class represents the name of the copyist. It is borrowed from UCI repository[28]. It is replicated to 12 GB.

5.1.1.2 Skin monarch:

Skin monarch dataset is a 2 class dataset, with 4 features. This dataset is prepared from Skin dataset, which is available on UCI repository[28]. Skin dataset is cross multiplied to another dataset, Monarch[29] to prepare a final dataset of size 13 GB.

5.1.1.3 Wine:

Wine dataset is a 3 class dataset, with 12 features.It is borrowed from UCI repository[28]. It is replicated to 11 GB.

5.1.2 RNA Datasets

In the experimental study, the three soybean RNA datasets of RNA-Seq Atlas[1] of Glycine max were preprocessed using the proposed 13d-SZcurve and 13d-SEIIP methods. Soybean (Glycine max) is a significant crop that provides significant amounts of protein and oil. The RNA Seq-Atlas offered on the soybean expression atlas contains high-resolution gene expression data from fourteen distinct tissues. We used raw data from three tissues: flower, shell, and pod RNA datasets. The size of Flower, Shell, and POD is 1 GB, 1.32 GB, and 1.06 GB, respectively. We used RNA dataset for studying the Scalable feature extraction techniques we proposed.

5.1.3 SNP Datasets

In this experimental study, we used three **Single Nucleotide Polymorphism (SNP)** datasets. They are:

5.1.3.1 soysnp50k wm82.a1:

This dataset is used to validate the superiority of our proposed feature extraction techniques and proposed Scalable Kernelized Fuzzy clustering algorithms.

5.1.3.2 soysnp50k wm82.a2:

This dataset is used to generate the novel dataset.

5.1.3.3 MAGIC-Raw-genotype-data:

This dataset is used to compare the performance of our proposed GPU accelerated 13d S-Integer feature extraction technique with CPU enabled 13d S-Integer feature extraction technique.

5.2 Performance Measures

We used two kinds of measures to evaluate the efficiency of our proposed algorithms. They are **External performance measures** and **Internal performance measures**. We used **external measures** to evaluate the efficiency of various clustering algorithms on **replicated Benchmark datasets** which have labels associated with them and **internal measures** to evaluate the efficiency of various algorithms on **RNA** and **SNP** datasets which don't have labels associated with them.

5.2.1 External Performance Measures

We used three external performance measures. They are:

[1] https://soybase.org/soyseq/tables_lists/index.php

5.2.1.1 Normalized Mutual Information (NMI)

The NMI [30] is used to evaluate clustering quality by calculating the proportion of common data for clustering, ground truth, and their harmonic mean. The NMI is characterized as pursues[13]:

$$NMI = \frac{\sum_{c=1}^{k}\sum_{q=1}^{r} s_c^q \log\left(\frac{s.s_c^q}{s_c.s_q}\right)}{\sqrt{\left(\sum_{c=1}^{k} s_c \log\left(\frac{s_c}{s}\right)\right)\left(\sum_{q=1}^{r} s_q \log\left(\frac{s_q}{s}\right)\right)}} \tag{5.1}$$

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

5.2.1.2 Adjusted Rand Index(ARI)

The ARI[31] is used to determine the similarity between two datasets' clustering. The fuzzy divisions of the maximum component in each column are set to 1 and all others to 0 to calculate the ARI. We use ARI to evaluate the proposed Kernelized Fuzzy clustering algorithms to the KSRSIO-FCM findings and to contrast the clustering arrangements and ground-truth labels[13]. The ARI is characterized as follows:

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

(5.2)

5.2.1.3 F-Score

F-score used to calculate the accuracy of a clustering output [32]. The precision and recall of the cluster for each given class is computed as follows:

$$p_{qc} = \frac{s_{qc}}{s_c}, r_{qc} = \frac{s_{qc}}{s_q} \tag{5.3}$$

Where, s_{qc} denotes the number of samples of class q that are also present in cluster c, s_q denotes the number of samples belonging to class q, and s_c denote the number of samples belongs to cluster c. The F-score of cluster c and class q is represented as follows:

$$f(q,c) = \frac{2 * p_{qc} * r_{qc}}{p_{qc} + r_{qc}} \tag{5.4}$$

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:

$$F - score = \sum_q \frac{s_q}{s} max\{f(q,c)\} \tag{5.5}$$

where, s is the total number of data samples. The higher value of the F-score indicates better clustering results. An F-score value approaching 1 reflects that the attained clustering results are similar to the ground truth value.[13]

5.2.2 Internal Performance Measures

We used two internal performance measures. They are:

5.2.2.1 Silhouette Index(SI)

The Silhouette Index[33] indicates the clustering efficiency by calculating the difference between the average distance within the cluster and the minimum distance between the clusters, and using them in the equation below:

$$SI = \frac{\sum_{i=1}^{s}\left(\frac{b(i)-a(i)}{maxb(i),a(i)}\right)}{s} \tag{5.6}$$

among them: a(i) represents the average distance of sample i to other samples in the cluster, b(i) represents the minimum distance of the sample from the sample i to the other clusters. The range of SI is [-1,1] and the higher the SI is, the better is the clustering algorithm.

5.2.2.2 Davies Bouldin Index(DBI)

David L. Davies and Donald W. Bouldin created a metric for analysing clusters called the Davies-Bouldin Index (DBI), which they named after themselves. The Davies-Bouldin Index contains an internal cluster evaluation method, where the quantity and proximity of cluster results determine whether the results are satisfactory or not. The Davies-Bouldin Index is one way for determining cluster validity in a grouping method. Cohesion is defined as the sum of the data's proximity to the cluster centre point. The distance between the cluster centre points and the cluster determines the separation. This index seeks to reduce the distance between points in a cluster while maximising the inter-cluster distance between Ci and Cj clusters. When the inter-cluster distance is maximum, the similarity of characteristics between clusters is low, allowing distinctions between clusters to be seen more clearly. Each object in the cluster has a high level of characteristic similarity if the minimum intra-cluster distance is low[34]. The lesser the value of DBI is, the better is the clustering.

5.3 Results

In this subsection, we will be discussing the results of various clustering algorithms on various datasets(benchmark and SNP datasets).

5.3.1 Results on Benchmark Datasets

Here, we will be comparing the NMI, ARI, and F-Score results of CKSRSIO-FCM, LKSRSIO-FCM and KSRSIO-FCM on benchmark datasets by dividing the datasets into a number of chunks.

5.3.1.1 Replicated Avila Dataset

Here, we divided the Avila dataset into 5,10,20,40 subsets and performed CKSRSIO-FCM in(Table 5.1), LKSRSIO-FCM in(Table 5.2) and KSRSIO-FCM in(Table 5.3) on each number of subsets. In the above 3 tables, we can see that in most of the cases, our

TABLE 5.1: Results of CKSRSIO-FCM algorithm on Avila Dataset after dividing the dataset into **Subsets** number of subsets

Subsets	**NMI**	**ARI**	**F-score**
5	0.0603333229	0.01146382045	0.1413236210
10	0.0584450636	0.01261832012	0.0741841185
20	0.0590992878	0.01236370186	0.0517084392
40	0.0588603369	0.01424574721	0.0788326065

TABLE 5.2: Results of LKSRSIO-FCM algorithm on Avila Dataset after dividing the dataset into **Subsets** number of subsets.

Subsets	**NMI**	**ARI**	**F-score**
5	0.116824029	0.037394917	0.08559379
10	0.07209577	0.017196797	0.12101026
20	0.07170751	0.016607399	0.10931659
40	0.11812616	0.036929863	0.06834084

TABLE 5.3: Results of KSRSIO-FCM algorithm on Avila Dataset after dividing the dataset into **Subsets** number of subsets.

Subsets	**NMI**	**ARI**	**F-score**
5	0.11687752	0.034971409	0.08602099
10	0.0690528	0.01183567084	0.111372022
20	0.06439268	0.0108595134	0.1273781569
40	0.11599797	0.0332438788	0.074327886136

Proposed LKSRSIO-FCM gives the best output.

5.3.1.2 Replicated Wine dataset

Here we divided the Wine dataset into 5,10,40,100 subsets and performed CKSRSIO-FCM in(Table 5.4), LKSRSIO-FCM in(Table 5.5) and KSRSIO-FCM in(Table 5.6) on each number of subsets. As you can see in the tables, in most of the cases, LKSRSIO-FCM gives better results, compared to other clustering algorithms.

TABLE 5.4: Results of CKSRSIO-FCM algorithm on Wine Dataset after dividing the dataset into **Subsets** number of subsets.

Subsets	NMI	ARI	F-score
5	0.4143879799	0.4103407526	0.0159523113
10	0.4144380082	0.4102990977	0.0698262179
40	0.4357136025	0.3636449239	0.3318475122
100	0.414713556	0.4109801849	0.3994680539

TABLE 5.5: Results of LKSRSIO-FCM algorithm on Wine Dataset after dividing the dataset into **Subsets** number of subsets.

Subsets	NMI	ARI	F-score
5	0.4181896644	0.37477401902	0.0741587611
10	0.4175601724	0.37406532136	0.0741254723
40	0.4174877930	0.37387290590	0.3936790275
100	0.417899652	0.37467090699	0.3937905582

TABLE 5.6: Results of KSRSIO-FCM algorithm on Wine Dataset after dividing the dataset into **Subsets** number of subsets.

Subsets	NMI	ARI	F-score
5	0.3412314369	0.274570767	0.08425009
10	0.41984259	0.3499788	0.1686616881
40	0.4148323095	0.41091484	0.0224701089
100	0.4148550148	0.4112008024	0.5564164593

5.3.1.3 Replicated Skin-Monarch dataset

Here we divided the Skin-monarch dataset into 40,100 subsets and performed CKSRSIO-FCM in(Table 5.7), LKSRSIO-FCM in(Table 5.8) and KSRSIO-FCM in(Table 5.9) on each number of subsets.

As you can see in the tables, we are getting the best results for LKSRSIO-FCM, compared to other clustering algorithms.

5.3.2 Results on SNP data

Here, we compared the performances of GPU accelerated 13d-SInteger and CPU enabled 13d-SInteger on SNP dataset and RNA dataset. We also validated the superiority of our proposed 13d-SZcurve Scalable Feature extraction technique and proposed LKSRSIO-FCM algorithm.

5.3.2.1 Performance comparison of GPU accelerated 13d-SInteger with CPU enabled 13d-SInteger

We used 5.1.3.3 dataset and a reference RNA dataset to examine how efficient our proposed GPU accelerated 13d-SInteger is.

TABLE 5.7: Results of Skin-Monarch Dataset applied on CKSRSIO-FCM algorithm.

Subsets	NMI	ARI	F-score
40	0.0306441681	0.0120130278	0.5602171692
100	0.001885462	0.0052590431	0.5394063174

TABLE 5.8: Results of Skin-Monarch Dataset applied on LKSRSIO-FCM algorithm.

Subsets	NMI	ARI	F-score
40	0.0546113725	0.02675413217	0.5856698299
100	0.555493138	0.66653842456	0.9082192021

TABLE 5.9: Results of Skin-Monarch Dataset applied on KSRSIO-FCM algorithm.

Subsets	NMI	ARI	F-score
40	0.033156369	0.01279330	0.562064421
100	4.55628e-06	0.0001182	0.487160705

TABLE 5.10: Results of time taken by GPU accelerated 13dS-Integer Feature extraction technique and CPU enabled 13dS-Integer Feature extraction technique on RNA dataset and 5.1.3.3 dataset

Genome Data	CPU time	GPU time
SNP	1:17:00	**0:54:00**
RNA	**1:01**	4:03

As we can see in the table 5.10, in the case of SNP data our GPU accelerated technique performed well, whereas, in the case of RNA dataset, the reverse is true. This is due to the nature of the dataset. SNP dataset has large sequences whereas, RNA dataset has short sequences. Using GPUs has some overheads associated with it. And in the case of large sequences, those overheads are overpowered by the positive impact of GPU parallelization whereas, in the case of shorter sequences the overheads dominate the performance. So, it results in performance degradation.

5.3.2.2 Validation of superiority of 13dS-Zcurve Feature Extraction Technique

We used 5.1.3.1 dataset to validate the superiority of 13dS-Zcurve. To validate this, we used another feature extraction technique which returns a 12 dimensional feature vector[35] and extracted two different sets of feature vectors obtained from these two feature extraction techniques and applied LKSRSIO-FCM algorithm on both sets of feature vectors and evaluated and compared the clustering using the internal performance measures5.2.2. Results pertaining to 13dS-Zcurve are presented in 5.11 and results pertaining to 12 dimensional Feature extraction[35] are presented in 5.12. Since the 5.1.3.1 dataset is unlabelled, we have the freedom to choose the number of clusters for clustering. In 5.11 and 5.12, the **Number of Clusters** column indicates the number of cluster centers set for LKSRSIO-FCM algorithm to cluster

As we can see 5.11 and 5.12, SI values are drastically better in the case of 13dS-Zcurve technique for all number of clusters ,whereas DBI values are also comparably lower than 12 dimensional technique in most of the number of clusters and absolutely speaking, since the values of DBI are very low and values of SI are very high, 13dS-Zcurve feature extraction technique is superior to other feature extraction techniques.

TABLE 5.11: Results of 13dS-Zcurve Feature Extraction technique and LKSRSIO-FCM clustering algorithm on Wm82.a1 dataset 5.1.3.1

Number of Clusters	DBI	SI
5	0.9305597102113623	0.9532872836318572
10	0.8913727135087518	0.928289926351393
15	0.8728390132216509	0.9064760999217982
20	0.8700481842851564	0.8148608044886282
25	0.8598892193804883	0.7958223204260552
30	0.8495176165865425	0.7232044277314903
35	0.8447098346501675	0.690046914142936

TABLE 5.12: Results of 12 dimensional Feature Extraction technique and LKSRSIO-FCM clustering algorithm on Wm82.a1 dataset 5.1.3.1

Number of Clusters	DBI	SI
5	0.7158779701400338	0.38826054121633585
10	0.881818655434597	0.30587565872030514
15	1.0922814128352203	0.2268066000766351
20	1.2857229478158436	0.188748255465775
25	1.3548787569244078	0.1775701248189999
30	1.4247755779180309	0.16621180429185317
35	1.5515900932443547	0.14768876413693718

5.3.2.3 Validation of superiority of LKSRSIO-FCM clustering algorithm

To prove the superiority of the proposed LKSRSIO-FCM algorithm, we applied the proposed 13dS-Zcurve feature extraction technique and applied two clustering algorithms, i.e. LKSRSIO-FCM and KSRSIO-FCM on this set of feature vectors and evaluated and compared the efficiencies of clustering by using internal performance measures5.2.2 and the values are presented in tables 5.13 and 5.14 respectively. As it is evident, LKSRSIO-FCM is giving the better output, when compared to KSRSIO-FCM.

5.4 Creation of Real Life Dataset

In the previous section, after investigating the best feature extraction technique and best Scalable Kernelized Fuzzy clustering algorithm, we used them in our genome analysis, which is the main objective of the project. In this section, we discuss the way we generated the novel dataset.

TABLE 5.13: Results of 13dS-Zcurve Feature Extraction technique and LKSRSIO-FCM clustering algorithm on Wm82.a1 dataset 5.1.3.1

Number of Clusters	DBI	SI
5	0.9305597102113623	0.9532872836318572
10	0.8913727135087518	0.928289926351393

TABLE 5.14: Results of 13dS-Zcurve Feature Extraction technique and KSRSIO-FCM clustering algorithm on Wm82.a1 dataset 5.1.3.1

Number of Clusters	DBI	SI
5	0.94563	0.74876
10	4.31	0.928289926351393

5.4.1 Cat sequence description

Initially, raw SNP test sequence of approximate size 24 GB, is created by ICAR-IISR Indore, and it was sent for sequencing to **Novogene** and after sequencing, the final sequenced test sequence has 10 million characters. The cat sequence data has a column named **Pos**. This column describes the position of the corresponding character in the sequence. This column is essential for the next step.

5.4.2 Concatenation of sequenced test sequence to Wm82.a2 dataset[5.1.3.2]

Even the Wm82.a2 dataset has a **Pos** column. Based on value of Pos columns in the test sequence and Wm82.a2 dataset, the intersection of these 2 data is computed and a novel dataset named **Wm82.a3** is generated. The length of sequences in the novel dataset is 2210.

5.5 Gene identification

In the previous section, we generated the novel **Wm82.a3** dataset and in this section, we discuss the way we used this dataset to perform gene identification of the given sequenced testing gene.

5.5.1 Performing clustering on the new dataset

As discussed in the previous sections, we used GPU accelerated 13d-SZcurve Feature extraction technique and LKSRSIO-FCM Fuzzy clustering algorithm to perform clustering.

5.5.2 Finding closely related genes

After clustering the novel dataset, we found out the cluster in which the testing gene fell into. In that cluster, on the basis of Euclidean distance, we found out the top 50 genes that were geometrically closely related to the testing gene. And we stored the gene ids of those genes in a seperate file. The 50 genes look like Figure 5.1

5.5.3 Genetic trait identification

The genetic traits of the genes present in Figure 5.1 are identified by ICAR-IISR Indore and if a considerable number of genes fall under a particular genetic trait bucket, then it is estimated that the testing gene is also likely to possess that particular genetic trait. In the current scenario only top 10 out of 50 genes were analyzed and among them, the first 5 closely related genes whose IDs are **PI567429D, PI506656, PI54855, PI567299A, PI603381B**, are found out to be **strongly disease resistant**. This means that the probability of the given testing gene, being strongly disease resistant is quite high. So ICAR-IISR Indore, concluded that the given testing gene is more likely strongly disease resistant as well.

Gene code	Euclidean distance with the reference gene
PI567429D	124384948349624.36
PI506656	136603514189268.67
PI54855	234276127116810.94
PI567299A	376986163083275.4
PI603381B	397066056962441.25
PI603295	579025463755652.0
PI437609B	655745675394040.0
PI307873D	1007374515269610.9
PI603150	1319670331588050.0
PI614808	1464185811653377.0
PI507460	1484493242688253.5
PI437770	1690860680976578.0
PI91119	1749919725931473.8
PI587682C	1840909840598926.8
PI606748	1954725748120527.8
PI592907C	2101226185831679.5
PI588015A	2207855418996223.0
PI200592	2288669916860528.0
PI561323	2315658816419719.0
PI567352A	2375583922781580.0
PI404195	2433099724264051.5
PI68778	2502514978182123.0
PI79761	2518899103844247.5
PI291293A	2518899103844247.5
PI79848-1	2518899103844247.5
PI79760	2518899103844247.5
PI96118	2597723600784678.0
PI54859	2600490114042825.0
PI458086	2667126762902665.5
PI464898	2952660635645395.0
PI437270A	2995928703707700.0
PI437413	3010288342594227.5
HS5N3417A	3223562817465927.5
PI424613	3258793548134594.5
PI408311-1	3459979196957696.0
PI399058	3496910113718145.0
PI612707B	3538734524586720.5
PI592937	3563166187156446.0
PI497956	3633579032251940.5
PI587716B	3817790420705033.5
PI632961	3911879709608895.5
PI384473	3965953573580187.5
PI408251	3993926661719166.5
PI458024A	4010341619500374.5
PI603692	4205564273282231.0
PI567166	4351151300159086.5
PI86153	4622756332074112.0
PI88805-4	4698035385414646.0
PI628920	4783841974838727.0
PI417096	5340133601098831.0

FIGURE 5.1: The list of top 50 closely related genes with the test sequence, accompanied with their euclidean distance with the same

Chapter 6

Conclusion and Future Work

In this thesis, we contributed the design and implementation of CPU Enabled Scalable Feature Extraction techniques and GPU accelerated Scalable Feature Extraction techniques. Finding the right Feature Extraction technique was crucial and challenging and then making the algorithm scalable by using Apache Spark framework was another challenge. We also validated the superiority of our proposed Scalable Feature Extraction techniques by comparing the results with another Feature Extraction technique which returns a 12-dimensional feature vector.

In this thesis, we also contributed the design and implementation of a novel scalable kernelized fuzzy clustering algorithms. This algorithm partitioned the Big Data into various chunks, and processed the data points present within the chunk in a parallel manner. One distinctive characteristic of the scalable kernelized fuzzy clustering algorithm is that due to the parallel processing of data points within the chunk, it achieves a significant reduction in run-time for the clustering of such huge amounts of data, without compromising the quality of clustering. 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, it makes the execution of the proposed algorithm much faster. The scalable online fuzzy clustering algorithms are implemented using the Big Data processing framework called Apache Spark to deal with the challenges associated with fuzzy clustering for handling Big Data. For comparing the performance of two basic algorithms present in literature are enhanced by making these algorithms scalable for handling Big Data. The detailed experimentation is carried out and the reported results in the previous chapters, show the dominance of the proposed algorithms over other comparable approaches in terms of various measures assessing the quality of clustering. The results show that the proposed scalable kernelized fuzzy clustering algorithm has a great potential in Big Data clustering.

The proposed scalable kernelized fuzzy clustering algorithms is applied to a real-life Big Data problem (i.e., RNA of soybean genome) for gene identification of huge RNA/SNP sequences of real-life soybean plant. The productivity of next generation RNA sequences can be enhanced by making RNA sequences resistance to disease, drought, heat, high temperature, and food allergies. For this, a real-life Big Data is collected from the ICAR-IISR, Indore which is consists of millions of sequences. First of all, this SNP dataset is required to be preprocessed efficiently for its proper clustering and classification. For this reason, we have proposed a novel scalable feature extraction methods of RNA sequences. The other important characteristic is that it represents each variable length RNA sequence consisting of a long chain of nucleotide with a fixed-length numeric vector consist of only thirteen dimensions. The exhaustive experiments reported

shows the efficacy of the proposed scalable feature extraction methods of SNP/RNA sequences on proposed scalable clustering algorithms in terms of validity index.

Our work has received tokens of appreciation from **Dr. Milind Ratnaparkhe**, Principal Scientist (Biotechnology), Indian Institute of Soybean Research- Indore (IISR), owing to its impact. Our work on gene identification has helped them in identifying the genetic traits of the genome sequences they created, as part of their research.

This works suggest some interesting directions for future work. As this work presented the clustering of Big Data. So, it is interesting to conduct the investigation in finding the number of clusters for specific data set A very important question that arises is the validity of the clustering. To measure the quality of clustering, cluster validity index for Big Data are needed. Many cluster validity measures for small sized data require full access to the data points. Hence, we aim to extend some well-known cluster validity measures for use on Big Data by using similar extensions as presented here. Additionally, we can create massive novel real-life SNP datasets. and to apply our proposed algorithm for other species of genome datasets for identification of disease and draught in real-life soybean crop.

Bibliography

[1] Vernon Turner et al. "The digital universe of opportunities: Rich data and the increasing value of the internet of things". In: *IDC Analyze the Future* 16 (2014), pp. 13–19.

[2] William Yu Chung Wang and Yichuan Wang. *Analytics in the era of big data: the digital transformations and value creation in industrial marketing*. 2020.

[3] Doug Fisher et al. "Applying AI clustering to engineering tasks". In: *IEEE Expert* 8.6 (1993), pp. 51–60.

[4] Aldo Faisal et al. "Clustering of patient comorbidities within electronic medical records enables high-precision COVID-19 mortality prediction". In: (2021).

[5] Johanna Barzen and Frank Leymann. "Quantum Humanities: A First Use Case for Quantum-ML in Media". In: *Digitale Welt* 4 (2020), pp. 102–103.

[6] Yu Tian et al. "A data-driven clustering recommendation method for single-cell RNA-sequencing data". In: *Tsinghua Science and Technology* 26.5 (2021), pp. 772–789.

[7] Ge Wang, Pengbo Pu, and Tingyan Shen. "An efficient gene bigdata analysis using machine learning algorithms". In: *Multimedia Tools and Applications* 79.15 (2020), pp. 9847–9870.

[8] Claire Nédellec. "Machine Learning for Information Extraction in Genomics—State of the Art and Perspectives". In: *Text Mining and its Applications*. Springer, 2004, pp. 99–118.

[9] Razvan Bunescu et al. "Comparative experiments on learning information extractors for proteins and their interactions". In: *Artificial intelligence in medicine* 33.2 (2005), pp. 139–155.

[10] Goo Jun et al. "An efficient and scalable analysis framework for variant extraction and refinement from population-scale DNA sequence data". In: *Genome research* 25.6 (2015), pp. 918–925.

[11] Mehrdad J Gangeh, Hadi Zarkoob, and Ali Ghodsi. "Fast and scalable feature selection for gene expression data using hilbert-schmidt independence criterion". In: *IEEE/ACM transactions on computational biology and bioinformatics* 14.1 (2017), pp. 167–181.

[12] Meena Tushir and Smriti Srivastava. "Exploring different kernel functions for kernel-based clustering". In: *International Journal of Artificial Intelligence and Soft Computing* 5.3 (2016), pp. 177–193.

[13] Preeti Jha et al. "A Novel Scalable Kernelized Fuzzy Clustering Algorithms Based on In-Memory Computation for Handling Big Data". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* (2020).

[14] Cong Pian et al. "LncRNApred: Classification of long non-coding RNAs and protein-coding transcripts by the ensemble algorithm with a new hybrid feature". In: *PloS one* 11.5 (2016), e0154567.

[15] Siyu Han et al. "LncFinder: an integrated platform for long non-coding RNA identification utilizing sequence intrinsic composition, structural information and physicochemical property". In: *Briefings in bioinformatics* 20.6 (2019), pp. 2009–2027.

[16] Eric Augusto Ito et al. "BASiNET—BiologicAl Sequences NETwork: a case study on coding and non-coding RNAs identification". In: *Nucleic acids research* 46.16 (2018), e96–e96.

[17] Robson P Bonidia et al. "Feature extraction approaches for biological sequences: a comparative study of mathematical features". In: *Briefings in Bioinformatics* 22.5 (2021), bbab011.

[18] Gerardo Mendizabal-Ruiz et al. "On DNA numerical representations for genomic similarity computation". In: *PloS one* 12.3 (2017), e0173288.

[19] Ren Zhang and Chun-Ting Zhang. "Z curves, an intutive tool for visualizing and analyzing the DNA sequences". In: *Journal of Biomolecular Structure and Dynamics* 11.4 (1994), pp. 767–782.

[20] Mohammed Abo-Zahhad, Sabah M Ahmed, and Shimaa A Abd-Elrahman. "Genomic analysis and classification of exon and intron sequences using DNA numerical mapping techniques". In: *International Journal of Information Technology and Computer Science* 4.8 (2012), pp. 22–36.

[21] Paul Dan Cristea. "Conversion of nucleotides sequences into genomic signals". In: *Journal of cellular and molecular medicine* 6.2 (2002), pp. 279–303.

[22] James C Bezdek, Robert Ehrlich, and William Full. "FCM: The fuzzy c-means clustering algorithm". In: *Computers & Geosciences* 10.2-3 (1984), pp. 191–203.

[23] Tianhao Li et al. "Interval kernel Fuzzy C-Means clustering of incomplete data". In: *Neurocomputing* 237 (2017), pp. 316–331.

[24] Thomas M Cover. "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition". In: *IEEE transactions on electronic computers* 3 (1965), pp. 326–334.

[25] Weiling Cai, Songcan Chen, and Daoqiang Zhang. "Robust fuzzy relational classifier incorporating the soft class labels". In: *Pattern Recognition Letters* 28.16 (2007), pp. 2250–2263.

[26] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[27] https://cupy.dev/.

[28] Moshe Lichman et al. *UCI machine learning repository*. 2013.

[29] http://homepages.uni-paderborn.de/frahling/instances/monarch.txt.

[30] Alexander Strehl and Joydeep Ghosh. "Cluster ensembles—a knowledge reuse framework for combining multiple partitions". In: *Journal of machine learning research* 3.Dec (2002), pp. 583–617.

[31] 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". In: *Bioinformatics* 17.9 (2001), pp. 763–774.

[32] Hui Xiong, Junjie Wu, and Jian Chen. "K-means clustering versus validation measures: a data-distribution perspective". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39.2 (2008), pp. 318–331.

[33] Xu Wang and Yusheng Xu. "An improved index for clustering validation based on Silhouette index and Calinski-Harabasz index". In: *IOP Conference Series: Materials Science and Engineering* 569.5 (2019), p. 052024. DOI: 10.1088/1757-899x/569/5/052024. URL: https://doi.org/10.1088/1757-899x/569/5/052024.

[34] M Mughnyanti, S Efendi, and M Zarlis. "Analysis of determining centroid clustering x-means algorithm with davies-bouldin index evaluation". In: *IOP Conference Series: Materials Science and Engineering* 725.1 (2020), p. 012128. DOI: 10.1088/1757-899x/725/1/012128. URL: https://doi.org/10.1088/1757-899x/725/1/012128.

[35] Preeti Jha et al. "Apache Spark based kernelized fuzzy clustering framework for single nucleotide polymorphism sequence analysis". In: *Computational Biology and Chemistry* 92 (2021), p. 107454.