“ELECTROMAGNETSIM BASED K-MEANS CLUSTERING FOR BIG DATA"

A Thesis

by

ABHINAV EERLAPATI

BS, Jawaharlal Nehru Technological University, 2014

Submitted in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

Texas A&M University-Corpus Christi
Corpus Christi, Texas

Fall 2017

Fall 2017

"ELECTROMAGNETSIM BASED K-MEANS CLUSTERING FOR BIG DATA"

A Thesis

By

Abhinav Eerlapati

This thesis meets the standards for scope and quality of

Texas A&M University-Corpus Christi and is hereby approved.

Dr. Longzhuang Li

Chair

Dr. Junfei Xie

Committee Member

Dr. Alaa Sheta

Committee Member

Graduation December 2017

ABSTRACT

Over the past few years, Nature has been the source of inspiration for many proposed successful algorithms. This paper proposes a new nature-inspired K-means clustering algorithm which is based on the concept of Electromagnetism. The proposed algorithm starts by initializing a set of particles and later in the second step, the best particle among them is chosen based on the fitness function. After choosing the best particle, an objective function value is calculated for each particle which is initialized. Then the force and movement are calculated for each particle except for the current best particle. This way, the algorithm at each iteration searches for a local best particle and then calculates objective function values. Due to this reason, the position of the initialized particles also changes. Algorithm terminates when it reaches the maximum iterations or when the change in Within Set Sum of Squared Error (WSSSE) is less than 0.0001. The detailed explanation of this algorithm is presented. From the results, Electromagnetism based K-means provides better accuracy when compared to K-means clustering. This can be seen from the results section.

TABLE OF CONTENTS

CONTENTS PAGE

ABSTRACT v

TABLE OF CONTENTS vi

LIST OF FIGURES viii

LIST OF TABLES ix

CHAPTER I: INTRODUCTION AND MOTIVATION 1

1.1 Introduction 1

1.2 Motivation 1

CHAPTER II: LITERATURE REVIEW 3

2.1 Firefly Algorithm 4

2.2 Genetic Algorithm 7

2.3 K-Means Clustering and its Importance 9

CHAPTER III: ELECTROMAGNETISM BASED GLOBAL OPTIMIZATION 13

CHAPTER IV: PROPOSED SYSTEM 18

4.1 Initialization 18

4.2 Local Optimum Search 19

4.3 Charge and Force Calculation 19

4.4 Movement 20

CHAPTER V: SYSTEM IMPLEMENTATION 25

5.1 Apache Spark 25

CHAPTER VI: RESULTS AND EVALUATION 32

6.1 Results with Small Dataset 32

6.2 Results with Big Dataset 34

CHAPTER VII: CONCLUSION AND FUTURE WORK 37

Bibliography 38

APPENDIX A: STEPS TO IMPLEMENT THE ALGORITHM IN HPC CLUSTER 40

APPENDIX B: CODE FOR THE PROJECT 43

LIST OF FIGURES

FIGURES PAGE

2.1 Data Structure and Movement of Firefly 5
2.2 Clustering by Firefly Algorithm 6
2.3 Chromosomes 8
2.4 Six genes in a CR 8
2.5 Final Clusters formed after applying K-means Clustering 11
3.6 Superposition principle 16
4.7 Particles View 21
4.8 Randomly Picked points in each particles 22
4.9 Assigning data points 22
4.10 New Centroid Calculation 22
4.11 Best particle 23
4.12 Charge Calculation 23
4.13 New Population Generated 23
5.14 Flow Chart 27
6.15 Convergence Curve using the EMK Algorithm for Iris Data 35
6.16 Convergence Curve using the K-means Algorithm for Iris Data 36
8.17 Putty Software 41
8.18 Login Screen 41
8.19 Sample Slurm File 42

LIST OF TABLES

TABLES PAGE

6.1 List of Datasets and its information 32

6.2 PCA and Run Time values for small datasets 33

6.3 Values for Big datasets Using EMK and K-means 35

CHAPTER I: INTRODUCTION AND MOTIVATION

1.1 Introduction

Data Clustering is one of the important tasks in Pattern Recognition and Data Analysis. It is an unsupervised technique, which is used to classify data into groups. There are mainly two types of clustering methodologies: hierarchical and partitional clustering. In hierarchical, the large dataset is broken down into smaller groups, and later the smaller groups are merged into their near centroid. In case of the partitional clustering, data is partitioned into smaller clusters, each holding a center. Based on the distance from the center, the data gets assigned to a cluster.

K-means clustering algorithm is one among the simple and efficient partitional clustering techniques used in solving real-world problems [15] [7] . The K-means method attempts to classify the given data set into K clusters, or groups. In the K-means method, each data is repeatedly assigned to a suitable cluster by calculating the distance between the data and the center of the cluster. Due to the lack of efficiency in K-means algorithm like the initialization, indefinite movement of cluster centers in each iteration etc., This makes it not suitable for using it to cluster big data. The goal of this research is to improve the K-means Clustering technique when applied to Big data sets. This is achieved by using the Evolutionary computation techniques for improving K-means. The Nature inspired local optimum finding technique called Electromagnetism is used for this purpose. In this paper, we propose a new algorithm, which makes use of this nature-inspired technique, and cluster technology to improve the K-means on Big data.

1.2 Motivation

Data Analysis and Machine Learning are some new fields, which have caught the attention of the world. Seeing its power, many IT firms and researches have started working in these areas to make their lives easier. Data Analysis helps us to predict the future trends in data, which can help

in making needed improvements or precautions to business. While machine learning is put into use for facial recognition, credit card fraud detection, building autonomous cars etc. In all these cases, computers are learning things from the data available instead of explicitly being programmed for a given task.

In all these areas of study, there is one common step which is very much needed to be performed before we start these actual algorithms called Data Clustering.

CHAPTER II: LITERATURE REVIEW

There are many known clustering techniques and all of them can be broadly classified into the following four main classes.

- Connectivity Based Clustering

  Connectivity based clustering is a kind of hierarchical clustering. The main idea is to build a binary tree of the data that successively merges similar groups of points. These algorithms connect objects to form clusters based on their distance's. They do not provide a single partition, instead they provide an extensive hierarchy of clusters that merge with each other at certain distance. In these algorithms, we do not need a fixed count of clusters to be given by the user, but it requires a termination condition. Some of the well-known algorithms that fall under this category are Diana, Agnes, BIRCH, and CAMELEON [19].

- Centroid Based Clustering

  In this method, we need to provide an input saying the number of clusters needed say K. In this approach, a database D that contains n objects is partitioned into K clusters, such that the sum of squared distances is minimized. The optimization problem here is known to be a NP-Hard problem. The algorithms, which fall under this category, are K-means, K-medoids, and CLARANS [14].

- Density Based Clustering

  Here, clustering is performed based on connectivity and density based functions. The most popular algorithm that falls under this category is the DBSCAN that is based on connecting points within certain distance thresholds. However, it only connects points that satisfy a density criterion, which is defined as a minimum number of objects, which belong to another cluster present within its radius. Finally, a cluster in this method will consist of all density-connected objects (which can form a cluster of an arbitrary shape, in contrast to many other

methods) plus all objects that are within its objects range. The other algorithms that fall under this category are OPTICS [2] and Denclue [19].

- Distribution Based Clustering

  One of the prominent models, which falls under this category, is the Gaussian mixture model. Here the dataset is modelled with a fixed number of Gaussian distributions [19] that are initialized randomly and whose parameters are iteratively optimized to fit better to the data set. This will converge to a local optimum, so multiple runs may produce different results. To obtain hard clusters, objects are often assigned to the Gaussian distribution they most likely belong to. While in case of soft clustering, this is not necessary.

One of the most prominently used algorithms among all the above listed categories is the K-means Clustering. Seeing some of the problems in K-means clustering algorithm, many nature-inspired algorithms which have become popular are being used for improving K-means. Some of the important algorithms of this kind are Particle Swarm Optimization (PSO), inspired from the swarm behavior like the fish and bird schooling in nature [15]. The working theory of this algorithm is based on two terms called particle (solutions) and swarm (population). The particles will be moving around in the solution space by simultaneously adjusting their position and velocity. They also exchange information about their current position with its neighbors in the search space as prior of their own earlier experience. In groups, they travel in search of food or shelter, without any collision among themselves. After communicating with their group, the particles modify their positions and velocities as per the best position appeared in the current movement. Thus, the particles would gradually get closer to the specified position and finally reach the optimal position with the help of interactive cooperation.

2.1 Firefly Algorithm

The firefly algorithm (FA) [20] is a population-based optimization inspired by the flashing behavior of the fireflies in nature where the attractiveness of a firefly is proportional to its brightness.

Fireflies that are less bright travel towards the brighter ones. The movement mainly consists of three components: the current position, attraction to another firefly and a random walk element. Some of the characteristics assumed in this algorithm are:

- Each particle present in the state space corresponds to a firefly within the problem state space. The fitness, or evaluation value, of the particle relates to the light intensity or attractiveness.
- The light intensity and distance are inversely proportional to each other.
- The moving direction of each firefly is stochastically fixed to be attracted by brighter flash-light, which is produced by neighboring fireflies.
- Fireflies would be constantly moving in random directions if there were no attractive fireflies present around their neighborhood.

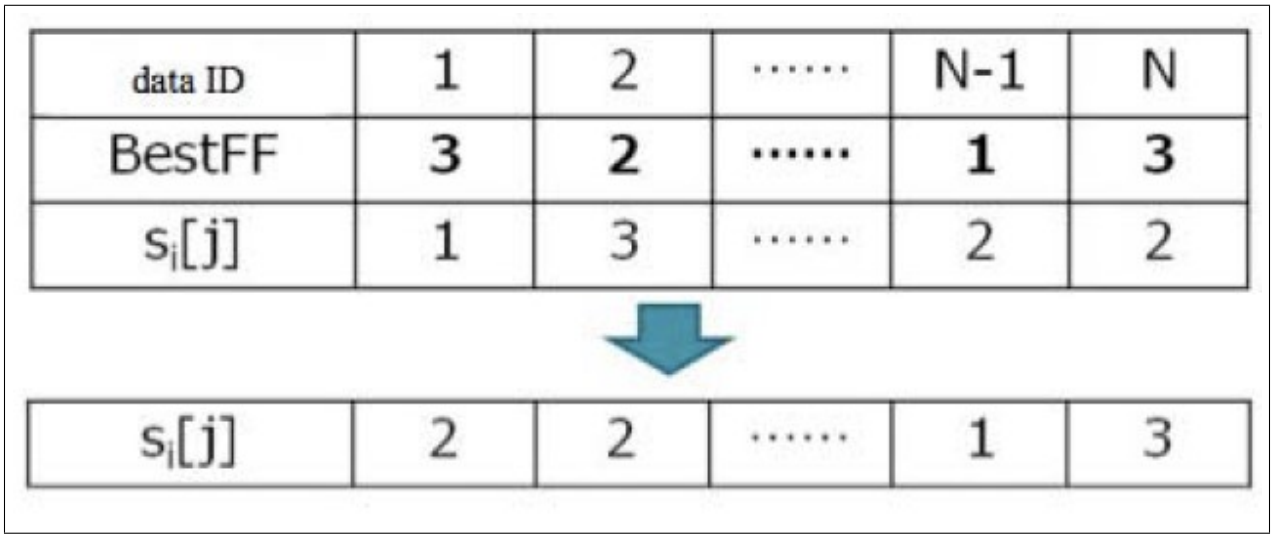

Figure 2.1: Data Structure and Movement of Firefly

Recently, two algorithms [12] [8] were proposed on K-means which are based on the firefly algorithm to resolve the data clustering problems. Both the algorithms consist of two stages:

- A quasi-optimal solution is sought by the FA algorithm.
- The quasi-optimal solution is then used as the initial cluster centers for the K-means methods.

The two algorithms differ in the data features stored in the fireflies. The value of each firefly in the FAK-A [8] algorithm are the locations of K clusters, while the values of each firefly in the FAK-B [12] algorithm are the cluster number of each data item. In other words, the FAK-A algorithm moves the cluster centers towards quasi-best locations and the FAK-B algorithm reassigns the data to quasi-optimal cluster numbers iteratively.

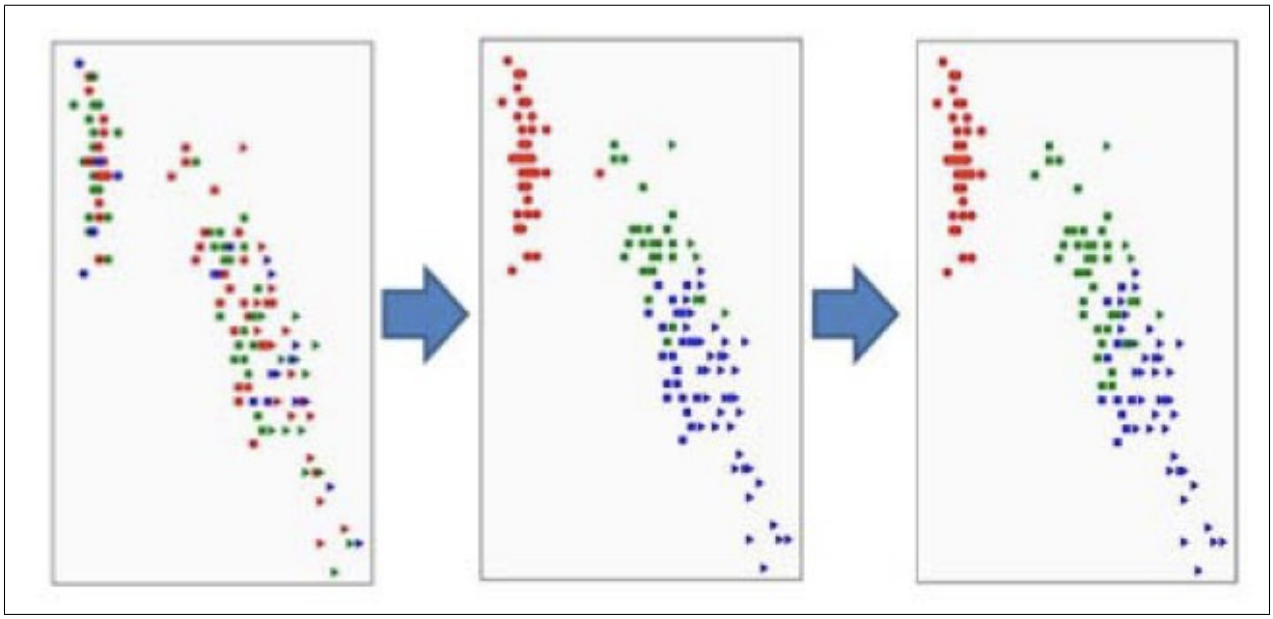

Figure 2.2: Clustering by Firefly Algorithm

In the standard FA algorithm, the brighter firefly exerts its influence over other fireflies and attracts them towards itself in each iteration. Similarly, in both FAK-A and FAK-B algorithms, the fireflies with the best solution in each iteration can attract and influence other firefly's movement and do not consider the combined influences of other fireflies in the current population. There is a lack of proof, for stating that the FAK algorithms can escape from the local optima.

Based on this concept, the author has built an algorithm where in the first phase they assign random cluster numbers to each data and run the firefly algorithm. Later, they apply the K-means clustering on to the solution obtained to form the final clusters [12]. This infers that the firefly algorithm has mainly tried to improve the initial step of the K-means clustering.

2.2 Genetic Algorithm

Another Nature inspired algorithm for K-means is the K-means clustering through Genetic Algorithm [17]. Using this algorithm, one can determine the number of clusters for the given data set but for this, the right choice of genes must be given as the initial clusters. This algorithm starts by selecting some of the chromosomes from the initial data point's population for capturing the clusters of different size and shape. Many of the clustering algorithms based on GAs suffer from degeneracy. According to Radcliffe et al. [16], degeneracy occurs when multiple chromosomes represent the same solution. Degeneracy can lead to inefficient coverage of the search space as the same configurations of clusters are repeatedly explored [5]. For eradicating the problems in GA based clustering, the following algorithms were proposed: Automatic Genetic Clustering for Unknown K (AGCUK) [10] and Genetic Algorithm with Gene Rearrangement (GAGR) [5]. They use the same kind of approach, but the initialization of the algorithm is done by the selection of points to be as genes. Here, they propose a new approach by selecting the initial genes deterministically and randomly. This is expected to be more exploratory when compared to the other two approaches. They also happen to propose a new fitness function and a cluster evaluation equation. The Proposed Technique is explained below

- The initial set of data points are broken down to chromosomes and these chromosomes consist of genes.
- For the selection of a chromosome, they initially assume a radius function which helps in calculating the density of the data set.
- The upper limit of the radius is assumed lesser than 0.5 and ranges from 0 to upper limit. This radius will be varying for each dataset.
- This way with help of the radius function genes are formed. The number of genes is based on the best value of the radius.

- Later, accurate formation of genes will be taken to be as the clusters. To reach this step, several gene rearrangements are to be made.

- Finally, some of the chromosomes are also combined based on the distance to form one chromosome and hence the final clusters are formed.

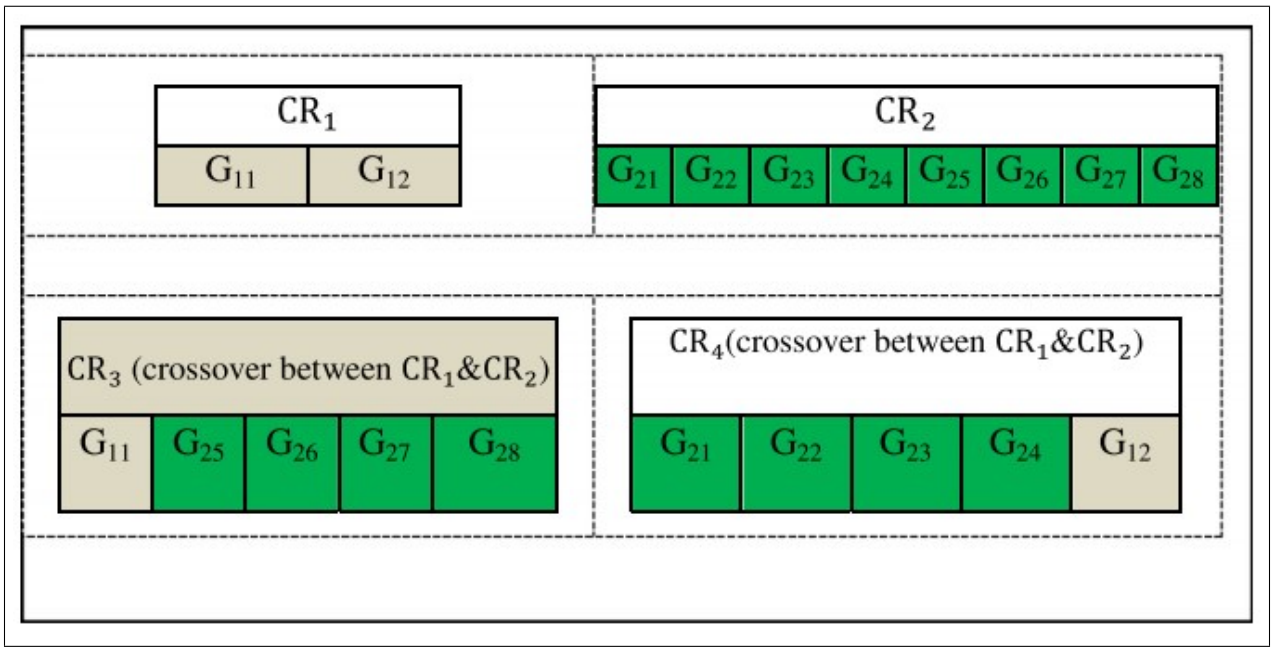

Figure 2.3: Chromosomes

The below picture shows scenario where in there are six different genes present in a single chromosome.

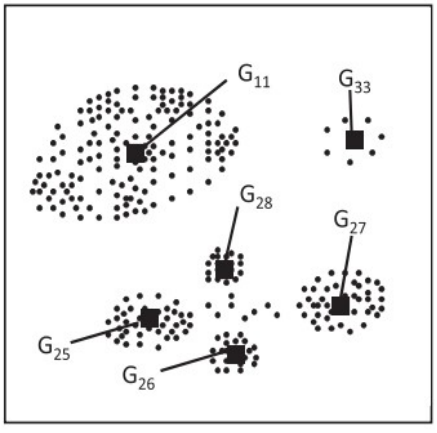

Figure 2.4: Six genes in a CR

The ant colony optimization was inspired from the behavior of a real ant colony, which derives from ants being able to find the nearest distance between its nest and food source [7]. The honeybee mating optimization (HBMO) algorithm was inspired by the process of marriage in real honeybees [11]. The Bat Algorithm (BA) was inspired by the echolocation behavior of bats [21]. The simulated annealing (SA) algorithm was developed by modeling the steel annealing process [9].These are some of the many algorithms proposed in recent days.

There were also some hybrid algorithms introduced based on the K-means. By hybridizing ant colony optimization, particle swarm optimization, K-means was developed as APSO-K-means clustering algorithm for speaker recognition. An improved PSO-based K-means algorithm was developed by Xiangwei and Yuanjiang [1] to avoid the local optima problem in normal K-means clustering.

The Efficient initialization of centroids plays a key role for improving the performance of the K-means [4]. This, as a result, would reduce the complexity and the number of iterations to reach and form the final clusters. There were many methods proposed to solve this and improve the accuracy of the algorithm. All these methods are broadly classified into the following three categories

- Linear Time-Complexity Initialization Methods
- Loglinear Time-Complexity Initialization Methods
- Quadratic Complexity Initialization Methods

Clustering is a very prominent and important method in various fields. The applications of various nature-inspired algorithms have also increased and these algorithms are being used to solve the challenges in various fields like engineering, computer science, computer vision, industry, data mining etc.

2.3 K-Means Clustering and its Importance

As said above, Clustering is the method of grouping similar objects so that they form a group. The K-means clustering technique falls under the NP-Hard Problem. The step by step process of

performing K-means is explained below.

If there are N points, which are starting from $g_1, g_2, \ldots, g_N$. The $\mathbb{R}^{\times}$ space is partitioned into K (the value of K is given in advance) sets and $G_1, G_2, \ldots, G_K$ as per their mutual similarity, which satisfies the following.

$$\begin{cases} G_i \neq \phi \quad \text{for } i = 1, 2, \ldots, K \\ G_i \cap G_j = \phi \quad \text{for } i, j = 1, 2, \ldots, K; i \neq j \\ \cup_{i=1}^{K} G_i = \{g_1, g_2, \ldots, g_N\} \end{cases}$$

Algorithm 1 Stepwise K-Means Clustering

1: Randomly choose K points from the N data points $g_1, g_2, \ldots, g_N$ as the initial cluster centers.
2: Assign each point to the cluster G_k, $(k = 1, 2, \ldots, K)$ where the point has the shortest distance to the cluster's centroid C_k, $(k = 1, 2, \ldots, K)$.
3: Calculate the mean value avg_k of the points for each cluster G_k, and use avg_k to update the cluster centroid C_k.
4: Repeat Steps 2 and 3 until the centroids no longer change.

The original K means was proposed in 1967. Since then, many new algorithms were proposed for improving K-means.

One of the different problems is the initial assignment of data points as centroids [18]. The right choice of centroid can reduce the computation and would help in improving the accuracy of the clustering. Final output for K-means can be seen from Figure 2.5, which is present below. In this image, there are 6 different clusters formed. All these clusters are being represented using 6 different colors. The centroids are highlighted using the symbol "+" which are being shown as a part of the data points.

All the above listed evolutionary based algorithms were successfully able to improve K-means but there are still some scope for improvements in them. In case of the K-means clustering algorithm, the data movement between the centers is very high and the initialization process must be improved as it does not cover the whole data space while picking the initial centers. In Firefly algorithm, both the algorithms only improved the initialization process of K-means but there is a problem of space complexity.When it comes to Genetic algorithm, the operational parameters are

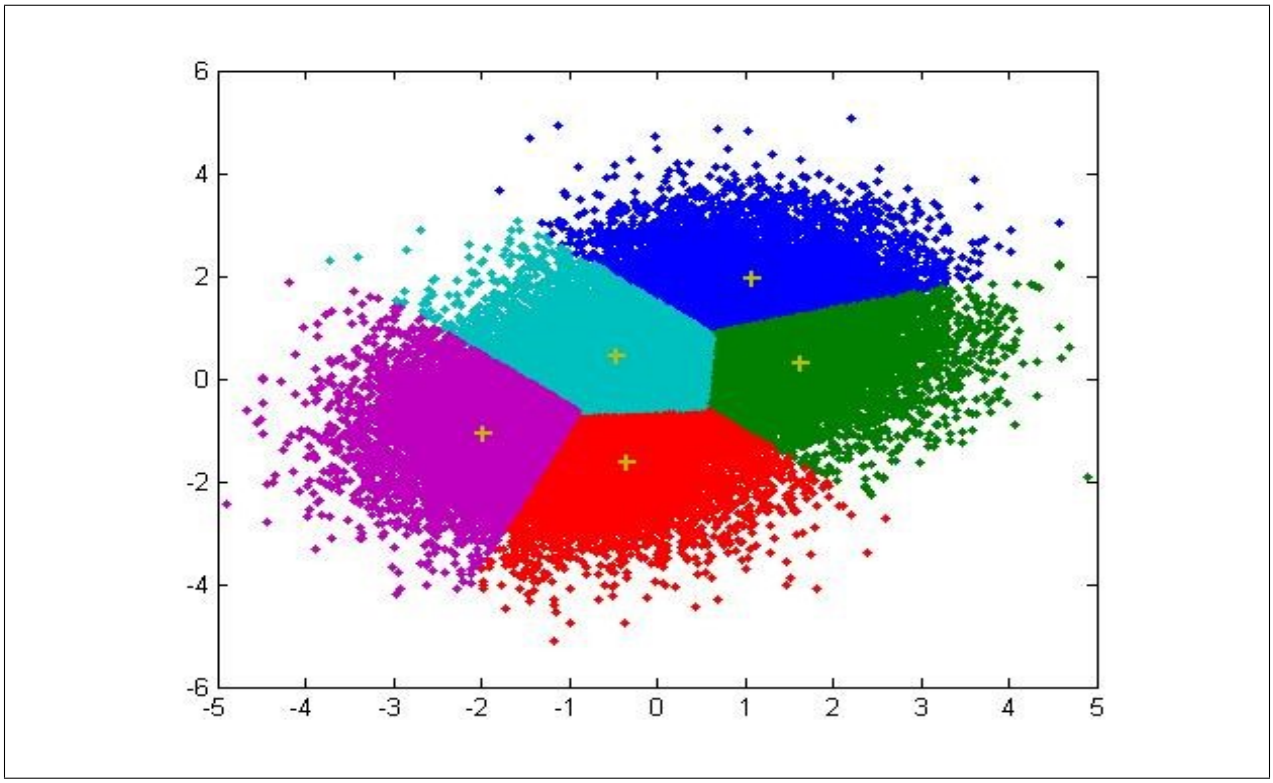

Figure 2.5: Final Clusters formed after applying K-means Clustering

complex which increases the computation cost. This would effect the whole performance of the algorithm. In case of the PSO, the convergence rate is slow.

Map-Reduce Frame Work:

Clustering Big-Data cannot be done directly. We need to reduce the dimensionality for processing. For this purpose, we make use of the Map-Reduce Framework. There are two important steps in the Map-Reduce Framework, the Map () and Reduce () functions. Some of the basic steps, which are involved in the Map-Reduce can be seen below.

1. Map () Input is prepared
2. Map code () written by the user is put into execution at this stage
3. Map output is put to “Shuffle”, reducing the processors
4. “Reduce” code is now put into execution

5. Final output.

As mentioned above, the Map () and the Reduce () functions are user definable.

CHAPTER III: ELECTROMAGNETISM BASED GLOBAL OPTIMIZATION

Finding a Global Optimum solution is one of the important goals of the mathematical optimization. Many real-life problems and research problems in various areas like Computer vision, Chemistry and Biology involve many functions . They need an optimal solution for their problems, and they use mathematical tools to optimize, which can be difficult to use. To overcome these difficulties, some search algorithms were proposed like the random search algorithm, Electromagnetism based global optimization etc.

Electromagnetism based global optimization [3] [6] is inspired from the concept of electromagnetism which is nothing but the physical interaction between charged particles due to the electromagnetic forces present among them. Here, the charges of the particles play a key role, as the force is directly proportional to charge. Therefore, the charge calculation is the objective function for this algorithm. The better the value of the objective function, then the higher would be the magnitude of attraction.

After the charge calculation, direction of movement for each particle in every iteration is determined. Like the electromagnetic forces, even here the force is calculated by vectorially adding forces from all the other particles which are calculated separately. Later, the amount of distance travelled by each particle is calculated, and the direction of movement is based on the resultant force.

This algorithm was used for solving the optimization problems in various fields such as the engineering design, feature selection, vehicle routing problem etc. This algorithm mainly consists of the following four steps as the core part

- Initialization of the particles
- Local optimum search
- Calculating the forces exerted on each individual particle.

- Distance moved along the direction of the particle.

The above steps are repeated for a fixed number of times, and this count is generally provided as an input by the user. There were many Electromagnetism based algorithms like the Rocha's method of shrinking Population, Debels's Method, Yurtkuran's method for reducing Movement etc. Many hybrid algorithms also came up which were mainly focused on the Local Optimum search , Force calculation and Distance movement.

In the Initialization step, sampling of points from the dataset is performed. This can be a data with n-dimensions. Later, the objective function for each of these points is calculated based on the function f(x). The point, which has the best objective function, is stored as x_{best}.

The second step is the local optimum search. The purpose of this is to make the particle move towards its local optimum. Various algorithms were proposed in verge of improving this local optimum search, mainly to reduce the computation cost. The original Electromagnetism like algorithm uses random line search. This random line search requires two parameters "δ" and LSIter. This LSIter represents the number of iterations to find the local optimum. The local search can be limited to only the current best or to all particles. The algorithm for Local search optimization is given below.

Algorithm 2 Local Search

INPUT: x_p, LSIter, δ
OUTPUT: x_p

1: **for** $d := 1$ to D **do**
2: $r_d := \delta(u_d - l_d)$;
3: counter: = 0;
4: **while** $counter < LSIter$ **do**
5: t := x_p;
6: $\lambda := random \in (-1,1)$;
7: $t_d := t_d + \lambda r_d$;
8: **if** $f(t) < f(x_p)$ **then**
9: $x_p := t$;
10: $counter := LSIter - 1$;
11: **end if**
12: $counter := counter + 1$;
13: **end while**
14: **end for**
15: **return** x_p;

The local search method is shown in Algorithm 1, which requires three parameters: the particle x_p, the number of iteration *LSIter*, and the parameter δ. The improvement in x_p is sought dimension-by-dimension (lines 1-14). For a give dimension $d \in (1, \cdots, D)$, first the maximum feasible step length r_d is calculated as the product of δ and the range of dimension-d (i.e. $u_d l_d$) (line 2). The particle x_p is assigned into a temporary particle t to store the initial information (line 5). Next, a random number $\lambda \in (-1,1)$ is selected as a step length and the particle x_p is moved along the dimension-d (lines 6-7). If f (x_p) is reduced within *LSIter*, the particle x_p is replaced by t, and the neighborhood searching for the particle x_p ends (lines 8-11).

In the third step, force exerted on each particle is calculated. This calculation is based on the superposition principle of electromagnetism theory. According to this principle, the force exerted on a charged particle is directly proportional to the product of their individual charges [13] and is inversely proportional to the distance among the points. The charged particle will be influenced and hence will be under moving according to Coulomb's force produced by other particles. The charge for each particle can be calculated by using the fitness value mentioned below in Equation

for calculating the charge on each particle.

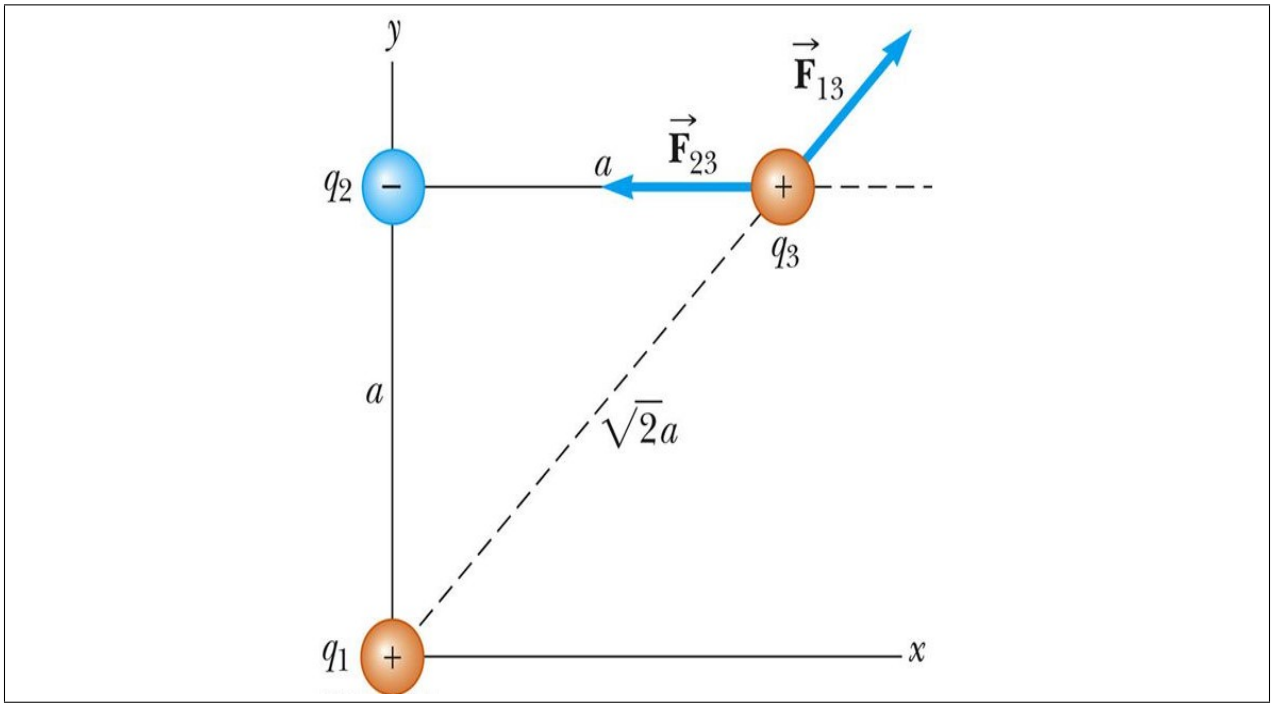

Figure 3.6: Superposition principle

$$q_p = exp(-D\frac{f(x^p) - f(x^{best}}{\sum_h = 1^m (f(x^h) - f(x^{best}}), \forall p \tag{3.1}$$

In the above equation, x_best is the particle with the smallest objective value in current population (i.e., $x_{best} \leftarrow argmin\{f(x_p), \forall p\}$), m is the number of particles, and D is the number of dimensions. The particles with smaller objective values have higher charges. Based on the calculation of charges and force, we determine the direction, which is the final force vector. This is evaluated for each particle by using the formula given below.

$$F^p = \sum_{h \neq p}^{m} \left\{ \begin{array}{ll} (x^h - x^p)\frac{q^p q^h}{||x^h - x^p||^2} & \text{if } f(x^h) < f(x^p) \\ (x^p - x^h)\frac{q^p q^h}{||x^h - x^p||^2} & \text{if } f(x^h) \geq f(x^p) \end{array} \right\}, \forall p \tag{3.2}$$

Here, $f(x_h) < f(x_p)$ represents that the particle x_p attracts the particle x_h, and $f(x_h) \geq f(x_p)$ represents that the particle x_p repulses the particle x_h. It is obvious that the particle x_{best} attracts all

other particles in the population because f (x_{best}) has the minimum value. The vector F_p should be normalized as follows:

$$\hat{F^p} = \frac{F^p}{||F^p||}, \forall p \tag{3.3}$$

Now to make the particle move in that direction we use the below mentioned formula in Equation 3.4. The movement is based on a random value, which is generated and named as. This random variable takes value between zero and one. The values u_d and l_d represent the lower and upper boundaries respectively for a d-dimensional data set.

$$x^p = \left\{ \begin{array}{ll} x^p + \lambda F^p (u_d - x_d^p) & \text{if } F^p > 0, \\ \\ x^p + \lambda F^p (x_d^p - l_d) & \text{if } F^p \leq 0, \end{array} \right\} \forall p \neq best \tag{3.4}$$

In the above equation, λ is a random step length that is uniformly distributed in (0,1), $\hat{F_d^p}$ is the d^th element of the force vector F^p, and u_d and l_d are the upper and lower limit for the d^th dimension, respectively. The resultant force $\hat{F_d^p}$ determines the movement of the particle x_p. The point x_p moves towards the upper limit if $\hat{F_d^p}$ is positive, or x_p moves toward the lowest boundary if $\hat{F_d^p}$ is zero or negative. The best particle is not moved because other particles in the population do not have enough force to attract or repel the best one.

The EM algorithm repeats the local search, force computation and movement steps until a maximum iteration number is reached or the value f (x_{best}) is small enough.

CHAPTER IV: PROPOSED SYSTEM

The Electromagnetism based K-means clustering is a new optimization method in the field of data clustering, which is inspired from the principle of Electromagnetism. The proposed EMK algorithm considers the combined forces from all the particles. In addition, the EMK algorithm constitutes both attraction and repulsion forces, which attract better solutions and repel worse ones. The FAK algorithms only have the attraction effect that brighter firefly attracts its neighbors. Furthermore, the K-means method is integrated into the EMK algorithm, while in the FAK algorithms, the K-means is executed as a separate step after the optimization procedures.

As mentioned above, the electromagnetism like algorithm was mainly used for solving the optimization problems in various fields. Now in this paper, we use the Electromagnetism concept for improving the K-means clustering. The four main steps used in this algorithm are Initialization, Local search, Force calculation and Movement. These are slightly modified for optimizing the K-means. The detailed explanation of this algorithm is present in this Chapter.

4.1 Initialization

At first, the K-means clustering algorithm starts by random selection of centroids. After the random selection of centroids, based on Euclidean distance these points are assigned to its nearest cluster. Similarly, in this Electromagnetism method m initial particles are generated with each one $x_p(p = 1, \cdots, m)$ holding K elements, which are the randomly picked cluster centers from the dataset, and they are represented as $x^{p_i} \quad (1 \leq i \leq K)$. Each particle element x^{p_i} is in the $\mathbb{R}^{\mathbb{D}}$ space. The objective function $f(x_p)$ is evaluated to determine the best particle x_{best} (where $x_{best} \leftarrow argmin\{f(x_p), \forall p\}$).

The objective function $f(x_p)$ is measured using the within cluster variation, which is the sum of squared error between the data points $g_j \in G_{x^{p_i}}$ and the centroid x^{p_i} for all the clusters, i.e.

$$f(x^p) = \sum_{i=1}^{K} \sum_{g_j \in G_{x^{p_i}}} (g_j - x^{p_i})^2 \tag{4.5}$$

The best particle x_{best} is the one with the smallest squared error (where $x^{best} \leftarrow argmin\{f(x^p), \forall p\}$). Here a data point g_j is assigned to the cluster with the nearest cluster center. This comes under the initialization step.

4.2 Local Optimum Search

After initialization of the data points, the local search is performed for all the particles, which helps in reducing the risk of falling onto a local solution, but this is a relatively time-consuming process. As mentioned above in the Electromagnetism approach, the step length is an important factor to be considered for local search. This step length depends on the limits for each dimension and determines the performance of the overall local search.

If x_{best} is the same as it was in the last iteration, to improve the efficiency of the local search, a one-step K-Means algorithm named K-means operator (KMO) is introduced on x_{best} which yields $x^{\tilde{best}}$. The following are the steps involved in the KMO

1. Reassign each data point to the cluster with the nearest cluster center;
2. Calculate the cluster centers.

If $f(x^{\tilde{best})}$ is smaller than f(x_{best}), the particle x_{best} is replaced by $x^{\tilde{best}}$, otherwise x_{best} is held. Finally, the current best particle x_{best} is updated.

4.3 Charge and Force Calculation

The charge for each particle element is determined by its fitness value as follows:

$$q^{p_i} = exp\Big(-n\frac{f(x^{p_i}) - f(x^{best_k})}{f(x^{worst}) - f(x^{best})}\Big), \forall p, \forall i \tag{4.6}$$

where x_worst is the particle with the largest squared error (where $x_worst \leftarrow argmax\{f(x_p), \forall p\}$), $f(x^{p_i})$ is the sum of squared error between the data points $g_j \in G_{x^{p_i}}$ and centroid x^{p_i} for cluster $i \in (1, \cdots, K)$ and $f(x^{best_k})$ is the sum of squared error between the data points $g_j \in G_{x^{best_k}}$ and centroid x^{best_k} for cluster $k \in (1, \cdots, K)$. For all the elements in x_p and x_{best}, x^{p_i} and x^{best_k} has the

shortest distance (where x^{p_i} and $x^{best_k} \leftarrow argmin\{||x^{p_i} - x^{best_k}||, \forall i, \forall k\}$. Else, if the objective function attains very big values, the fraction may become too small and cause an overflow problem in computing the exponential function. Hence, K is used to avoid such a problem. By using this, the above-mentioned charge equation (Equation 4.2) can be modified and written as follows.

$$q^{p_i} = exp\Big(- DK \frac{f(x^{p_i}) - f(x^{best_k})}{f(x^{worst}) - f(x^{best})} \Big), \forall p, \forall i \tag{4.7}$$

While coming to the total force calculation, this is based on the principle of superposition, which is mentioned above. The overall resultant force between all particle elements determines the actual effect of the optimization process. The final force vector for each particle element is evaluated under the Coulomb's law and the superposition principle as follows:

$$F^{p_i} = \sum_{h \neq p}^{m} (x^{p_i} - x^{h_k}) \frac{q^{p_i} q^{h_k}}{||x^{h_k} - x^{p_i}||^2} \text{ if } f(x^{h_k}) \geq f(x^{p_i}) \tag{4.8}$$

$$F^{p_i} = \sum_{h \neq p}^{m} (x^{h_k} - x^{p_i}) \frac{q^{p_i} q^{h_k}}{||x^{h_k} - x^{p_i}||^2} \text{ if } f(x^{h_k}) < f(x^{p_i}) \tag{4.9}$$

Here, the distance between x^{p_i} and x^{h_k} is the shortest, i.e., x^{p_i} and $x^{h_k} \leftarrow argmin\{||x^{p_i} - x^{h_k}||, \forall i, \forall k\}$.

The vector F^{p_i} will be normalized by using the below equation:

$$\hat{F^{p_i}} = \frac{F^{p_i}}{||F^{p_i}||}, \forall p, \forall i \tag{4.10}$$

4.4 Movement

The change of the d-coordinate $(d(1, \cdots, K))$ for each particle element x^{p_i} is computed with respect to the resultant force as follows:

$$x_{d_i}^{p_i} = \left\{ \begin{array}{ll} x_{d_i}^{p_i} + \lambda \hat{F}_{d_i}^{p_i} (u_d - x_d^p) & \text{if } \hat{F}_{d_i}^{p_i} > 0, \\ x_{d_i}^{p_i} + \lambda \hat{F}_{d_i}^{p_i} (x_{d_i}^{p_i} - l_{d_i}) & \text{if } \hat{F}_{d_i}^{p_i} \leq 0, \end{array} \right\} \forall p \neq best, \forall i, \forall d \tag{4.11}$$

In the above equation, λ is a random step length that is uniformly distributed between zero and one. u_{d_i} and l_{d_i} represent the upper and lower boundary for the d-coordinate in cluster i, respectively.$\hat{F}_{d_i}^{p_i}$ represents the di^th element of the vector $\hat{F}^{p_i}$. The particle moves towards the highest boundary by a random step length if the resultant force is positive. Otherwise, it moves toward the lowest boundary. The best particle does not move at all, because it holds the absolute attraction pulling or repelling all others in the population.

Step 5: The KMO is performed on each particle except x^{best}. The particle x^{best} is replaced by x^p if $f(x^p)$ is smaller than f (x^{best}), otherwise there is no change to x^{best}.

Step 6: The iteration index is increased. If iteration = MAXITER (e.g., 25) or if the x^{best} particle remains the same in next iteration and percentage change in f (x^{best}) value is less than 1%, then the algorithm is stopped and the flow jumps to step-7. Otherwise, it jumps to Step 2.

Step 7: The best particle x^{best} is selected from the last iteration.

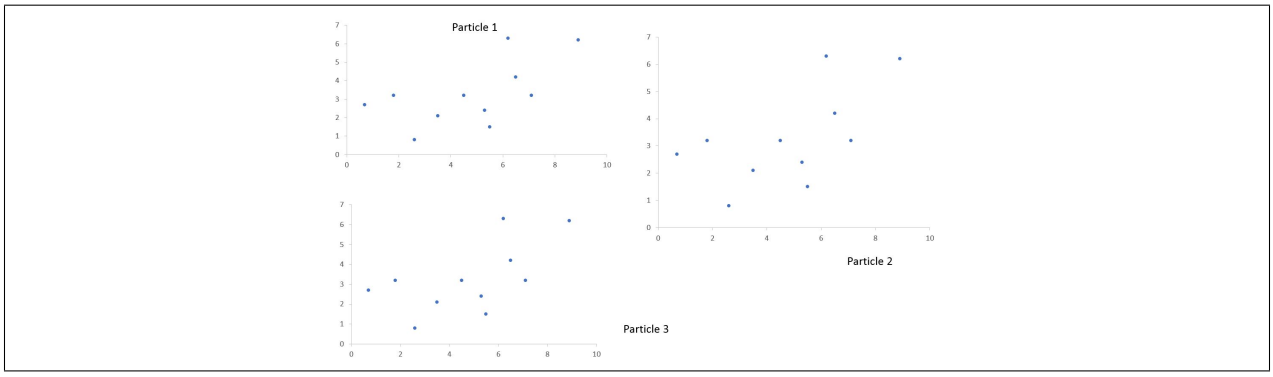

Figure 4.7: Particles View

Same data set is replicated n types where n is number of particles. In this case, particle count is 3, so we see that the data set is replicated 3 times as shown in Figure7.

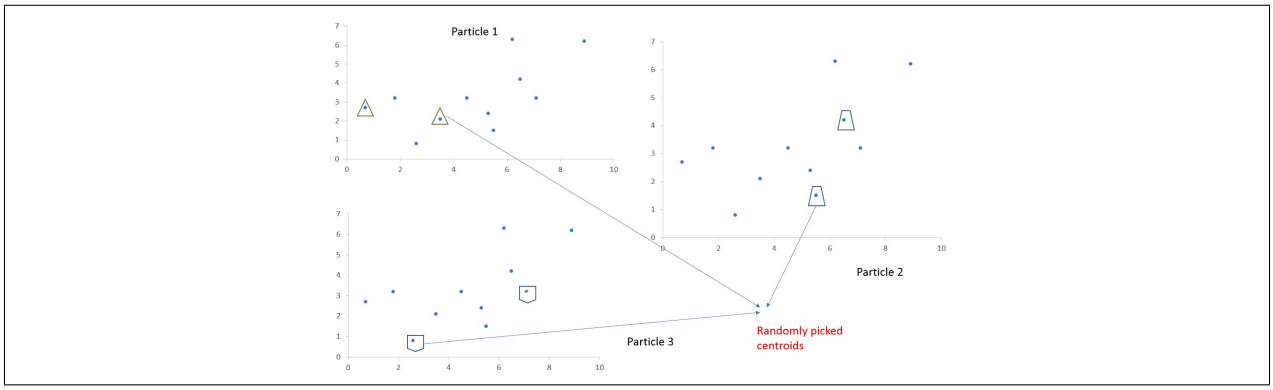

Figure 4.8: Randomly Picked points in each particles

Randomly chosen points in each replica are called a particle in Figure 8. The cluster count chosen to be 2 in this example.

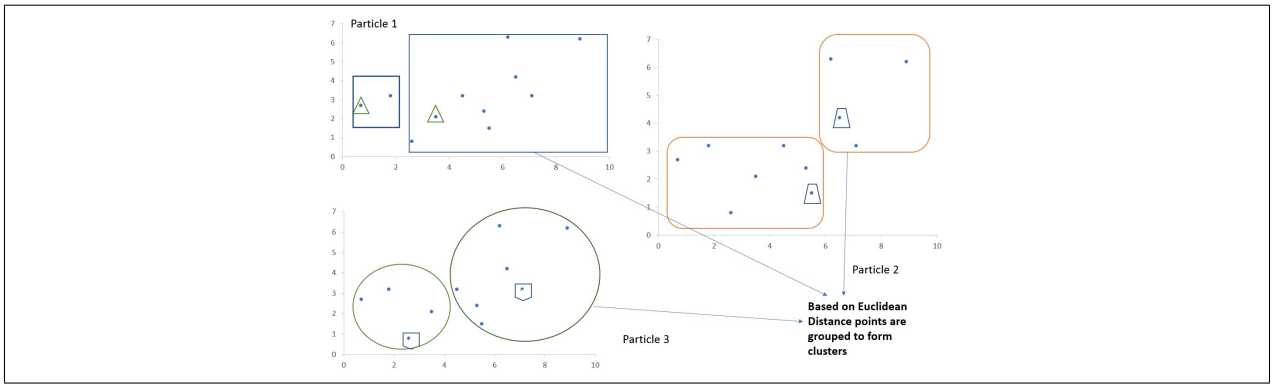

Figure 4.9: Assigning data points

Clusters are formed after assigning the data points to the nearest random centroids formed in the previous step, which can be seen in Figure 9.

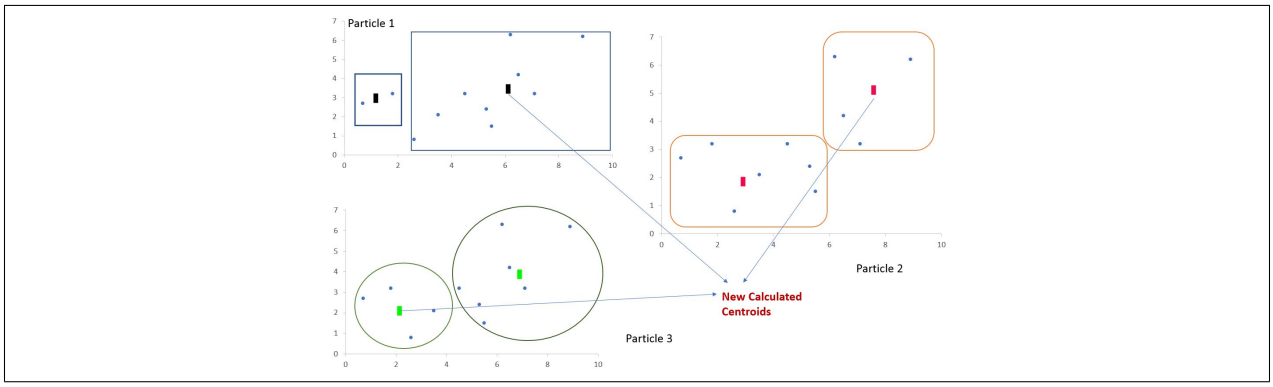

Figure 4.10: New Centroid Calculation

Then we recalculate the cluster centers in each particle. The new centers can be seen in Figure 10.

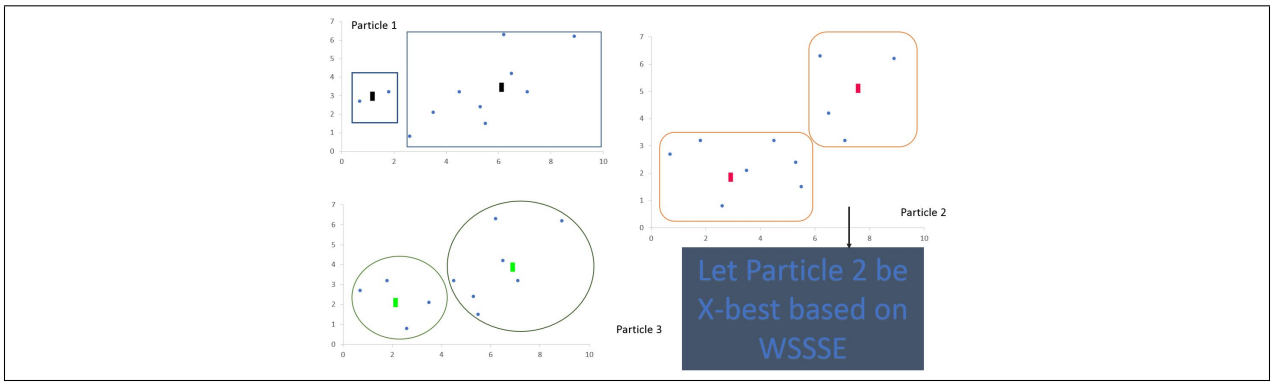

Figure 4.11: Best particle

The best particle is chosen based on WSSSE value as in Figure 11. Here best particle is 2.

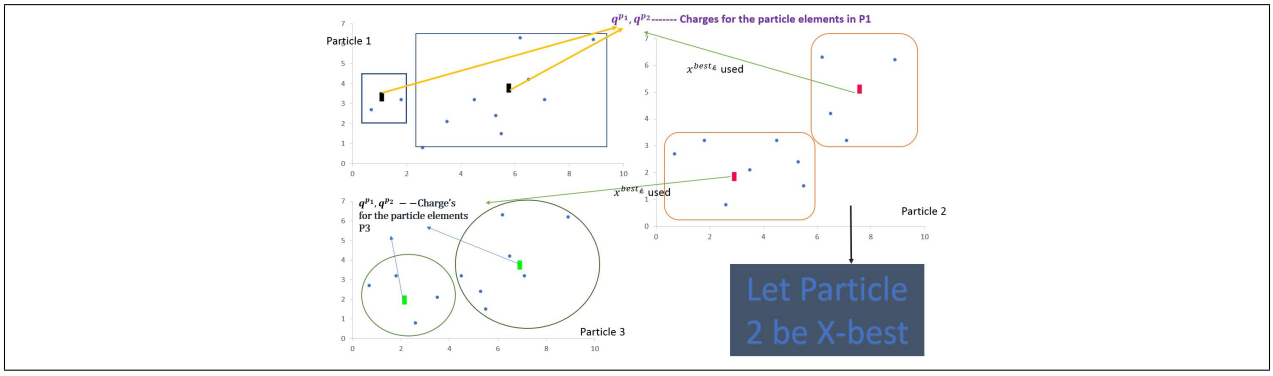

Figure 4.12: Charge Calculation

Later the charge calculation is performed for all the particles leaving the best particle, which can be seen from the Figure 12.

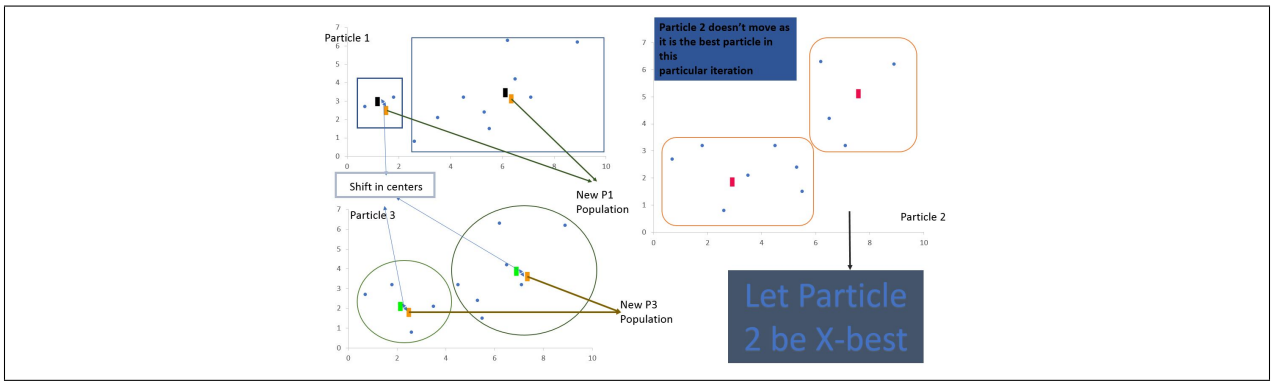

Figure 4.13: New Population Generated

After the force calculation, we move the centers of the particles, which is not X-best and thus end up getting new centers. This process can be seen from the above Figure 13.

CHAPTER V: SYSTEM IMPLEMENTATION

To evaluate the performance of the proposed algorithm for small and big datasets, the implementation is performed in High Performance Computing System using Spark Library. The Spark Library is available in three different languages: Python, Scala and Java. It is an open source library which provides an interface for programming entire clusters with implicit data parallelism and fault tolerance and is mainly used for performing computations on Big data. It is developed to overcome some of the disadvantages in Hadoop.

The usage of Python over other languages is due to the availability of libraries for performing the scientific computations in a much faster way and, it is the most used open source languages in the world.

Hardware and Software Components Used:

The hardware and the software components that were used for simulation are listed below.

- Environment: High Performance Computing System
- Operating System: Linux 2.6.32
- Framework: Spark 2.0.0
- Programming Language: Python 2.6.6

5.1 Apache Spark

Spark began life in 2009 as a project within the AMP Lab at the University of California, Berkeley. Later it became an incubated project of the Apache Software Foundation in 2013. It is general purpose data processing engine mostly used by data scientists to rapidly query, analyze and transform data at scale. Mostly frequently associated tasks with Spark are queries across large datasets, processing data streaming from sensors and several machine learning tasks. It can handle several peta bytes of data at a time. It is often used alongside on the HDFS but it can also work

well with several other data sources like HBase, Cassandra, MapR-DB, MongoDB and Amazon's S3.

RDD stands for Resilient Distributed Dataset. This is the default data structure used in spark. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. These RDD's are stored in main memory and can be of as much size and as long possible. If once created, then RDD cannot be changed. These are immutable in nature.

Advantages of Spark:

The following are some of the important reasons for choosing the Spark library:

Simplicity: Spark's capabilities are accessible via a set of rich APIs, all designed specifically for interacting quickly and easily with data at scale. These APIs are well documented and structured in a way that makes it straightforward for data scientists and application developers to quickly put Spark to work.

Speed: Spark is designed for speed, operating both in memory and on disk. Spark can perform even better when supporting interactive queries of data stored in memory. In those situations, there are claims that Spark can be 100 times faster than Hadoop's Map Reduce.

Support: Spark supports a range of programming languages, including Java, Python, R, and Scala. Although often closely associated with HDFS, Spark includes native support for tight integration with a number of leading storage solutions in the Hadoop ecosystem and beyond. Furthermore, the Apache Spark community is large, active, and international. A growing set of commercial providers including Data bricks, IBM, and all of the main Hadoop vendors deliver comprehensive support for Spark-based solutions.

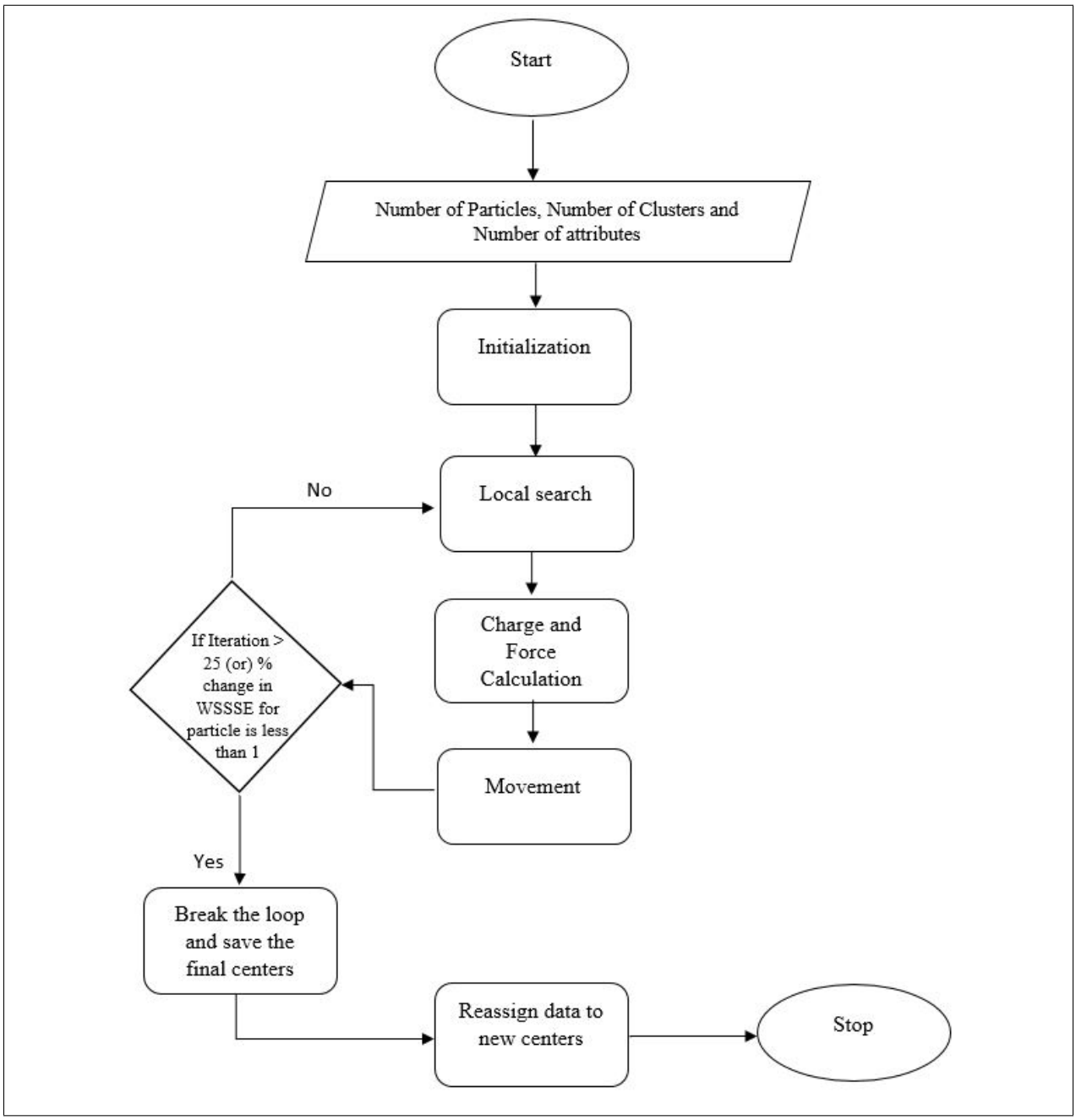

Figure 5.14: Flow Chart

In the above flow chart, WSSSE stands for Within Set Sum of Squared Error. Slurm Jobs are used for submitting the requests to HPC for running the code of proposed algorithm. In the Slurm file, there are multiple parameters mentioned like the output file, error file, number of computational nodes etc. The parallelism parameter is also present in the Slurm job request file. The user mentions the name of the code file in this Slurm file, then the Slurm manager executes the code

mentioned in the request file and output is saved into a new file and is saved with the name specified by the requester. At the end of every Slurm job there will be two files created: output file and error file.

Initialization:

First, a population of m particles are generated with each one containing K randomly and uniformly chosen data points as the centroids. Then the shortest distance between a data point and a centroid is calculated using Map/Reduce.

Algorithm 1 map $(key, value)$

INPUT: cluster centroids X, cluster number K, population size m, the offset of a data point g as key
OUTPUT: $< id, pointObj >$ pair, where id is the cluster index of the data point g and *pointObj* is a data structure that contains the data point g and the shortest distance to any centroid in the particle x^p

1: **for** $p := 1$ to m **do**
2: **for** $k := 1$ to K **do**
3: $dis :=$ calculateDistance$(g - x_k^p)$;
4: **end for**
5: $index :=$ the cluster index where the nearest centroid is;
6: $id :=$ "p" + " $-$ " + "$index$";
7: instantiate a data structure $pointObj$ for the data point g;
8: create and store a $(id, pointObj)$ pair for the particle x^p;
9: **end for**
10: **return** the set of $< id, pointObj >$ pairs of the data point g;

Each map function calculates the distances between a data point g of the dataset and the cluster centroids of the whole population (represented as a single matrix C). The output of each map function is a set of $< id, pointObj >$ pairs, where id is the cluster index to which the data point g is assigned and pointObj is a data structure that contains the data point g and the shortest distance dis to any centroid in the particle x_p.

Local Search:

Algorithm 2 reduce $(id, pointObj)$

INPUT: id is the cluster index, *pointObj* is a data structure that contains the data point g and the shortest distance dis to any centroid in the particle x^p.
OUTPUT: $< id, centroidObj >$ pair, where the id is the cluster index and *centroidObj* is a data structure composed of the new centroid and the total distance between all the data points in the same cluster and the new centroid of the particle x^p.

1: initialize a counter count, a distance f and a vector S with attributes zero;
2: **for all** (id, pointObj) pairs **do**
3: $f := f + dis$;
4: $S := S +$ *pointValue*;
5: $count := count + 1$;
6: **end for**
7: *centroidValue*: = $S/$count;
8: instantiate a data structure *centroidObj*;
9: **return** the $< id, centroidObj >$ pair;

If x^{best} is different from what it was in the last iteration, local search is performed to find $\widetilde{x^{best}}$ such that f (x^{best}) is smaller than f (x^{best}). This is done by one-step of K-means operator (KMO) is performed on x^{best}. In the initialization, the new centroids $\widetilde{x^{best}}$ have been computed, we only need to calculate f ($\widetilde{x^{best}}$) in the local search.

Algorithm 3 map $(key, value)$

INPUT: cluster centroids x^{best}, cluster number K, the offset of a data point g as *key*, the data point g as *value*.
OUTPUT: $< id, shortestDis >$ pair, where *id* is the cluster index of the data point g and *shortestDis* is the shortest distance between the data point g and the *K* centroids in the particle x^{best}.

1: **for** k: = 1 to K **do**
2: dis = calculateDistance$(g - x_k^{best})$;
3: **end for**
4: *id*: = the cluster index where the nearest centroid is;
5: *shortestDis*:= the shortest distance between the data point g and the K centroids;
6: **return** the $< id, shortestDis >$ pair;

Each map function calculates the distances between a data point g of the dataset and the cluster

centroids of the particle x^{best}. The output of each map function is a set of $< id, shortestDis >$ pairs, where id is the cluster index to which the data point g is assigned and shortestDis is the shortest distance between the data point g and the K centroids in the particle x^{best}.

Algorithm 4 reduce (id, shortestDis)

INPUT: *id* is the cluster index of the data point *g* and the shortest distance between the data point *g* and the K centroids in the particle x^{best} as *shortestDis*
OUTPUT: $< id, totalDistance >$ pair, where the *id* is the cluster index of the data point *g* and *totalDistance* is the total

1: initialize a distance f;
2: **for** $(id, shortestDis)$ pairs **do**
3: f := f + shortestDis;
4: **end for**
5: totalDistance := f;
6: **return** the $< id, totalDistance >$ pair;

The charge of each particle element $q^{p_i}, i \in (1, K)$ is calculated locally using the proposed charge computation formula mentioned above.

Total Force Vector Computation:

Algorithm 5 map(key, value)

INPUT:the charge q, the objective function values f, the element index i of the particle x^p as key, the i-th element x^{p_i} as *value*
OUTPUT: $< id, pairedForce >$ pair, where id is an integer pair $< p, i >$ and *pairedForce* is the resultant force between x^{p_i} and $x^{h_k}(p! = h)$

1: identify the element x^{h_k} that is closest to x^{p_i};
2: **if** $f(t) < f(x_p)$ **then**
3: variation := $x^{h_k} - x^{p_i}$;
4: variation := $x^{p_i} - x^{h_k}$;
5: **end if**
6: squaredDis:=computeSquaredDistance(x^{p_i}, x^{h_k});
7: **return** the set of $< id, pointObj >$ pairs of the data point g;

Each Map function here computes the paired force between each particle element present in x_p with

respect to particle elements present in x_h. The particle x_h is determined by $x^{h_k} \leftarrow argmin\{||x^{p_i} - x^{h_k}||, \forall i, \forall k\}$.Later, the output of this map function is used by reduce function for computing the magnitude of each $< id, pairedForce >$ pair present.

Algorithm 6 reduce *(id, pairedForce)*

INPUT: *id* is an integer pair$< p, i >$ and the resultant force between x_i^p and $x_k^h (p! = h)$ as *paired force*
OUTPUT:*(id, overallForce)* pair, where id is an integer pair $< p, h >$ and *overallForce* is the normalized overall resultant force exerting on x^{p_i}

1: initialize overallForce as zero;
2: overallForce := overallForce + pairedForce;
3: overallForce := overallForce/$||overallForce||'$;
4: **return** the $< id, overallForce >$ pair;

Movement:

Because the size of each particle is small, the movement can be efficiently executed on one local machine, and no map/reduce is needed.

CHAPTER VI: RESULTS AND EVALUATION

In this part of the document, the testing using different datasets are explained in detail. Percentage of Correct Answers (PCA), running time and within-cluster sum of squared error are the main parameters used to compare the performance of proposed algorithm with other algorithms. The results of each experiment are provided in tables of values.

Experimentation Setup

The experimentation is carried on various small and big datasets. The small datasets used are the Iris dataset, Glass Identification dataset, Breast Cancer Wisconsin (Diagnostic) dataset, Haberman's Survival dataset. The Big data set used is the Watch-accelerometer dataset which is a part of Heterogeneity Activity Recognition dataset. All the above-listed datasets are available in the UCI machine learning library. The Big dataset consists 10 attributes and 3.5 million records.

This Big dataset consists of readings of two motion sensors commonly found in smartphones. Reading were recorded while users executed activities scripted in no specific order carrying smartwatches and smartphones. The information related to small dataset is preset in Table-1.

Table 6.1: List of Datasets and its information

Dataset	Instances	Attributes present	Clusters	Attributes taken
Iris	150	4	3	4
Glass	214	9	6	9
Cancer-Int	699	9	2	9
Haberman	306	3	2	3
Watch-accelerometer	3.5M	10	6	3
Watch-accelerometer	1.5M	10	3	3

6.1 Results with Small Dataset

Iris Dataset

The Iris Dataset as mentioned above in table-2 consists of 150 instances and it consists of values

pertaining to three flowers (Setosa, Versicolor, and Virginica). The attributes present provides information like sepal length, sepal width, petal length, petal width in cm that are classified into three clusters.

Glass Dataset

This dataset contains information related to various glasses and they are classifies based on oxide levels. The dataset has 214 instances with 9 attributes (RI, Na, Mg, Al, Si, K, Ca, Ba, and Fe) that are classified into 6 clusters (building windows float processed, building windows non-float processed, vehicle windows float processed, containers, tableware, and headlamps).

Cancer-Int Dataset

This data is based on the diagnosis of Breast Cancer Wisconsin (Original) dataset. It provides information pertaining to patient if one is malignant or benign. It consists of 9 attributes and 699 instances. The data is grouped into 2 clusters.

Haberman Dataset

This dataset consists of 306 records with 3 attributes and 2 clusters. The data provides information on patients who survived from breast cancer by undergoing surgery. The cluster states information about patients who survived more than 5 years and who survived less than 5 years.

Table 6.2: PCA and Run Time values for small datasets

Algorithm	Measurements	Iris	Glass	Cancer-Int	Haberman
EMK	PCA(%)	89.4	59.8	61.76	68.4
K-means	PCA(%)	77.1	53.5	60.6	67.3
EMK	Time(SS)	30	48	9	72
K-means	Time(SS)	3	5	5	6

In case of above-mentioned small datasets, the parameters set for running the proposed algorithm are number of particles which varies from 20 to 60, number of iterations is fixed to 100 and

number of computational nodes used are 6. There is a common break condition used, when the percentage change in the WSSSE value is less than 1% for a particle, loop breaks and centers in that loop are taken as the final cluster centers. For each dataset, 10 trails are performed using EMK and K-means algorithms. The results shown above are the average PCA and time for 10 trails.

For PCA calculation, the initial targets provided in data set are taken for calculating the accuracy of clustering. After running the proposed clustering algorithm, the initial data is grouped into clusters based on the targets provided in the dataset. Later the initial grouping is used for calculating the accuracy of the clusters formed by the two algorithms mentioned above table.

6.2 Results with Big Dataset

In case of Big datasets, the parameters set for running the proposed algorithm are number of particles which is fixed to 20, number of iterations is fixed to 10 and number of computational nodes used are 6. The break condition used is same as the one used for the small dataset, when the percentage change in the WSSSE value is less than 1% for a particle, loop breaks and centers in that loop are taken as the final cluster centers. For each dataset, 10 trails are performed using EMK and K-means algorithms. The results shown above are the average WSSSE values for 10 trails. The results while using Big data can be seen in below Table – 3.

Watch-accelerometer

The dataset consists of 3.5million records with 10 attributes and 6 clusters. Out of which only three attributes were taken during clustering named (x, y, and z). These consists of information of six axes from the accelerometer.

A new file is also created holding only 1.5 million records and three clusters (walk, bike, stairs up). This is a portion from the Watch-accelerometer data. The number of attributes remain the same, which is three.

The PCA calculation for 3.5 million records is not available, as the dataset was not provided

Table 6.3: Values for Big datasets Using EMK and K-means

Algorithm	Instances	Clusters	PCA(%)	WSSSE	Time (HH:MM:SS)
EMK	3.5 M	6		9564016.28	5:10:0
K-means	3.5 M	6		14986221.11	0:13:40
EMK	1.5 M	3	44.86	8614425.12	01:26:16
K-means	1.5 M	3	41.22	8725422.22	00:01:51

with crisp target values, which ended up showing inaccurate PCA values. There is huge difference in run times of both the algorithms, this do to the Local search, charge, force and movement calculation present in the proposed algorithm. While coming to K-means it only calculates the mean for finding the new centers at every iteration. If the equations can be optimized, there can be a significant decrease in the run time of the algorithm.

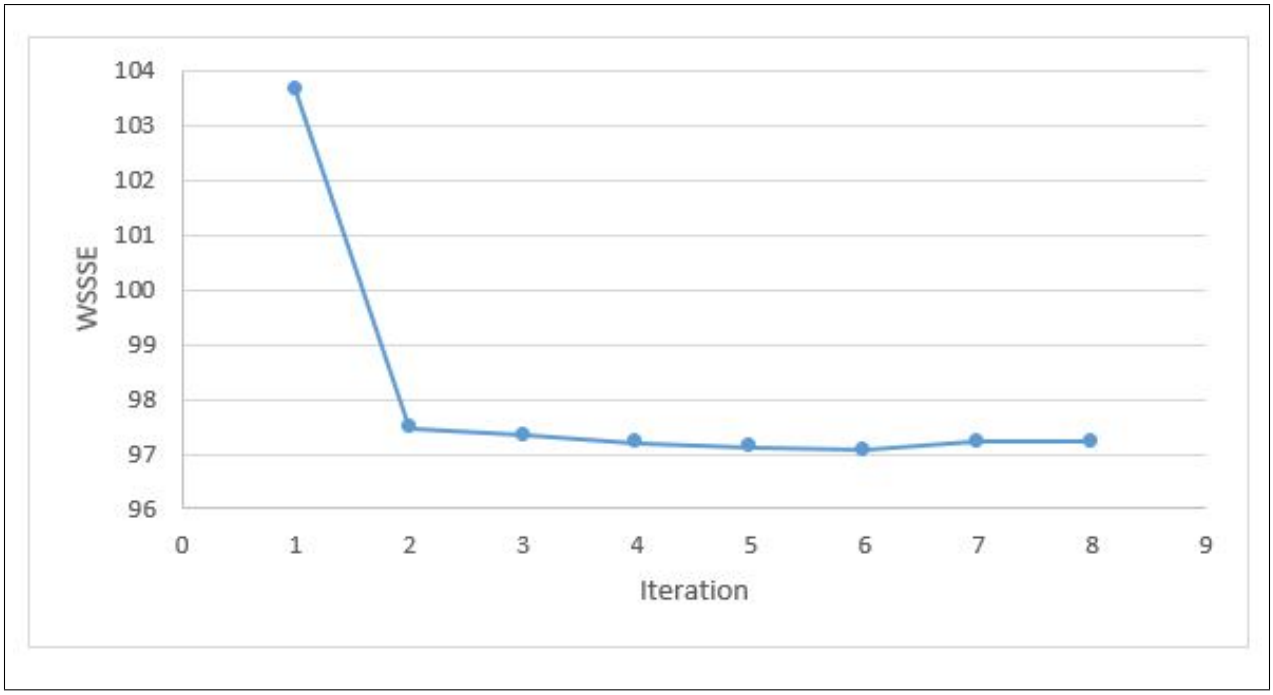

Figure 6.15: Convergence Curve using the EMK Algorithm for Iris Data

The WSSSE value recorded at the end of EMK Algorithm is 97.22 .

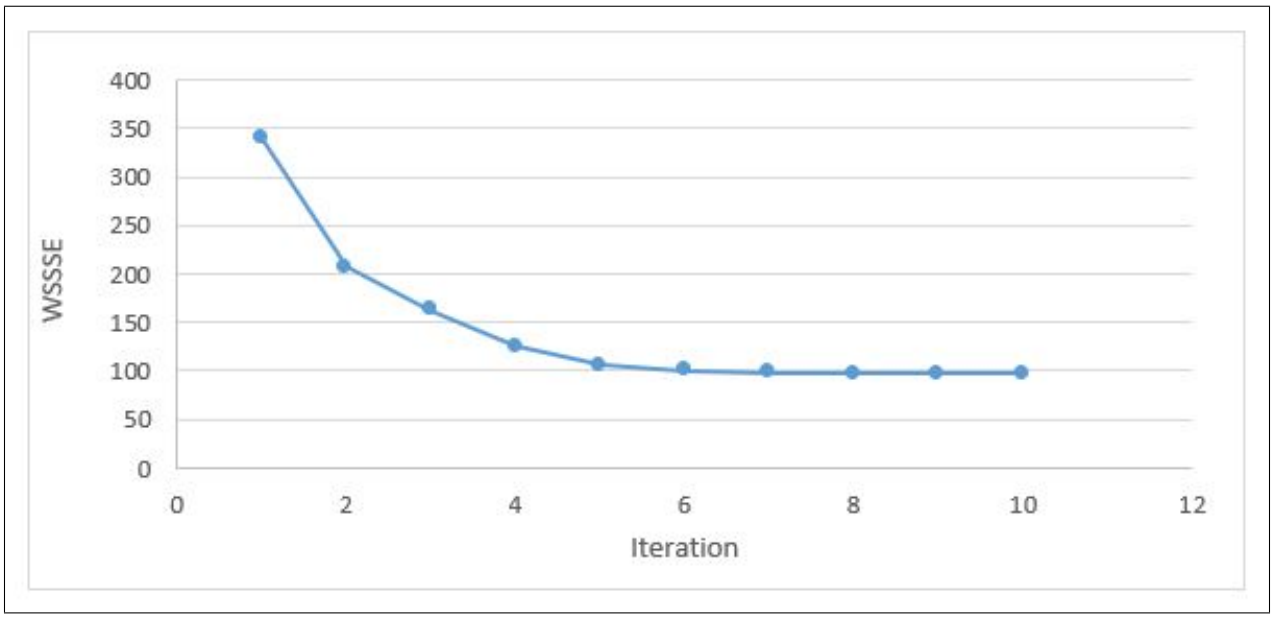

Figure 6.16: Convergence Curve using the K-means Algorithm for Iris Data

The WSSSE value recorded at the end of EMK Algorithm is 97.35

CHAPTER VII: CONCLUSION AND FUTURE WORK

From above the results, it is seen that the performance of EMK algorithm is better when compared to K-means clustering algorithm. The PCA and WSSSE values for EMK is more than the K means algorithm. Accuracy is an important factor to be considered while clustering. Formation of accurate clusters would help in grouping future unseen data more efficiently. The Time taken by EMK is very huge when compared to that with K-means clustering algorithm. Some improvements can be done in the algorithm level to decrease the time taken by the algorithm.

Since, the coding is done using Spark 2.0.0 and Python 2.6.6, there are some limitations in coding. RDD's are the data structures used while programing the proposed algorithm, If RDD were to be replaced with Data Frames, there could be an in increase in the speed of clustering of the proposed system, when compared to that of the current run time. Spark 2.2.0 is a strong library with lot of support provided for data frames. By using Python version 2.7 or 3.5, would help in using more effective and efficient default functions and could also serve as good replacements for some of the current functions used in the code. Testing is done using only few datasets as shown above, testing with more number of labeled big datasets would help in knowing more about the performance of the proposed algorithm. Testing with increased count of nodes would also help in increasing the performance of proposed clustering. This would also require increasing the parallelism while reading the input data into the code to serve the purpose of increased nodes.

The input file type taken here is CSV; there are many other file formats present, which can be used as input for the proposed algorithm. Some of the data types like HDFS, parquet would help in increasing the performance of proposed clustering technique. They would also help in increasing the parallelism in spark.

Bibliography

[1] Abdel-Kader, R. F. Genetically improved pso algorithm for efficient data clustering. In *Machine Learning and Computing (ICMLC), 2010 Second International Conference on* (2010), IEEE, pp. 71–75.

[2] Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J. Optics: ordering points to identify the clustering structure. In *ACM Sigmod record* (1999), vol. 28, ACM, pp. 49–60.

[3] Birbil, Ş. İ., and Fang, S.-C. An electromagnetism-like mechanism for global optimization. *Journal of global optimization 25*, 3 (2003), 263–282.

[4] Celebi, M. E., Kingravi, H. A., and Vela, P. A. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert systems with applications 40*, 1 (2013), 200–210.

[5] Chang, D.-X., Zhang, X.-D., and Zheng, C.-W. A genetic algorithm with gene rearrangement for k-means clustering. *Pattern Recognition 42*, 7 (2009), 1210–1222.

[6] Cuevas, E., Oliva, D., Zaldivar, D., Pérez-Cisneros, M., and Sossa, H. Circle detection using electro-magnetism optimization. *Information Sciences 182*, 1 (2012), 40–55.

[7] Dorigo, M., and Blum, C. Ant colony optimization theory: A survey. *Theoretical computer science 344*, 2-3 (2005), 243–278.

[8] Hassanzadeh, T., and Meybodi, M. R. A new hybrid approach for data clustering using firefly algorithm and k-means. In *Artificial Intelligence and Signal Processing (AISP), 2012 16th CSI International Symposium on* (2012), IEEE, pp. 007–011.

[9] Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C. Optimization by simulated annealing: an experimental evaluation; part i, graph partitioning. *Operations research 37*, 6 (1989), 865–892.

[10] Liu, Y., Wu, X., and Shen, Y. Automatic clustering using genetic algorithms. *Applied mathematics and computation 218*, 4 (2011), 1267–1279.

[11] Manoj, V., and Elias, E. Artificial bee colony algorithm for the design of multiplier-less nonuniform filter bank transmultiplexer. *Information Sciences 192* (2012), 193–203.

[12] Mizuno, K., Takamatsu, S., Shimoyama, T., and Nishihara, S. Fireflies can find groups for data clustering. In *Industrial Technology (ICIT), 2016 IEEE International Conference on* (2016), IEEE, pp. 746–751.

[13] Moliton, A. *Basic electromagnetism and materials*. Springer Science & Business Media, 2007.

[14] Ng, R. T., and Han, J. Clarans: A method for clustering objects for spatial data mining. *IEEE transactions on knowledge and data engineering 14*, 5 (2002), 1003–1016.

[15] Qu, B.-Y., Liang, J. J., and Suganthan, P. N. Niching particle swarm optimization with local search for multi-modal optimization. *Information Sciences 197* (2012), 131–143.

[16] Radcliffe, N. J., and Surry, P. D. Fitness variance of formae and performance prediction. In *FOGA* (1994), vol. 3, pp. 51–72.

[17] Rahman, M. A., and Islam, M. Z. A hybrid clustering technique combining a novel genetic algorithm with k-means. *Knowledge-Based Systems 71* (2014), 345–365.

[18] Sha, M., and Yang, H. Speaker recognition based on apso-k-means clustering algorithm. In *Artificial Intelligence and Computational Intelligence, 2009. AICI'09. International Conference on* (2009), vol. 2, IEEE, pp. 440–444.

[19] Wikipedia. Cluster analysis — wikipedia, the free encyclopedia, 2017. [Online; accessed 2-December-2017].

[20] Yang, X.-S. Firefly algorithm, stochastic test functions and design optimisation. *International Journal of Bio-Inspired Computation 2*, 2 (2010), 78–84.

[21] Yang, X.-S. A new metaheuristic bat-inspired algorithm. *Nature inspired cooperative strategies for optimization (NICSO 2010)* (2010), 65–74.

APPENDIX A: STEPS TO IMPLEMENT THE ALGORITHM IN HPC CLUSTER

This simulation requires HPC access. HPC stands for High Performance Computing environment, which are generally used to solve large problems in Science, Engineering, or Business. Programs are run on HPC by using the Slurm system. Using Slurm job, a request is submitted to run a particular code. Later, the output files and the error files are generated by the system. To run the code against any dataset the following steps needs to be followed.

Procedure:

1. Login to “hpcm.tamucc.edu" with the login credentials using Putty Software.
2. Load the code into HPCM server using the below command.

   scp filename location

   Ex: scp file1.py lsmith@hpcm.tamucc.edu/home
3. As mentioned above, A Slurm file needs to be prepared consisting of program names and user preferred names for the output and error files.
4. Copy the dataset into the same location where the code file is present in the HPCM server.
5. Make sure all the three files are present in the same path in the HPCM server.
6. Update the parameters in code for selecting the number of clusters and choosing the attributes for clustering.
7. Update the values of targets in the code for accuracy calculation.
8. Increase the maximum user processes to 5000 before you run the job using the below command.

   Ulimit –u 5000
9. Use the command below to submit a job in HPCM.

   Sbatch “slurm file name”

10. To see the status of the job use the below command.

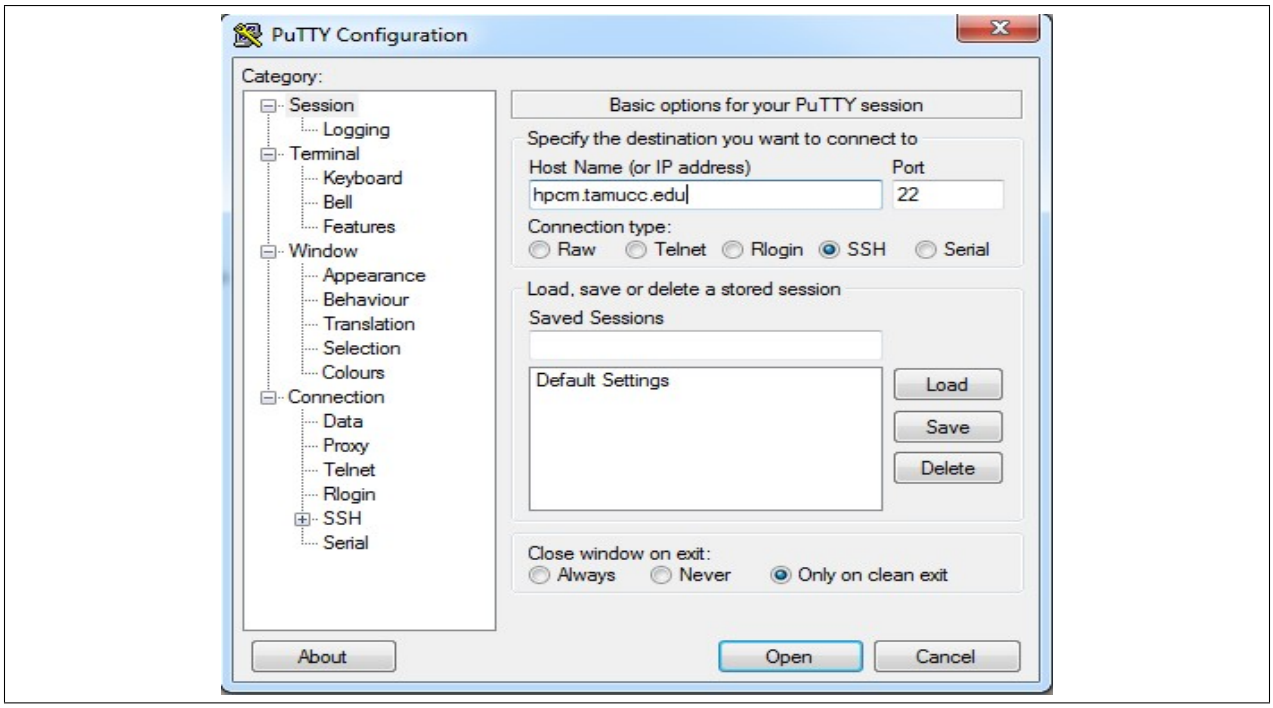

Figure 8.17: Putty Software

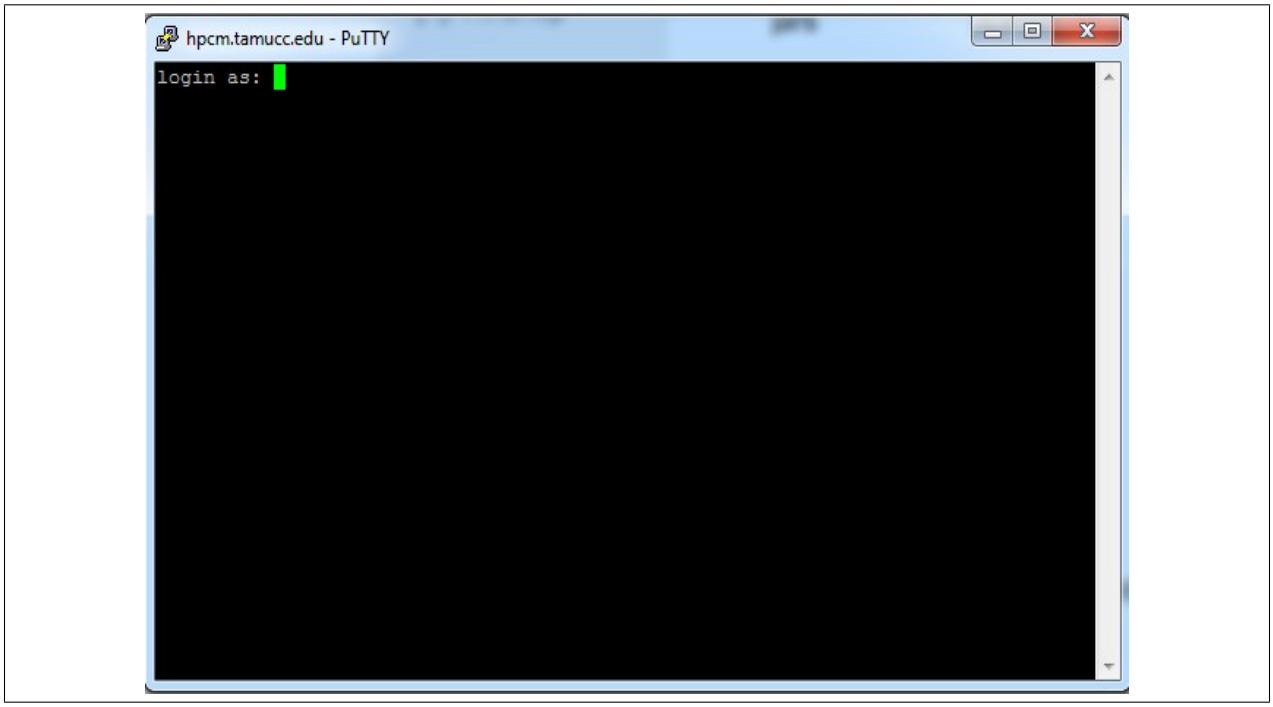

Figure 8.18: Login Screen

```
]#!/bin/bash
 #SBATCH -J bigdatj               # Name of the job
 #SBATCH -o output.out            # Name of file that will have program output
 #SBATCH -e error.err             # Name of the file that will have job errors, if any
 #SBATCH -N 6                     # Number of nodes ( the normal cluster partion has 22 total )
 #SBATCH --ntasks-per-node 4
 #SBATCH --cpus-per-task 4
 #SBATCH --mem 128G               #Total memory which is caliculated by multiplying the executor memry value with 4
 #SBATCH --exclusive
 #SBATCH -t 50:00:00    # Time limit
·#SBATCH -p normal      # Partition
 spark-submit --total-executor-cores 96             \
              --executor-memory 32G                 \
              --driver-memory 100G                  \
              --driver-cores 16                     \
              --conf spark.default.parallelism=64   \
              EMK_Algorithm.py  #File name of the code to be executed
```

Figure 8.19: Sample Slurm File

APPENDIX B: CODE FOR THE PROJECT

1)The input parameters, no of particles, no of clusters needed etc are provided to the program through the follwoing set of instructions.

```
no_of_attributes = 10
no_of_particles = 20
no_of_clusters = 3
no_of_iterations = 10
#Place the input file for which the clustering
needs to be performed in the below "data" variable
data = sc.textFile("Watch-1.5.csv",minPartitions=200)
header = data.first()
data_valid = data.filter(lambda row: row != header)
.map(lambda p:data_validation(p))
grped_data_valid = data_valid
.reduceByKey(lambda x,y:add_tuples_tolist(x,y))
```

2)Based on the no of input particles provided, randomly centroids are picked from the input to form particles

```
for i in range(no_of_particles):
    index_list.append(i)
for i in range(no_of_particles):
    ran_num = random.randrange(0,100)
    new_arr[i] = data6.takeSample(False,no_of_clusters,ran_num)
    final_temp[i] = map(list,list(zip(index_list,new_arr[i])))
dict1 = dict(zip(index_list,final_temp))
```

3)This function takes only the specified attributes from the inpur csv file and load then into an RDD. The comma seperated values are split and saved column wise into RDD.

```
def initial_data(p,cnt):
    z = []
    cols = [3,4,5]
    after_split = p.split(',')
    for i in range(len(after_split)):
    if i in cols:
       if i < cnt:
          z.append(float(after_split[i]))
    return z
```

4)This function calculates the distance between centroid and the data point and return backs the index of the closest centroid for each data point.This is repeated for all the particles present.

```
def closestPoint(p, dict):
      bestIndex = 0
      particle = 0
      closest = float("+inf")
      key_iterator = 0
      final_new_array = [[]] * no_of_particles
      for key,value in dict.items():
          v1= value
          for i in range(len(v1)):
               tempDist = numpy.sqrt
               (numpy.sum(numpy.subtract(p, v1[i][1]) ** 2))
```

```
                if tempDist < closest:
                    closest = tempDist
                    bestIndex = v1[i][0]
                    particle = key
            key_iterator =
            key_iterator + 1
            final_new_array[key] =
            ((particle,bestIndex), [p, closest])
            closest = float("+inf")
        return final_new_array
```

5)Function used for calculating the charge of each particle element.

```
def chargefun(p,q,r,s):
    charge = []
    for u in range(len(q)):
    if q[u][0][0] == p[0][0][0]:
        best_dist = q[u][1]
        mul_factor = s
    for i in range(len(p)):
        numerator = p[i][1][1] - best_dist
        temp_fin_val = numpy.exp((-mul_factor)*(numerator/r))
        charge.append((p[i][0],temp_fin_val))
    return charge
```

6)Following function is used for calculating the movement. This return the new positions of each centroid present in particle.

```
def movement_fun(p,q,cnt):
    lam = numpy.random.random(1)[0]
    final_move = []
    nan_val = 0
    tot = p[1][0]
    if p[0][0] != q[0]:
        if nan_val == 0:
            if tot <= 0:
                move = lam * tot
                sub = list(numpy.subtract(p[1][1],p[1][3][0]))
                final_move = list(numpy.multiply(move,sub))
            else:
                move = lam * (p[1][0] + p[1][1])
                sub = list(numpy.subtract(p[1][2][0],p[1][1]))
                final_move = list(numpy.multiply(move,sub))
            return (p[0],list(numpy.sum([p[1][1],final_move],axis=0)))
    elif p[0][0] == q[0]:
        return (p[0],p[1][1])
    else:
        return 0
```

7)This function is used for preparing the input dataset into clusters based on the targets in the dataset. These are used for accuracy calculation of the clusters formed from EMK algorithm.

```
def data_validation(p):
    z = []
    target = 9
    cols = [3,4,5]
```

```
k = []
after_split = p.split(',')
for i in range(len(after_split)):
    if i in cols:
       z.append(float(after_split[i]))
if after_split[target] == 'walk':
    mm = 0s
elif after_split[target] == 'bike':
    mm = 1
elif after_split[target] == 'stairsup':
    mm = 2
else:
    print after_split[target]
return (mm,tuple(z))
```

ABSTRACT

Over the past few years, Nature has been the source of inspiration for many proposed successful algorithms. This paper proposes a new nature-inspired K-means clustering algorithm which is based on the concept of Electromagnetism. The proposed algorithm starts by initializing a set of particles and later in the second step, the best particle among them is chosen based on the fitness function. After choosing the best particle, an objective function value is calculated for each particle which is initialized. Then the force and movement are calculated for each particle except for the current best particle. This way, the algorithm at each iteration searches for a local best particle and then calculates objective function values. Due to this reason, the position of the initialized particles also changes. Algorithm terminates when it reaches the maximum iterations or when the change in Within Set Sum of Squared Error (WSSSE) is less than 0.0001. The detailed explanation of this algorithm is presented. From the results, Electromagnetism based K-means provides better accuracy when compared to K-means clustering. This can be seen from the results section.