CONTENTS

Contents **Page No.**

List of Figures

List of Tables

1. INTRODUCTION

1.1 Overview of Data Clustering

Data Clustering is a data mining technique of grouping set of data objects into multiple groups or clusters so that objects within the cluster have high similarity, but are very dissimilar to objects in the other clusters. Dissimilarities and similarities are assessed based on the attribute values describing the objects. Data clustering is a method in which we make cluster of objects that are somehow similar in characteristics. The criterion for checking the similarity is implementation dependent. It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.

Clustering is often confused with classification, but there is some difference between the two. In classification the objects are assigned to predefined classes, whereas in clustering the classes are also to be defined. Data Clustering is a technique in which, the information that is logically similar is physically stored together. In order to increase the efficiency in the database systems the numbers of disk accesses are to be minimized. In clustering the objects of similar properties are placed in one class of objects and a single access to the disk makes the entire class available.

Clustering algorithms are used to organize data, categorize data, for data compression and model construction, for detection of outliers etc. Common approach for all clustering techniques is to find clusters center that will represent each cluster. Cluster centre will represent with input vector can tell which cluster this vector belong to by measuring a similarity metric between input vector and all cluster centre and determining which cluster is nearest or most similar one. Cluster analysis can be used as a standalone data mining tool to gain insight into the data distribution, or as a preprocessing step for other data mining algorithms operating on the detected clusters. Many clustering algorithms have been developed and are categorized from several aspects such as partitioning methods, hierarchical methods, density-based methods, and grid-based methods.Further data set can be numeric or categorical. Inherent geometric properties of numeric data can be exploited to naturally define distance function between data points. Whereas categorical data can be derived from either quantitative or qualitative data where observations are directly observed from counts.

This project consists of K-Means, Particle Swarm Optimization (PSO) and PSO with K-means document clustering algorithm that performs fast document clustering and can avoid being trapped in a local optimal solution on various high dimensional datasets. The hybrid PSO with K-means algorithm combines the ability of the globalized searching of the PSO algorithm and the fast convergence of the K-means algorithm. The results obtained are analyzed and compared for accuracy and performance of the algorithm on large datasets.

1.2 Requirements of Clustering in Data Mining

The following points throw light on why clustering is required in data mining −

- **Scalability** − We need highly scalable clustering algorithms to deal with large databases.
- **Ability to deal with different kinds of attributes** − Algorithms should be capable to be applied on any kind of data such as interval-based (numerical) data, categorical, and binary data.
- **Discovery of clusters with attribute shape** − The clustering algorithm should be capable of detecting clusters of arbitrary shape. They should not be bounded to only distance measures that tend to find spherical cluster of small sizes.
- **High dimensionality** − The clustering algorithm should not only be able to handle low-dimensional data but also the high dimensional space.
- **Ability to deal with noisy data** − Databases contain noisy, missing or erroneous data. Some algorithms are sensitive to such data and may lead to poor quality clusters.

- **Interpretability** − The clustering results should be interpretable, comprehensible, and usable.

1.3 Clustering Methods

Clustering methods can be classified into the following categories −

- Partitioning Method
- Hierarchical Method
- Density-based Method
- Grid-Based Method
- Model-Based Method
- Constraint-based Method

1.3.1 Partitioning Method

Suppose we are given a database of ‘n’ objects and the partitioning method constructs ‘k’ partition of data. Each partition will represent a cluster and k ≤ n. It means that it will classify the data into k groups, which satisfy the following requirements −

- Each group contains at least one object.
- Each object must belong to exactly one group.

Points to remember

- For a given number of partitions (say k), the partitioning method will create an initial partitioning.
- Then it uses the iterative relocation technique to improve the partitioning by moving objects from one group to other.

1.3.2 Hierarchical Methods

This method creates a hierarchical decomposition of the given set of data objects. We can classify hierarchical methods on the basis of how the hierarchical decomposition is formed. There are two approaches here

- Agglomerative Approach
- Divisive Approach

Agglomerative Approach

This approach is also known as the bottom-up approach. In this, we start with each object forming a separate group. It keeps on merging the objects or groups that are close to one another. It keep on doing so until all of the groups are merged into one or until the termination condition holds.

Divisive Approach

This approach is also known as the top-down approach. In this, we start with all of the objects in the same cluster. In the continuous iteration, a cluster is split up into smaller clusters. It is down until each object in one cluster or the termination condition holds. This method is rigid, i.e., once a merging or splitting is done, it can never be undone.

Approaches to Improve Quality of Hierarchical Clustering

Here are the two approaches that are used to improve the quality of hierarchical clustering −

- Perform careful analysis of object linkages at each hierarchical partitioning.
- Integrate hierarchical agglomeration by first using a hierarchical agglomerative algorithm to group objects into micro-clusters, and then performing macro-clustering on the micro-clusters.

1.3.3 Density-based Method

This method is based on the notion of density. The basic idea is to continue growing the given cluster as long as the density in the neighborhood exceeds some threshold, i.e., for each data point within a given cluster, the radius of a given cluster has to contain at least a minimum number of points.

1.3.4 Grid-based Method

In this, the objects together form a grid. The object space is quantized into finite number of cells that form a grid structure.

Advantage

- The major advantage of this method is fast processing time.
- It is dependent only on the number of cells in each dimension in the quantized space.

1.3.5 Model-based methods

In this method, a model is hypothesized for each cluster to find the best fit of data for a given model. This method locates the clusters by clustering the density function. It reflects spatial distribution of the data points.

This method also provides a way to automatically determine the number of clusters based on standard statistics, taking outlier or noise into account. It therefore yields robust clustering methods.

1.3.6 Constraint-based Method

In this method, the clustering is performed by the incorporation of user or application-oriented constraints. A constraint refers to the user expectation or the properties of desired clustering results. Constraints provide us with an interactive way of communication with the clustering process. Constraints can be specified by the user or the application requirement.

1.4 Applications of Cluster Analysis

- Clustering analysis is broadly used in many applications such as market research, pattern recognition, data analysis, and image processing.
- Clustering can also help marketers discover distinct groups in their customer base. And they can characterize their customer groups based on the purchasing patterns.
- In the field of biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionalities and gain insight into structures inherent to populations.
- Clustering also helps in identification of areas of similar land use in an earth observation database. It also helps in the identification of groups of houses in a city according to house type, value, and geographic location.
- Clustering also helps in classifying documents on the web for information discovery.
- Clustering is also used in outlier detection applications such as detection of credit card fraud.
- As a data mining function, cluster analysis serves as a tool to gain insight into the distribution of data to observe characteristics of each cluster.

1.5 Data Mining

Another important application of clustering is in the field of data mining. Data mining is defined as follows.

1.5.1 Definition1: "Data mining is the process of discovering meaningful new correlation, patterns and trends by shifting through large amounts of data, using pattern recognition technologies as well as statistical and mathematical techniques."

1.5.2 Definition2: Data mining is a "knowledge discovery process of extracting previously unknown, actionable information from very large databases."

1.5.3 Use of Clustering in Data Mining: Clustering is often one of the first steps in data mining analysis. It identifies groups of related records that can be used as a starting point for exploring further relationships. This technique supports the development of population segmentation models, such as demographic-based customer segmentation. Additional analyses using standard analytical and other data mining techniques can determine the characteristics of these segments with respect to some desired outcome. For example, the buying habits of multiple population segments might be compared to determine which segments to target for a new sales campaign.

1.6 Advantages of Data Mining

- Predict future trends,customer purchase habits
- Help with decision making
- Improve company revenue and lower costs
- Market based analysis
- Fraud detection

1.7 Problem Definition

The aim of the project is to do data clustering and data retrieval on large datasets within the given time limit. To classify the datasets into clusters, three algorithms namely K-Means, Particle Swarm Optimization (PSO), Hybrid K-Means with PSO have been used. This project presents a hybrid PSO with K-Means data clustering algorithm that performs fast data clustering and can avoid being trapped in a local optimal solution as well.

The above mentioned algorithms are compared under various environments. We can arrive at a conclusion in deciding the best algorithm among them.

1.8 Objectives

The purpose of the project is to develop PSO, K-Means and Hybrid K-Means with PSO Clustering. The aim is to compare these algorithms in various environments and study their performance using various parameters.

The following are the objectives:

- Apply PSO algorithm on the data sets. Find the Object function Value and CPU time.
- Apply K-Means algorithm on various datasets.Calculate Fitness, Overall Purity, Minimum and Maximum values, CPU time.
- Now combine K-Means and PSO algorithms, i.e. give output of K-Means algorithm as input to the PSO algorithm and thus obtain K-Means with PSO algorithm. Calculate Fitness, Overall Purity, Distance, Minimum and Maximum values, CPU time.
- Comparison between Fitness, Overall Purity, Minimum and Maximum values, Distance and CPU time is done to compare the algorithms.
- To understand the performance other combinations like K-PSO-K, PSO-K were also implemented.

1.9 Methodology

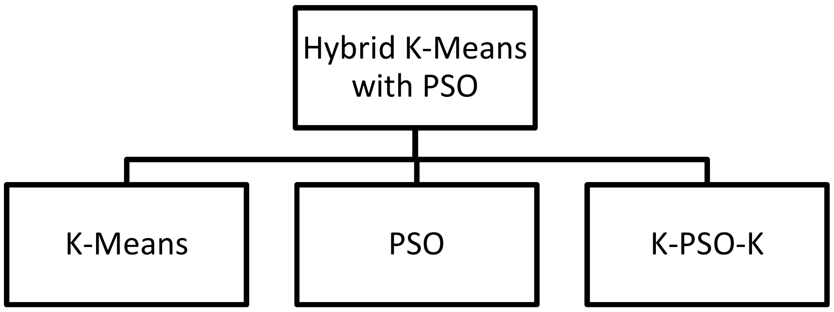


Fig 1.1: Flow of the project

1.10 Study Area and Characteristics

This project uses 6 data sets. These data sets are taken from UCI machine repository:

1.10.1Heart Disease Dataset: This dataset contains 14 attributes and 303 instances. The Cleveland dataset was taken under consideration.

1.10.2 GPS Trajectories Dataset: This dataset contains 15 attributes and 163 instances. The Cleveland dataset was taken under consideration.

1.10.3 Online News Popularity Dataset: This dataset contains 61 attributes and 39797 instances. The Cleveland dataset was taken under consideration.

1.10.4 Seeds Dataset: This dataset contains 7 attributes and 210 instances. The Cleveland dataset was taken under consideration.

1.10.5 Wholesale Customers Data: This dataset contains 8 attributes and 440 instances. The Cleveland dataset was taken under consideration.

1.10.6 Prostate Cancer Dataset: This dataset contains various attributes and instances. The Cleveland dataset was taken under consideration.

Datasets	**Data types**	**Attribute types**	**No. of Instances**	**No. of Attributes**
Heart Disease	Multivariate	Categorical/ Integer/Real	303	14
Online News Popularity	Multivariate	Real/ Integer	39797	61
Seeds	Multivariate	Real	210	7
GPS Trajectories	Multivariate	Real	163	15
Wholesale Customers Data	Multivariate	Integer	440	8

Table 1.10.1: Datasets under study

2 SOFTWARE DESCRIPTION

2.1. Introduction to MATLAB

2.1.1What is MATLAB?

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

1. Math and computation
2. Algorithm development
3. Data acquisition
4. Modeling, simulation, and prototyping
5. Data analysis, exploration, and visualization
6. Scientific and engineering graphics
7. Application development, including graphical user interface building.

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non interactive language such as C or FORTRAN.

The name MATLAB stands for *matrix laboratory*. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of add-on application-specific solutions called *toolboxes*. Very important to most users of MATLAB, toolboxes allow you to *learn*and *apply* specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

2.2 The MATLAB System:

The MATLAB system consists of five main parts:

2.2.1 Development Environment:

This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, an editor and debugger, and browsers for viewing help, the workspace, files, and the search path.

2.2.2 The MATLAB Mathematical Function:

This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigen values, Bessel functions, and fast Fourier transforms.

2.2.3 The MATLAB Language:

This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.

2.2.4 Graphics:

MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

2.2.5 The MATLAB Application Program Interface (API):

This is a library that allows you to write C and Fortran programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

2.3 MATLAB WORKING ENVIRONMENT:

2.3.1. MATLAB desktop:

Matlab Desktop is the main Matlab application window. The desktop contains five sub windows, the command window, the workspace browser, the current directory window, the command history window, and one or more figure windows, which are shown only when the user displays a graphic.

The command window is where the user types MATLAB commands and expressions at the prompt (>>) and where the output of those commands is displayed. MATLAB defines the workspace as the set of variables that the user creates in a work session. The workspace browser shows these variables and some information about them. Double clicking on a variable in the workspace browser launches the Array Editor, which can be used to obtain information and income instances edit certain properties of the variable.

The current Directory tab above the workspace tab shows the contents of the current directory, whose path is shown in the current directory window. For example, in the windows operating system the path might be as follows: C:\MATLAB\Work, indicating that directory “work” is a subdirectory of the main directory “MATLAB”; WHICH IS INSTALLED IN DRIVE C. clicking on the arrow in the current directory window shows a list of recently used paths. Clicking on the button to the right of the window allows the user to change the current directory.

MATLAB uses a search path to find M-files and other MATLAB related files, which are organize in directories in the computer file system. Any file run in MATLAB must reside in the current directory or in a directory that is on search path. By default, the files supplied with MATLAB and math works toolboxes are included in the search path. The easiest way to see which directories are soon the search paths, or to add or modify a search path, is to select set path from the File menu the desktop, and then use the set path dialog box. It is good practice to add any commonly used directories to the search path to avoid repeatedly having the change the current directory.

The Command History Window contains a record of the commands a user has entered in the command window, including both current and previous MATLAB sessions. Previously entered MATLAB commands can be selected and re-executed from the command history window by right clicking on a command or sequence of commands. This action launches a menu from which to select various options in addition to executing the commands. This is useful to select various options in addition to executing the commands. This is a useful feature when experimenting with various commands in a work session.

2.3.2. Using the MATLAB Editor to create M-Files:

The MATLAB editor is both a text editor specialized for creating M-files and a graphical MATLAB debugger. The editor can appear in a window by itself, or it can be a sub window in the desktop. M-files are denoted by the extension .m, as in pixelup.m. The MATLAB editor window has numerous pull-down menus for tasks such as saving, viewing, and debugging files. Because it performs some simple checks and also uses color to differentiate between various elements of code, this text editor is recommended as the tool of choice for writing and editing M-functions. To open the editor, type edit at the prompt opens the M-file filename.m in an editor window, ready for editing. As noted earlier, the file must be in the current directory, or in a directory in the search path.

2.3.3. Getting Help:

The principal way to get help online is to use the MATLAB help browser, opened as a separate window either by clicking on the question mark symbol (?) on the desktop toolbar, or by typing help browser at the prompt in the command window. The help Browser is a web browser integrated into the MATLAB desktop that displays a Hypertext Markup Language(HTML) documents. The Help Browser consists of two panes, the help navigator pane, used to find information, and the display pane, used to view the information. Self-explanatory tabs other than navigator pane are used to perform a search.

Hardware Requirements:

- RAM: 512 MB to 1GB
- CPU: 1133 MHz
- Hard disc space: 200MB

Software Requirements:

- MATLAB R2008b (7.7 version)
- Windows XP and above versions

3. K-MEANS ALGORITHM

3.1 Introduction

In recent years, it has been recognized that the partitional clustering technique is well suited for clustering a large document dataset due to their relatively low computational requirements. The time complexity of the partitioning technique is almost linear, which makes it widely used. The best-known partitioning clustering algorithm is the K-means algorithm and its variants. This algorithm is simple, straightforward and is based on the firm foundation of analysis of variances. The *k*-means algorithm assigns each point to the cluster whose center (also called centroid) is nearest. The center is the average of all the points in the cluster — that is, its coordinates are the arithmetic mean for each dimension separately over all the points in the cluster. The K-means algorithm clusters a group of data vectors into a predefined number of clusters. It starts with a random initial cluster center and keeps reassigning the data objects in the dataset to cluster centers based on the similarity between the data object and the cluster center. The reassignment procedure will not stop until a convergence criterion is met.

3.1.1 Similarity Metric

In most clustering algorithms, the dataset to be clustered is represented as a set of vectors *X={x1, x2, ….,xn}*, where the vector *xi* corresponds to a single object and is called the feature vector. The feature vector should include proper features to represent the object.

The similarity between two documents needs to be measured in a clustering analysis. Over the years, two prominent ways have been proposed to compute the similarity between documents m_p and $m_{j.}$

3.1.2 Fitness

In Each iteration, the particle adjusts the centroid vector' position in the vector space according to its own experience and those of its neighbors. The average distance between a cluster centroid and a document is used as the fitness value to evaluate the solution represented by each particle. The fitness value is measured by the equation below:

$$f=\sum_{i=1}^{N_c}\{\sum_{j=1}^{P_i} d(o_i,m_{ij})/P_i\}/N_c \quad (8)$$

where m_{ij} denotes the jth document vector, which belongs to cluster i; O_i is the centroid vector of **i**th cluster; $d(o_i, m_{ij})$ is the distance between document m_{ij} and the cluster centroid O_i.; $P_i$stands for the document number, which belongs to cluster C_i; N_c stands for the cluster number.

3.1.3 Purity

It is known that the K-means algorithm tends to converge faster than other clustering algorithms, but usually with a less accurate clustering result. So, here in our project our clustering precision is measured using Purity.

$$\mathrm{Purity}(C_j)=\max(|C_j|_{class=i})/|C_j|$$

$$\text{Overall Purity}=\sum_{j=1}^{k}(|C_j|\mathrm{Purity}(C_j))/|D| \quad (9)$$

Where k represents the number of clusters, D represents the size of the dataset, $|C_j|$ is size of the clusters and $|C_j|_{class=i}$ denotes the number of items of class I assigned to cluster j.

3.1.4 Cluster Centroid

Calculate the cluster centroid vector cj, by using equation as follows.

$$cj=1/n_j\Sigma d_j$$
$$\text{for all } d_j \in S_j \quad (10)$$

Where d_j denotes the document vectors that belong to cluster S_j; c_j stands for the centroid vector; n_j is the number of document vectors belong to cluster S_j.

3.1.5 Silhouette distance

silhouette(X,clust) plots cluster silhouettes for the *n*-by-*p* data matrix X, with clusters defined by clust. Rows of X correspond to points, columns correspond to coordinates. clust can be a categorical variable, numeric vector, character matrix, or cell array of strings containing a cluster name for each point. silhouette treats NaNs or empty strings in clust as missing values, and ignores the corresponding rows of X. By default, silhouette uses the squared Euclidean distance between points in X.

3.2 Algorithm for K-means

(1) Inheriting cluster centroid vectors from the datasets.

(2) Assigning each data vector to the closest cluster centroids.

(3) Recalculating the cluster centroid vector cj using the above equation.

(4) Repeating step 2 and 3 until the convergence is achieved.

3.3 Problems with K-means

The main drawback of the K-means algorithm is that the cluster result is sensitive to the selection of the initial cluster centroids and may converge to the local optima. Therefore, the initial selection of the cluster centroids decides the main processing of K-Means and the partition result of the dataset as well. The main processing of K-means is to search the local optimal solution in the vicinity of the initial solution and to refine the partition result. The same initial cluster centroids in a dataset will always generate the same cluster results. However, if good initial clustering centroids can be obtained using any of the other techniques, the K-means would work well in refining the clustering centroids to find the optimal clustering centers. It is necessary to employee some other global optimal searching algorithm for generating this initial cluster centroids. The Particle Swarm Optimization (PSO) algorithm is a population based stochastic optimization technique that can be used to find an optimal, or near optimal, solution to a numerical and qualitative problem.

3.4 Advantages of K-means

- Fast and High quality data clustering.
- Efficiently navigate, summarize, and organize the information.

- It is one of the partitional clustering algorithm which is more suitable for large datasets
- It can be easily implemented.

4. PARTICLE SWARM OPTIMIZATION

4.1. Introduction

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling.PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles.

Each particle keeps track of its coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called *pbest*. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the neighbors of the particle. This location is called *lbest*. when a particle takes all the population as its topological neighbors, the best value is a global best and is called *gbest*.

The particle swarm optimization concept consists of, at each time step, changing the velocity of (accelerating) each particle toward its p*best* and *lbest* locations (local version of PSO). Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward *pbest* and *lbest* locations. In past several years, PSO has been successfully applied in many research and application areas. It is demonstrated that PSO gets better results in a faster, cheaper way compared with other methods.

4.2 Notations

The position of the i-th particle of a swarm of size n, is represented by the D-dimensional vector, $xi= (xi1, xi2, ...,xiD)$

The best previous position (i.e., the position giving the best function value) of the i-th particle is recorded and represented by *pi*= (ρ*i*1, ρ*i*2, ...,ρ*iD*).

The position change (velocity) of the i-th particle is *Veli*=(*Veli*1, *Veli*2, ..., *VeliD*).

The position of the best particle of the swarm (i.e., the particle with the smallest function value) is denoted by index *pg*[3].

The particles are then manipulated according to the following equations.

$$Vel_{id}(t+1)= \chi\{w\ Vel_{id}(t)+c_1\varphi_1[\rho_{id}(t)-x_{id}(t)]+ c_2\varphi_2 [\rho_{gd}(t)-x_{id}(t)]\} \quad (1)$$

$$x_{id}(t+1)= x_{id}(t)+ Vel_{id}(t+1) \quad (2)$$

where d=1,2,…..D and i=1,2,…..n.

w: inertia weight=0.72

c1,c2 : positive acceleration constants=1.49

φ_1,φ_2: random numbers

χ : constriction factor=1.0

4.3. Algorithm

```
for each particle:
        Initialize particle: $x_i$
For each particle:
        Calculate fitness value: $p_i$
        If the fitness value is better than the best fitness value ($p_g$) in History
        Set current value as the new fitness value.
End
For each particle:
Find in the particle neighborhood, the particle with the best fitness
Calculate particle velocity: Veli ,using equation(1)
Apply the velocity constriction.
Update particle position: xi , using equation (2)
Apply the position constriction.
```

4.4 Problems with PSO

- Requires much more iteration (generally more than 500 iterations) to converge to the optima than the K-mean algorithm does.
- Computation requirement for clustering extremely huge document datasets is still high.
- In terms of execution time, the K-means algorithm is the most efficient for the large dataset.

4.5 Applications:-

These are the applications of Particle Swarm Optimization Algorithm.

- In computational biology and bioinformatics
- In medical imaging, such as PET scans
- Market research
- Educational Research
- Segmenting the market and determining target markets
- Product positioning
- New product development
- Selecting test markets

5. HYBRID K-MEANS AND PARTICLE SWARM OPTIMIZATION (KPSO)

5.1. Introduction

The major problem with K-means algorithm is that it is sensitive to the selection of the initial partition and may converge to local optima. In this module, we present a hybrid Particle Swarm Optimization K-means and PSO data clustering algorithm that performs fast data clustering and can avoid being trapped in a local optimal solution. Here the output of K-means algorithm is given as input for particle swarm optimization to inherit its properties and avoid local optimal solution.

5.2 Problem with KPSO

Though it is local optimal solution its performance is not satisfactory when compared to K-means algorithm. Computing the total number of clusters and their positions is difficult in KPSO. This computation takes more time therefore to reduce the overall time KPSO has to be merged with some partitioning algorithm.

5.3 Advantages

- Avoids local optima.
- It converges faster when compared to other optimization algorithms.

5.4 K-Means + PSO +K-Means

To increase the performance of KPSO algorithm another new hybrid version of K-means and PSO is implemented. In this algorithm, the output of KPSO is given as input to K-means which provides high performance when compared to other versions of K-means. In the hybrid KPSO+K-means algorithm, the multidimensional document vector space is modelled as a problem space. Each term in the document dataset represents one dimension of the problem space. Each document vector can be represented as a dot in the problem space. The whole document dataset can be represented as a multiple dimension space with a large number of dots in the space. The hybrid KPSO+Kmeans algorithm includes two modules, the PSO module and K-means module. At the initial stage, the PSO module is executed for a short period to search for the clusters' centroid locations. The locations are transferred to the K-means module for refining and generating the final optimal clustering solution.

5.5 K-Means + PSO +K-Means algorithm

(1) Inheriting cluster centroid vectors from the PSO module.

(2) Assigning each document vector to the closest cluster centroids.

(3) Recalculating the cluster centroid vector c_j using equation 8..

(4) Repeating step 2 and 3 until the convergence is achieved.

In the K-means+PSO+K-means algorithm, the ability of globalized searching of the PSO algorithm and the fast convergence of the K-means algorithm are combined. The KPSO algorithm is used at the initial stage to help discovering the vicinity of the optimal solution by a global search. The result from KPSO is used as the initial seed of the K-means algorithm, which is applied for refining and generating the final result.

5.6 Comparisons of K-means, KPSO and KPSOK

The three versions of K-means use the following parameters for comparison. They are:

- Minimum and Maximum values
- Fitness
- Overall Purity
- Elapsed time

From all these comparisons the final estimated result is that the hybrid combination of K-means, PSO, and K-means (KPSOK) gives high quality clustering with best performance.

6. SYSTEM DESIGN

The Design phase includes the description of scope of the project using the Unified Modeling Language (UML) diagrams. The following were the diagrams:

1. Use Case Diagrams
2. Sequence Diagrams
3. Component Diagrams
4. Collaboration Diagrams

6.1 Use case diagram

K-Means:

In this use case diagram use cases such as select the data set, select no of clusters, KPSO and KPSOK, calculate Fitness, Time elapsed and calculate Purity and MIN-MAX values are taken.

When the datasets are given as inputs to the algorithms, number of clusters should have to be chosen and the Time Elapsed, Fitness, Purity and MIN-MAX values are calculated.

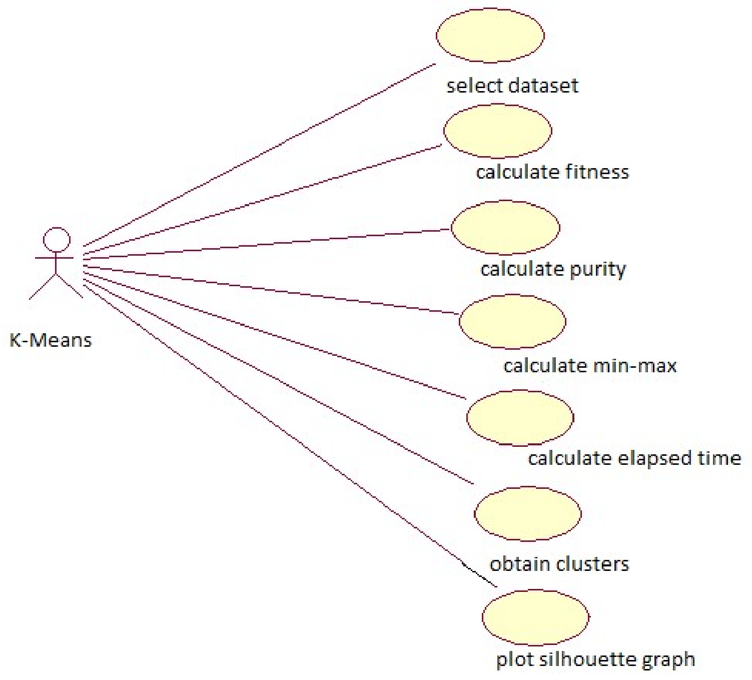


Fig 6.1.1: Use case diagram for K-Means

PSO:

In this use case diagram use cases such as select the data set, select no of clusters, calculate Fitness, Time elapsed and calculated global best, personal best, cluster centroid values are taken.

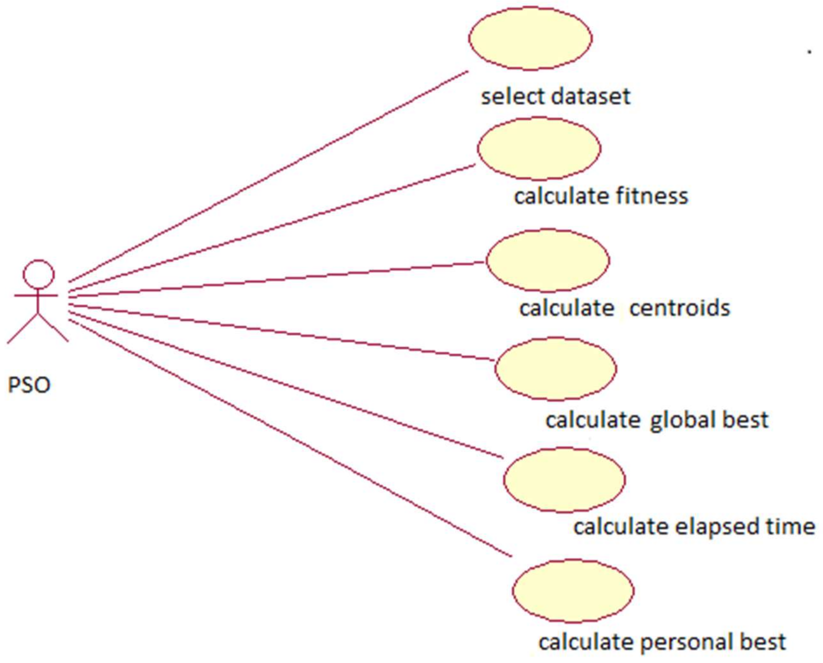


Fig 6.1.2: Use case diagram for PSO

Hybrid K-Means with PSO:

In this use case diagram use cases such as select the data set, obtain and compare the results through various parameters of K-Means and PSO modules.

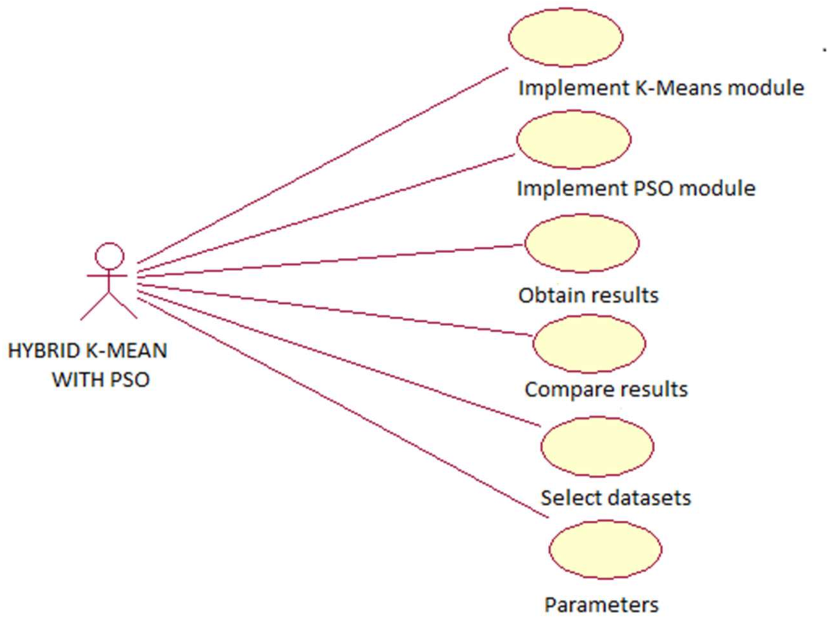


Fig 6.1.3: Use case diagram for Hybrid K-Means with PSO

6.2 Class Diagram:

This is a statistic structure diagram that describes the structure of a system by showing system classes like GUI, PSO, K-Means , their attributes like centroid, fitness, distance etc and operations. It connects these classes with suitable relationship.

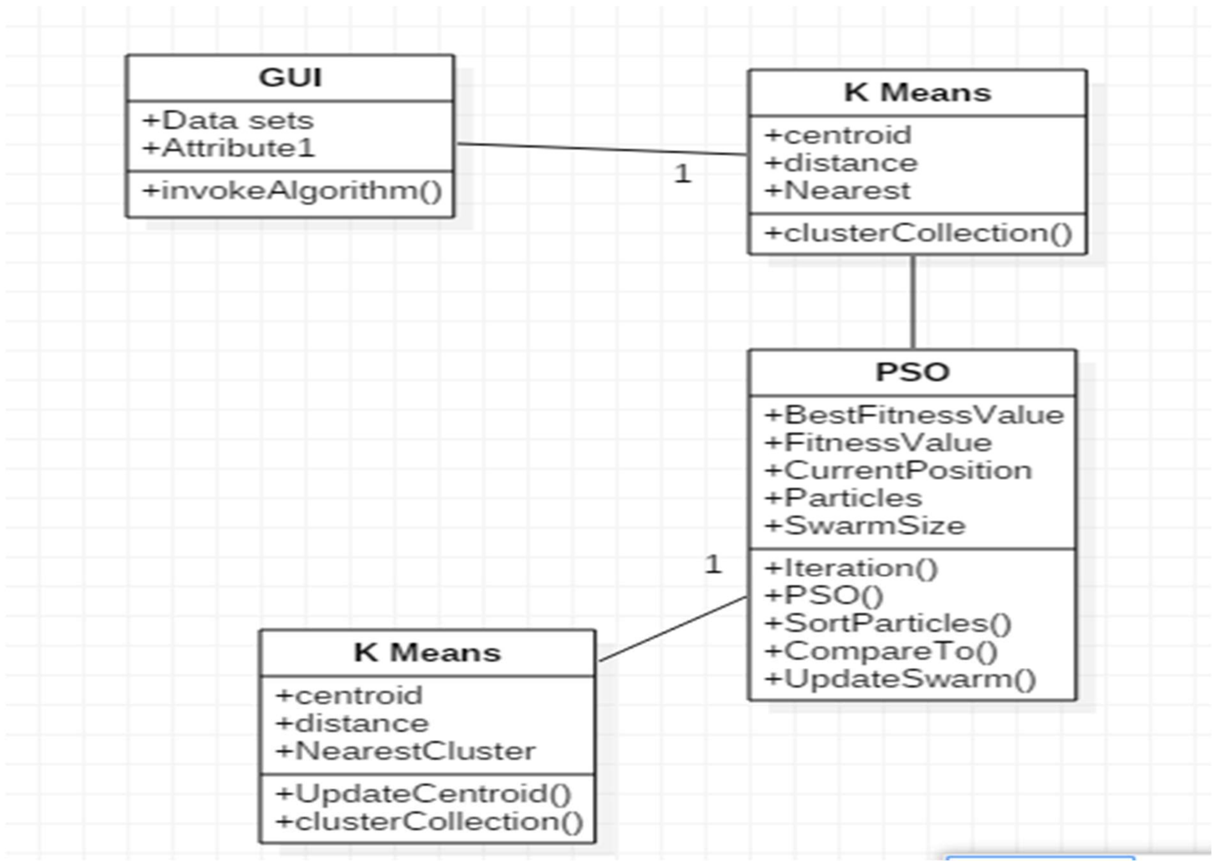


Fig 6.2.1: Class diagram for Hybrid K-Means with PSO

6.3 Sequence Diagram

K-Means:
In this Sequence diagram objects such as Select Dataset, select no of clusters, KPSO and KPSOK are taken.

When the datasets are given as inputs to the algorithms, number of clusters should have to be chosen and the Time Elapsed, Fitness, Purity and MIN-MAX values are calculated.

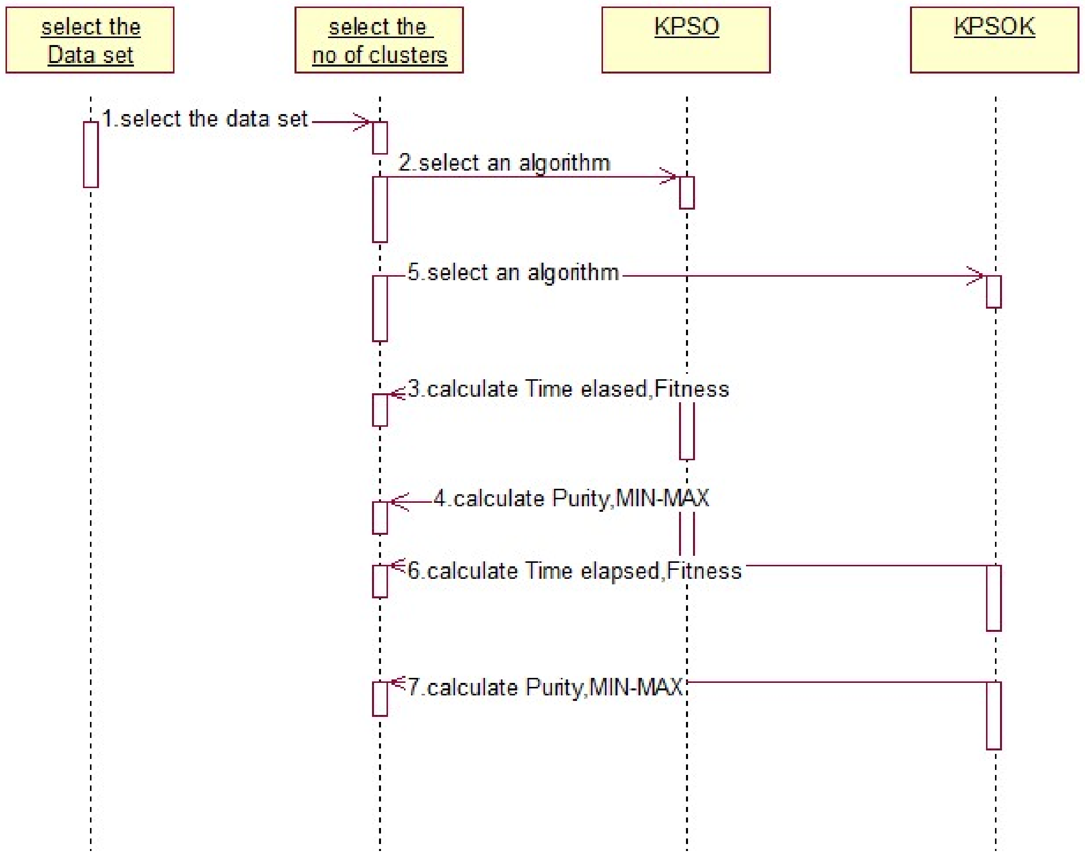

select the Data set
select the no of clusters
KPSO
KPSOK
1.select the data set
2.select an algorithm
5.select an algorithm
3.calculate Time elased,Fitness
4.calculate Purity,MIN-MAX
6.calculate Time elapsed,Fitness
7.calculate Purity,MIN-MAX

Fig 6.3.1: Sequence diagram for K-Means

PSO:

In this Sequence Diagram, objects such as Select the Dataset, Select no of clusters, PSO are taken.When the datasets are given as inputs to the algorithms, number of clusters should have to be chosen and the Time Elapsed and best Fitness values are calculated.

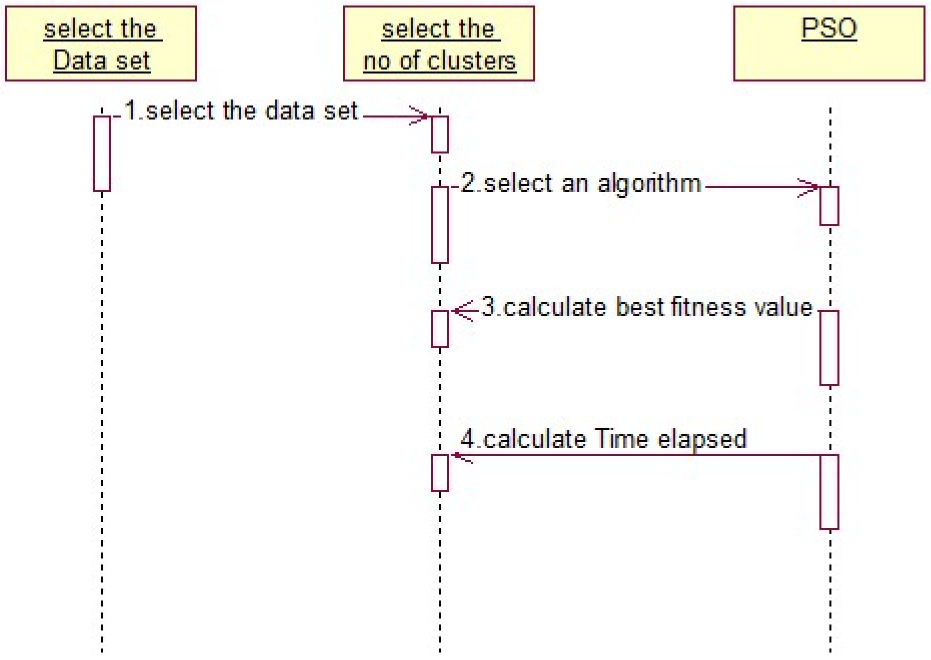


Fig 6.3.2: Sequence diagram for PSO

6.4: Activity Diagram

Activity diagrams are graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency.

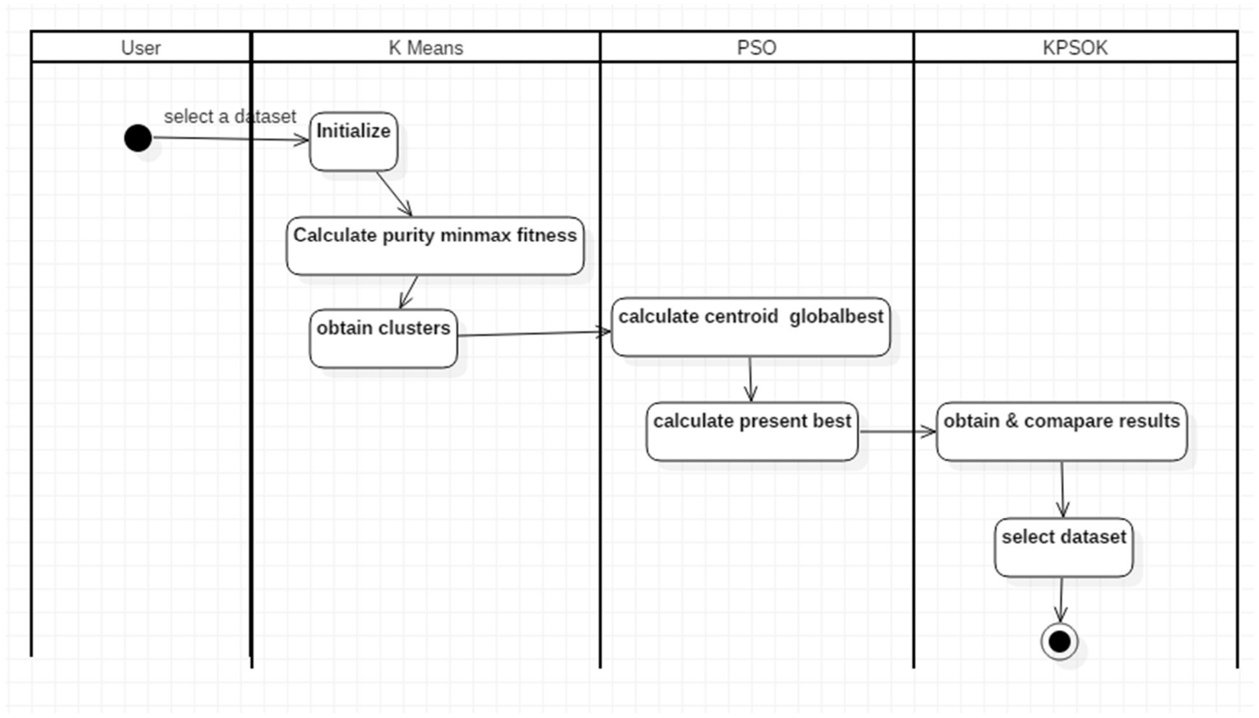


Fig 6.4.1: Class diagram for Hybrid K-Means with PSO

6.5 Collaboration Diagram

K-Means:

In this collaboration diagram objects such as Select Dataset, select no of clusters, KPSO and KPSOK are taken.

When the datasets are given as inputs to the algorithms, number of clusters should have to be chosen and the Time Elapsed, Fitness, Purity and MIN-MAX values are calculated.

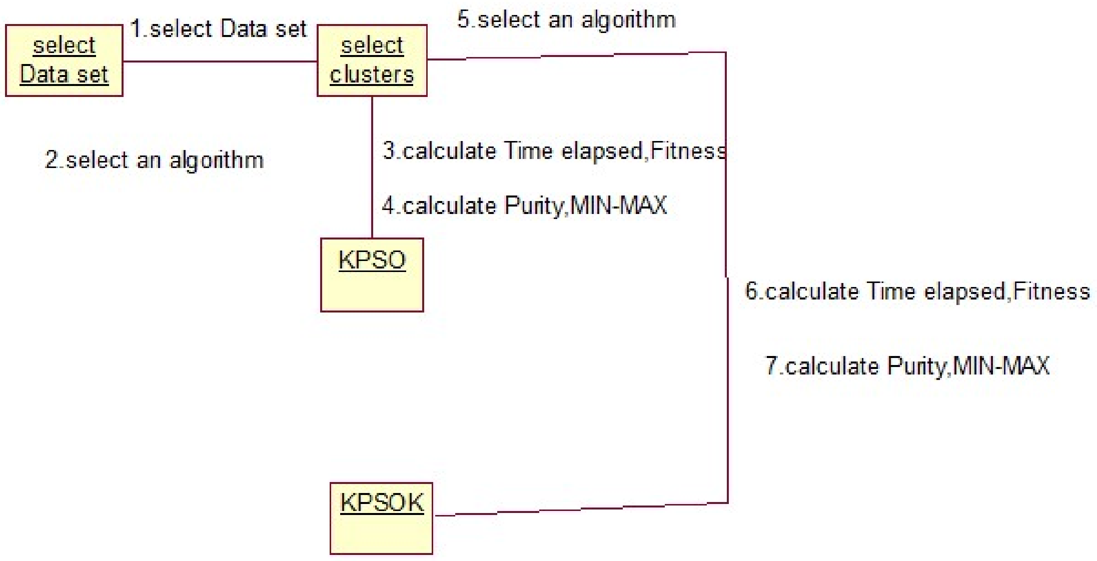

select Data set
1.select Data set
select clusters
5.select an algorithm
2.select an algorithm
3.calculate Time elapsed,Fitness
4.calculate Purity,MIN-MAX
KPSO
6.calculate Time elapsed,Fitness
7.calculate Purity,MIN-MAX
KPSOK

Fig 6.5.1: Collaboration diagram for K-Means

PSO:

In this Collaboration Diagram, objects such as Select the Dataset, Select no of clusters, PSO are taken.When the datasets are given as inputs to the algorithms, number of clusters should have to be chosen and the Time Elapsed and best Fitness values are calculated.

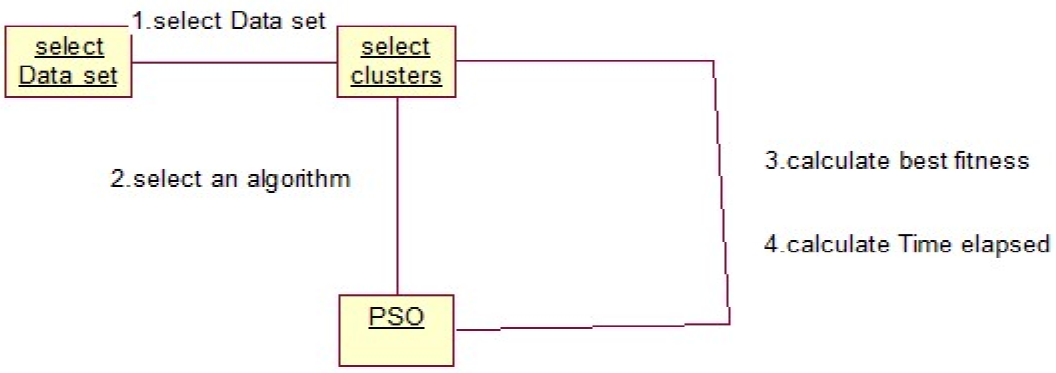


Fig 6.5.2: Collaboration diagram for PSO

6.6 Component Diagram:

In this component diagram different components such as data set, K-Means and PSO modules, results and their dependency relationships are presented.

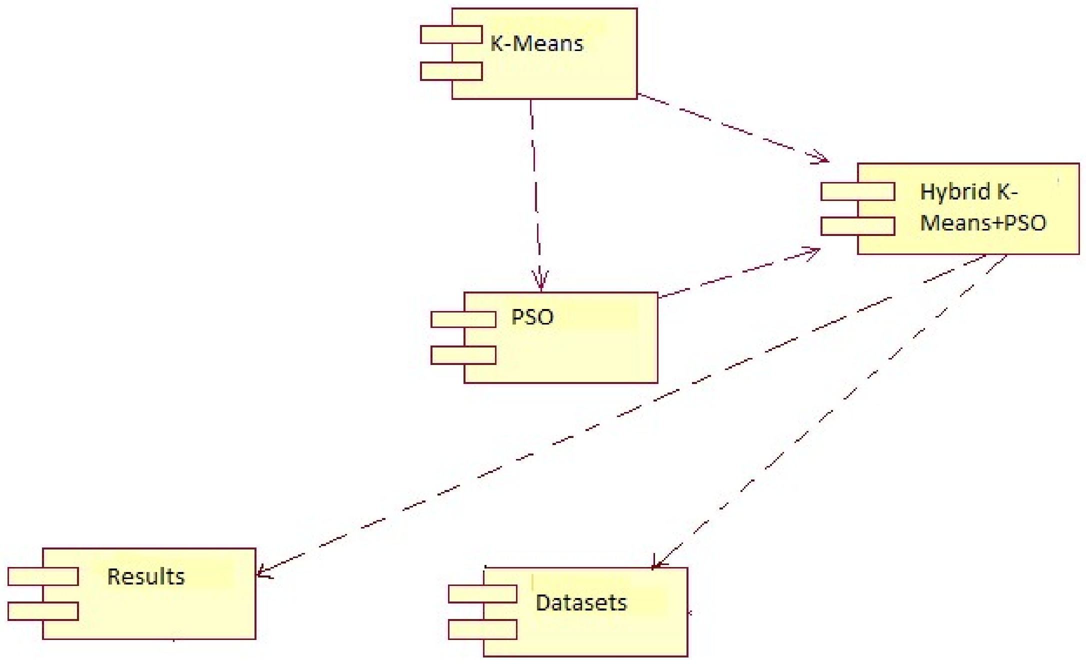


Fig 6.6.1: Component Diagram for hybrid K-Means with PSO

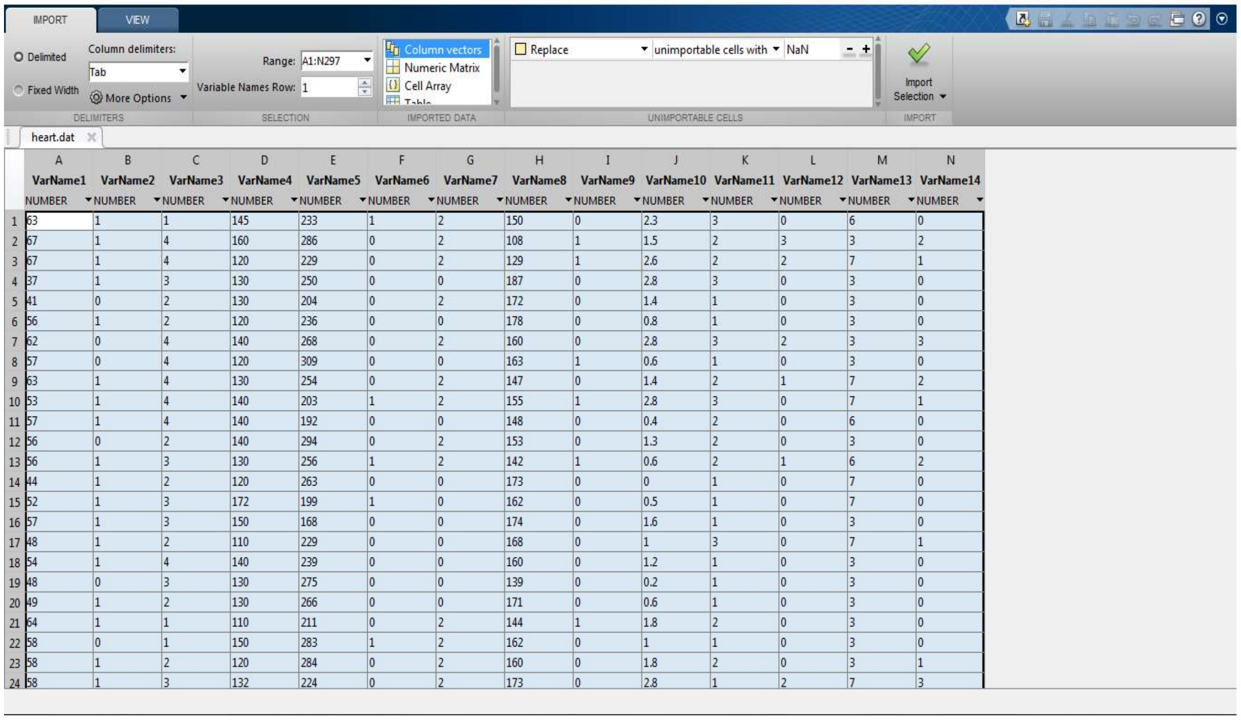
7. EXPERIMENTAL RESULTS

7.1 Introduction

Theoretical comparisons among all the algorithms has met with a result, the main goal of the experimental result is to match the theoretical result. Among the various combinations of K-Means and PSO Algorithms, KPSOK combination is showing the best result in theoretical aspect and also in experimental point of view. The below values are the solved and practical result that has been seen in this project.

Import Dataset : Heart Disease

Data import and export functions provide access to data from files, other applications, web services, and external devices. We can use popular file formats like, such as Microsoft excel spread sheets, text, scientific data format etc.



Delimited
Column delimiters: Tab
Range: A1:N297
Variable Names Row: 1
Column vectors
Replace unimportable cells with NaN

heart.dat

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	VarName1	VarName2	VarName3	VarName4	VarName5	VarName6	VarName7	VarName8	VarName9	VarName10	VarName11	VarName12	VarName13	VarName14
	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER
1	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
2	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
3	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
4	37	1	3	130	250	0	0	187	0	2.8	3	0	3	0
5	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
6	56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
7	62	0	4	140	268	0	2	160	0	2.8	3	2	3	3
8	57	0	4	120	309	0	0	163	1	0.6	1	0	3	0
9	63	1	4	130	254	0	2	147	0	1.4	2	1	7	2
10	53	1	4	140	203	1	2	155	1	2.8	3	0	7	1
11	57	1	4	140	192	0	0	148	0	0.4	2	0	6	0
12	56	0	2	140	294	0	2	153	0	1.3	2	0	3	0
13	56	1	3	130	256	1	2	142	1	0.6	2	1	6	2
14	44	1	2	120	263	0	0	173	0	0	1	0	7	0
15	52	1	3	172	199	1	0	162	0	0.5	1	0	7	0
16	57	1	3	150	168	0	0	174	0	1.6	1	0	3	0
17	48	1	2	110	229	0	0	168	0	1	3	0	7	1
18	54	1	4	140	239	0	0	160	0	1.2	1	0	3	0
19	48	0	3	130	275	0	0	139	0	0.2	1	0	3	0
20	49	1	2	130	266	0	0	171	0	0.6	1	0	3	0
21	64	1	1	110	211	0	2	144	1	1.8	2	0	3	0
22	58	0	1	150	283	1	2	162	0	1	1	0	3	0
23	58	1	2	120	284	0	2	160	0	1.8	2	0	3	1
24	58	1	3	132	224	0	2	173	0	2.8	1	2	7	3

Fig 7.1.1: Data Set

Dataset under study: Heart Disease

Data clustering is preceded by data cleaning. Data cleaning is a process used to determine inaccurate, incomplete or unreasonable data and then improve the quality through correcting of detected errors and omissions. The below figure illustrates the quality of data.

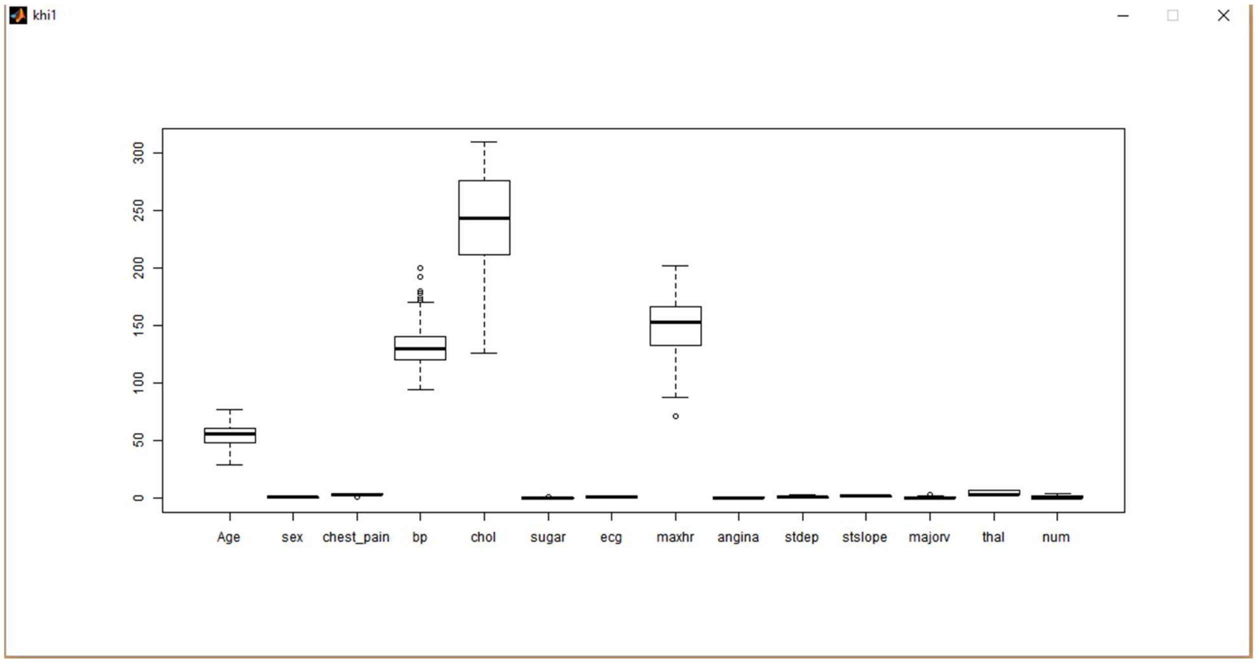


Fig 7.1.2: Box plot

The execution of various algorithms is shown along with the tables and graphs which represent the comparisons among the different algorithms.

7.2 GUI for Clustering

This GUI consists of data sets like Heart disease data set, Iris data set. The user selects any of the data set and apply algorithms on them. The result will be visualized in a graphical format.

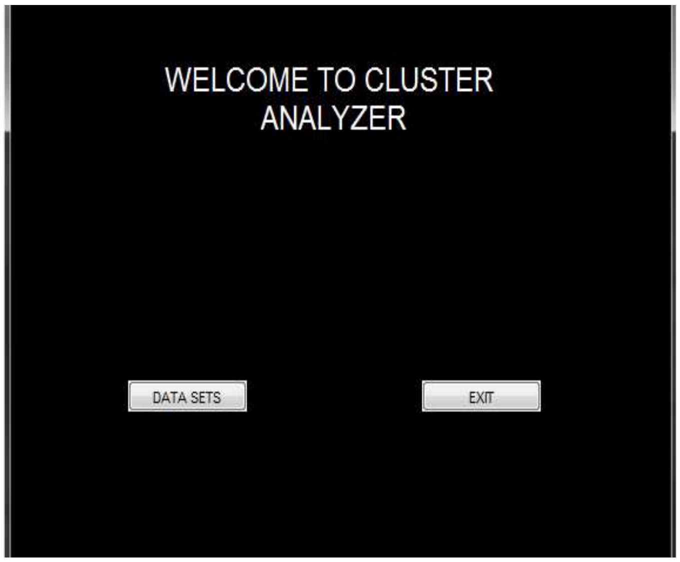


Fig 7.2.1: Welcome Page

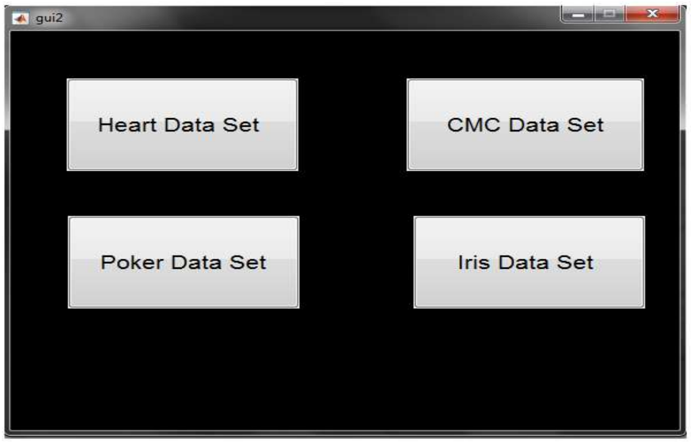


Fig 7.2.2: GUI for Data Set selection

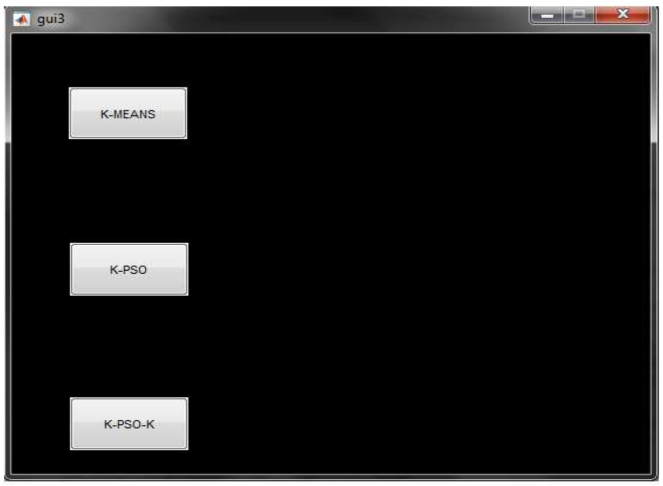
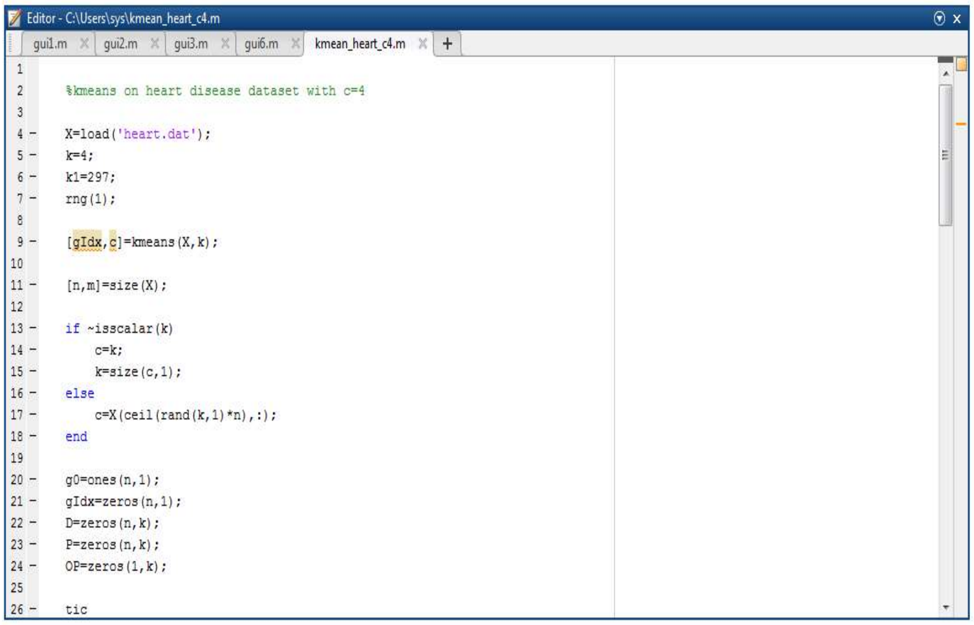


Fig 7.2.3: GUI for Algorithm selection

7.3 K-Means

The takes the data set and initializes the centroid and distance by default. It performs iterations using k-Means algorithm and updates centroid and other metrics simultaneously. The best arrangement out of all initializations is selected for further process.



```

%kmeans on heart disease dataset with c=4

X=load('heart.dat');
k=4;
k1=297;
rng(1);

[gIdx,c]=kmeans(X,k);

[n,m]=size(X);

if ~isscalar(k)
    c=k;
    k=size(c,1);
else
    c=X(ceil(rand(k,1)*n),:);
end

g0=ones(n,1);
gIdx=zeros(n,1);
D=zeros(n,k);
P=zeros(n,k);
OP=zeros(1,k);

tic
```

Fig 7.3.1: Algorithm for K-Means

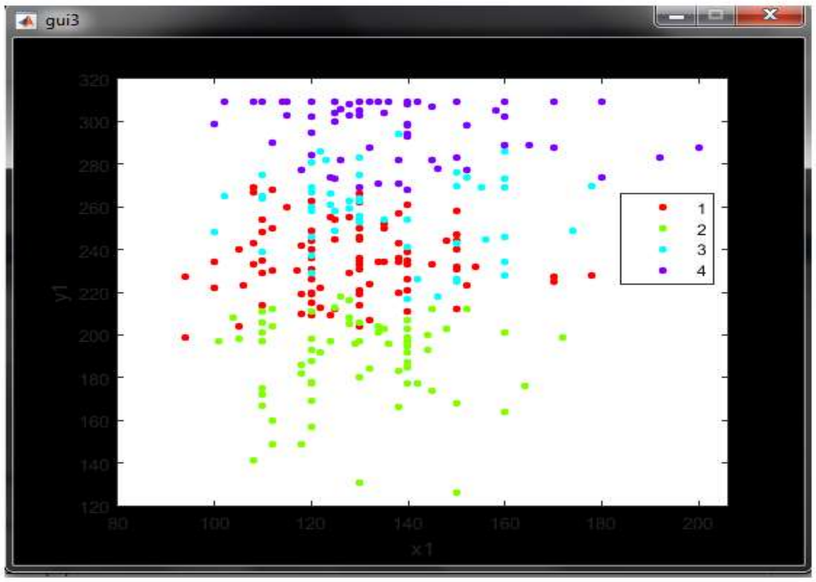
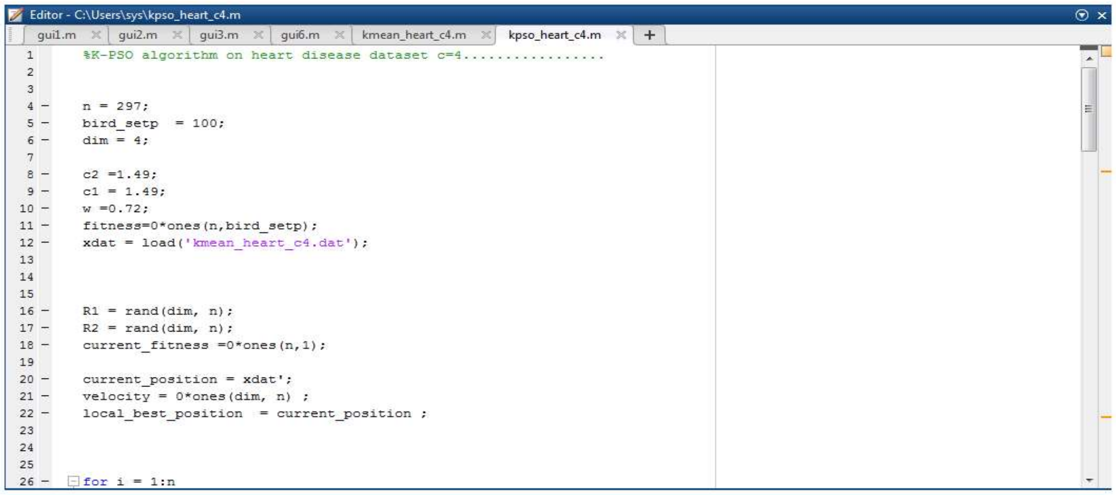


Fig 7.3.3: K-Means cluster

7.4 PSO



```
%K-PSO algorithm on heart disease dataset c=4................


n = 297;
bird_setp  = 100;
dim = 4;

c2 =1.49;
c1 = 1.49;
w =0.72;
fitness=0*ones(n,bird_setp);
xdat = load('kmean_heart_c4.dat');



R1 = rand(dim, n);
R2 = rand(dim, n);
current_fitness =0*ones(n,1);

current_position = xdat';
velocity = 0*ones(dim, n) ;
local_best_position  = current_position ;



for i = 1:n
```

Fig 7.4.1: PSO algorithm

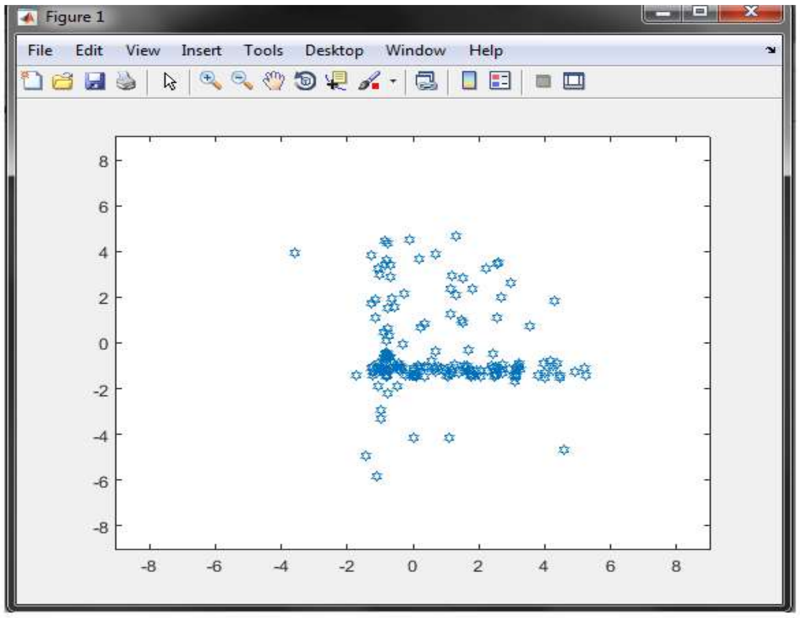
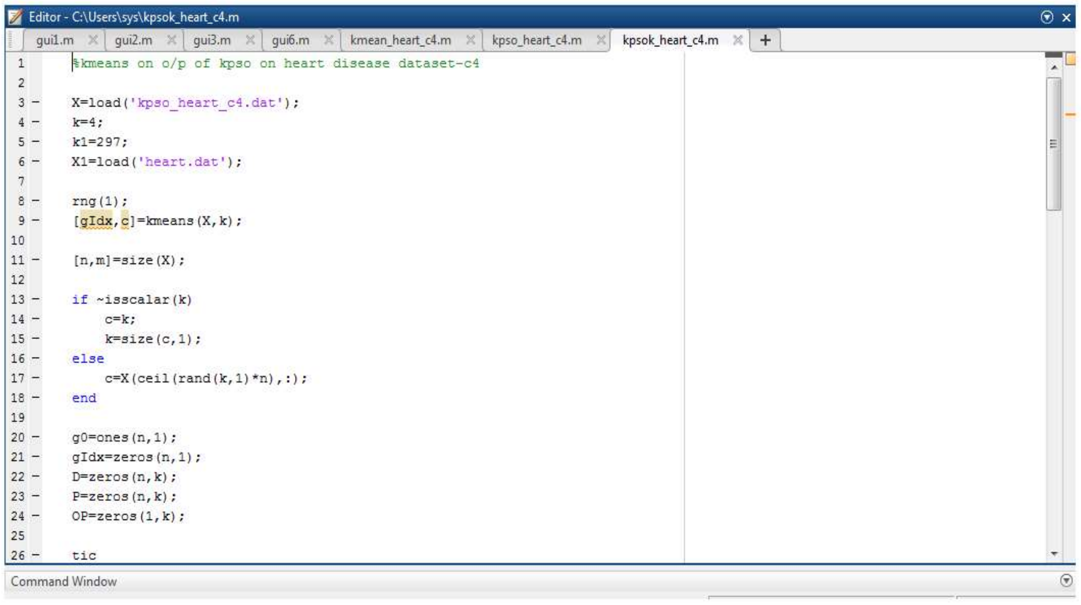


Fig 7.4.2: PSO algorithm execution

7.4 K-PSO-K



```
%kmeans on o/p of kpso on heart disease dataset-c4

X=load('kpso_heart_c4.dat');
k=4;
k1=297;
X1=load('heart.dat');

rng(1);
[gIdx,c]=kmeans(X,k);

[n,m]=size(X);

if ~isscalar(k)
    c=k;
    k=size(c,1);
else
    c=X(ceil(rand(k,1)*n),:);
end

g0=ones(n,1);
gIdx=zeros(n,1);
D=zeros(n,k);
P=zeros(n,k);
OP=zeros(1,k);

tic
```

Fig 7.5.1: Algorithm for KPSOK

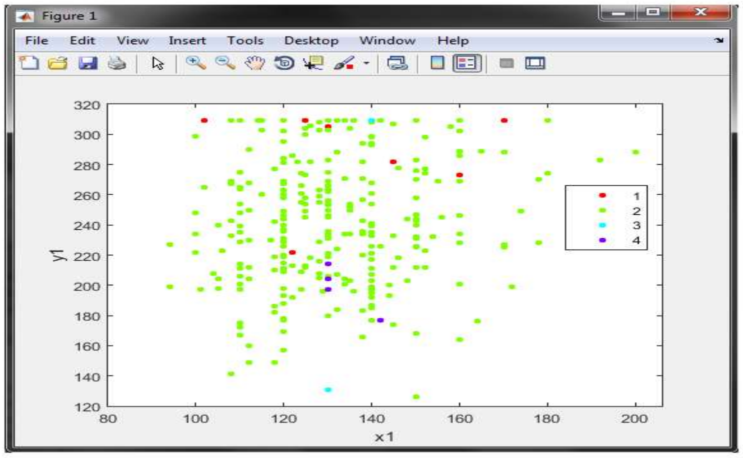


Fig 7.5.2: Cluster for KPSOK

The above figure illustrates the resultant clusters formed after applying KPSOK algorithm on the heart disease dataset.

7.6 Comparisions

The results obtained for the 6 datasets under study were tabulated and compared as shown below. The parameters under consideration were purity, fitness, time and minimuma and maximum values.

K-Means					
	Purity	Min value	Max value	Fitness	Time
Heart Disease	42.8804	6.6705	171.5218	9.2323	0.085835
Online News Popularity	4.20E+05	6.76E+03	1.68E+06	181.6116	39.05932
GPS Trajectories	1.27E+04	5.9774	3.81E+04	10.7632	0.068326
Seeds	3.4639	0.2402	10.3916	4.7617	0.041063
Wholesale Customer Data	3.63E+04	2.03E+03	1.09E+05	2.11E+03	0.066719
Prostate Cancer	6.59E+03	2.30E+03	2.64E+04	5.9277	0.710561

Table 7.6.1: Comparision table for K-Means

From the above results it can be concluded that Seeds dataset yields best results as it has the least OFV and elapsed time.As the Online News Popularity dataset shows contrasting results in OFV and time it is least suitable to use K-Means for this dataset.

	OFV	Time
Heart Disease	21.6638	33.14016
Online News Popularity	2.33E+04	165.2719
GPS Trajectories	1.57E+03	32.90393
Seeds	2.0879	33.02583
Wholesale Customers Data	2.81E+03	33.20557
Prostate Cancer	0.4511	155.5463

Table 7.6.4: Comparision table for PSO

The above table shows that GPS Trajectories dataset has least fitness and elapsed time for the PSO algorithm when compared to the other 5 datasets under study.

KPSO		
	OFV	Time
Heart Disease	71.5654	14.73
Online News Popularity	8.32E+05	56.90647
GPS Trajectories	0.0121	12.71791
Seeds	0.6942	13.13221
Wholesale Customers Data	0.2433	13.02499
Prostate Cancer	0.7433	12.71107

Table 7.6.5: Comparision table for KPSO

The above table shows that GPS Trajectories dataset has least fitness and elapsed time for the KPSO algorithm when compared to the other 5 datasets under study.

KPSO-K					
	Purity	Max value	Min value	Fitness	Time
Heart Disease	31.7235	126.894	0.8054	1.3188	0.182718
Online News Popularity	8.95E+05	3.58E+06	263.7914	1.4739	9.58221
GPS Trajectories	1.5461	4.6384	0.0225	1.4258	0.128748
Seeds	1.7073	5.1219	0.0507	1.039	0.151915
Wholesale Customer Data	0.2589	0.7766	8.08E-04	0.1978	0.149529
Prostate Cancer	0.7705	3.0819	0.0112	1.3005	0.110385

Table 7.6.6: Comparision table for KPSOK

From the above results it can be concluded that Wholesale Customer Data yields best results as it has the least OFV and elapsed time.As the Online News Popularity dataset executes for a longer time,it is least suitable to use KPSOK for this dataset.

	K-Means					KPSO-K				
	Purity	Min value	Max value	Fitness	Time	Purity	Max value	Min value	Fitness	Time
Heart Disease	42.8804	6.6705	171.5218	9.2323	0.085835	31.7235	126.894	0.8054	1.3188	0.182718
Online News Popularity	4.20E+05	6.76E+03	1.68E+06	181.6116	39.05933	8.95E+05	3.58E+06	263.7914	0.4739	9.58221
GPS Trajectories	1.27E+04	5.9774	3.81E+04	-0.7632	0.028326	1.5461	4.6384	0.0225	1.4258	0.128748
Seeds	3.4639	0.2402	10.3916	4.7617	0.041063	1.7073	5.1219	0.0507	1.039	0.151915
Wholesale Customer Data	3.63E+04	2.03E+03	1.09E+05	2.11E+03	0.066719	0.2589	0.7766	8.08E-04	0.1978	0.149529
Prostate Cancer	6.59E+03	2.30E+03	2.64E+04	-5.9277	0.710561	0.7705	3.0819	0.0112	1.3005	0.110385

Table 7.6.7: Comparision table for K-Means and KPSOK

From the above comparision,it can be concluded that KPSOK yields better results in terms of fitness and time than K-Means.

<u>Graphical comparisions of K-PSO-K and K-Means in terms of purity,fitness and time:</u>

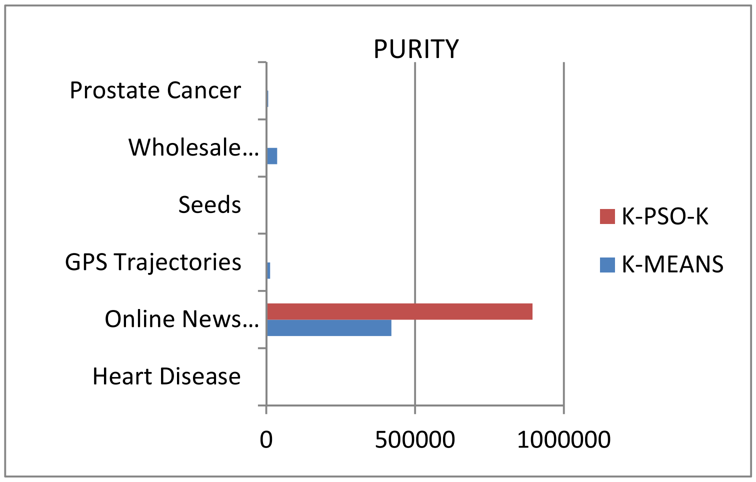


Fig 7.5.1: Purity(K-Means and K-PSO-K)

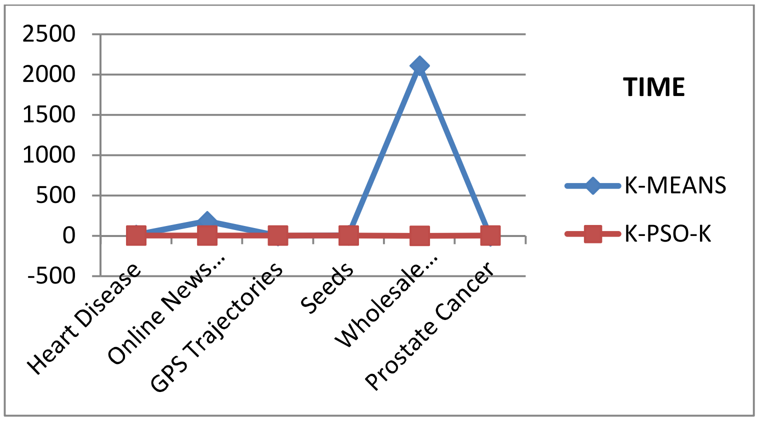


Fig 7.5.2: Time(K-Means and K-PSO-K)

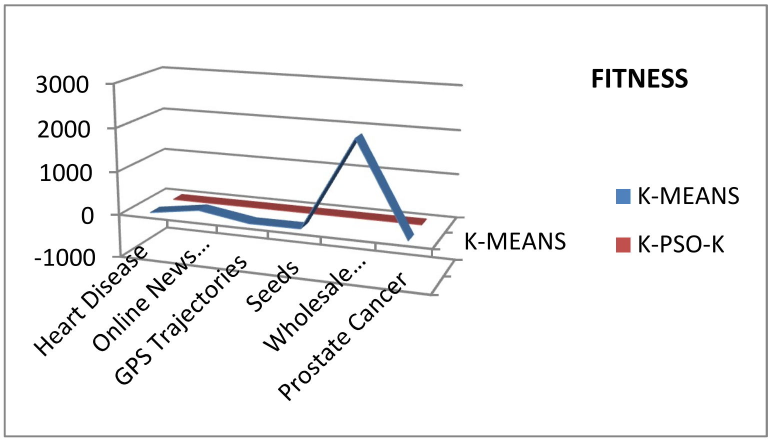


Fig 7.5.3: Fitness(K-Means and K-PSO-K)

	PSO		KPSO	
	OFV	Time	OFV	Time
Heart Disease	21.6638	33.14016	71.5654	14.73
Online News Popularity	2.33E+04	165.2719	8.32E+05	56.90647
GPS Trajectories	1.57E+03	32.90393	0.0121	12.71791
Seeds	2.0879	33.02583	0.6942	13.13221
Wholesale Customers Data	2.81E+03	33.20557	0.2433	13.02499
Prostate Cancer	0.4511	155.5463	0.7433	12.71107

Table 7.5.6: Comparision table for PSO and KPSO

Graphical comparisions of PSO and KPSO in terms of fitness and time:

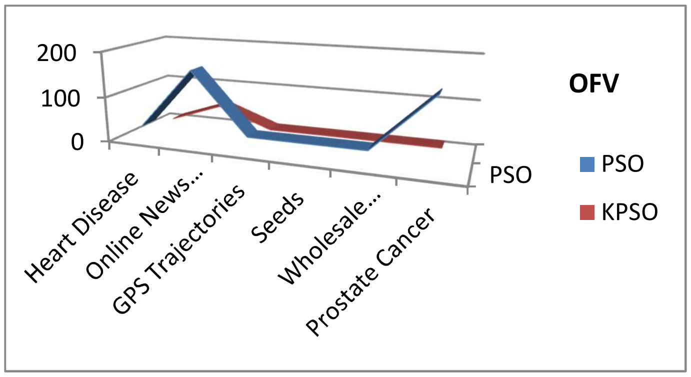


Fig 7.5.4: Fitness(PSO and KPSO)

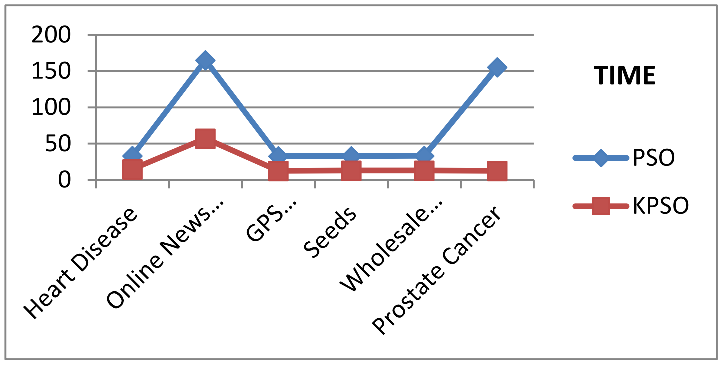


Fig 7.5.5: Time(PSO and KPSO)

From the above comparisions,it can be concluded that KPSO algorithm performs better than PSO algorithm as it has least OFV and elapsed time.

CODE

GUI-1

```
GUI1('Property','Value',...) creates a new GUI1 or raises the existing
singleton*.  Starting from the left, property value pairs are applied
to the GUI before gui1_OpeningFcn gets called.  An unrecognized
property name or invalid value makes property application stop.  All
inputs are passed to gui1_OpeningFcn via varargin.



function varargout = gui1(varargin)
% GUI1 MATLAB code for gui1.fig
%      GUI1, by itself, creates a new GUI1 or raises the existing
%      singleton*.
%
%      H = GUI1 returns the handle to a new GUI1 or the handle to
%      the existing singleton*.
%
%      GUI1('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI1.M with the given input
arguments.
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui1_OpeningFcn, ...
                   'gui_OutputFcn',  @gui1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before gui1 is made visible.
function gui1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gui1 (see VARARGIN)

% Choose default command line output for gui1
handles.output = hObject;

% Update handles structure
```

```
guidata(hObject, handles);

% UIWAIT makes gui1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = gui1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
closereq;
gui2;

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
closereq;
```

GUI-2

```
function varargout = gui2(varargin)
% GUI2 MATLAB code for gui2.fig
%      GUI2, by itself, creates a new GUI2 or raises the existing
%      singleton*.

%      H = GUI2 returns the handle to a new GUI2 or the handle to
%      the existing singleton*.

%      GUI2('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI2.M with the given input
arguments.
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui2_OpeningFcn, ...
                   'gui_OutputFcn',  @gui2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before gui2 is made visible.
function gui2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gui2 (see VARARGIN)

% Choose default command line output for gui2
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes gui2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = gui2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
closereq;
gui3;


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
closereq;
gui6;


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
closereq;
gui5;


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data
closereq;
gui4;
```

GUI-3

```
function varargout = gui3(varargin)
% GUI3 MATLAB code for gui3.fig
%      GUI3, by itself, creates a new GUI3 or raises the existing
%      singleton*.
%
%      H = GUI3 returns the handle to a new GUI3 or the handle to
%      the existing singleton*.
%
%      GUI3('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI3.M with the given input
arguments.
%
%      GUI3('Property','Value',...) creates a new GUI3 or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before gui3_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to gui3_OpeningFcn via varargin.

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui3_OpeningFcn, ...
                   'gui_OutputFcn',  @gui3_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% --- Executes just before gui3 is made visible.
function gui3_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gui3 (see VARARGIN)

% Choose default command line output for gui3
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes gui3 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = gui3_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 kmean_heart_c4;



% --- Executes on button press in pushbutton2.
```

```
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
kpso_heart_c4;




% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
kpsok_heart_c4;
```

GUI-4

```
function varargout = gui4(varargin)
% GUI4 MATLAB code for gui4.fig
%      GUI4, by itself, creates a new GUI4 or raises the existing
%      singleton*.
%
%      H = GUI4 returns the handle to a new GUI4 or the handle to
%      the existing singleton*.
%
%      GUI4('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI4.M with the given input
arguments.
%
%      GUI4('Property','Value',...) creates a new GUI4 or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before gui4_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to gui4_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help gui4
% Last Modified by GUIDE v2.5 15-Apr-2017 00:23:03

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui4_OpeningFcn, ...
                   'gui_OutputFcn',  @gui4_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before gui4 is made visible.
function gui4_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gui4 (see VARARGIN)

% Choose default command line output for gui4
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes gui4 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = gui4_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 kmean_iris4;



% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
kpso_iris4;



% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
kpsok_iris4;
```

GUI-5

```
function varargout = gui5(varargin)
% GUI5 MATLAB code for gui5.fig
%      GUI5, by itself, creates a new GUI5 or raises the existing
%      singleton*.
%
%      H = GUI5 returns the handle to a new GUI5 or the handle to
%      the existing singleton*.
%
%      GUI5('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI5.M with the given input
arguments.
%
%      GUI5('Property','Value',...) creates a new GUI5 or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before gui5_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to gui5_OpeningFcn via varargin.
%
```

```
%       *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only 
one
%       instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help gui5

% Last Modified by GUIDE v2.5 15-Apr-2017 00:39:03

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui5_OpeningFcn, ...
                   'gui_OutputFcn',  @gui5_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before gui5 is made visible.
function gui5_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gui5 (see VARARGIN)

% Choose default command line output for gui5
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes gui5 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = gui5_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 kmean_cmc;



% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
kpso_cmc;




% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
kpsok_cmc;
```

GUI-6

```
function varargout = gui6(varargin)
% GUI6 MATLAB code for gui6.fig
%      GUI6, by itself, creates a new GUI6 or raises the existing
%      singleton*.
%
%      H = GUI6 returns the handle to a new GUI6 or the handle to
%      the existing singleton*.
%
%      GUI6('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI6.M with the given input
arguments.
%
%      GUI6('Property','Value',...) creates a new GUI6 or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before gui6_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to gui6_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help gui6

% Last Modified by GUIDE v2.5 15-Apr-2017 00:39:23

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui6_OpeningFcn, ...
                   'gui_OutputFcn',  @gui6_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before gui6 is made visible.
function gui6_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gui6 (see VARARGIN)

% Choose default command line output for gui6
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes gui6 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = gui6_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 kmean_poker;



% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
kpso_poker;




% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

kpsok_poker;
```

POKER DATASET

K-MEANS:

```
X=load('poker.dat');
k=4;
k1=500;
rng(1);

[gIdx,c]=kmeans(X,k);

[n,m]=size(X);

if ~isscalar(k)
    c=k;
    k=size(c,1);
else
    c=X(ceil(rand(k,1)*n),:);
end

g0=ones(n,1);
gIdx=zeros(n,1);
D=zeros(n,k);
P=zeros(n,k);
OP=zeros(1,k);

tic


while any(g0~=gIdx)

    g0=gIdx;

    for t=1:k
        d=zeros(n,1);

        for s=1:m
            d=d+(X(:,s)-c(t,s)).^2;
        end
        D(:,t)=sqrt(d);

        x=D(:,1);
        y=D(:,2);
        x1=X(:,2);
        y1=X(:,11);

       clf
     %   plot(x, y , 'o', x1, y1, '*')
     %   axis([-50 50 -50 50]);
     %   pause(.10)
    end

    [z,gIdx]=min(D,[],2);

    for t=1:k
        c(t,:)=mean(X(gIdx==t,:));
    end
end

for j=1:n
     p=max(D);
     P=p/D(j);
end

for m=1:k
     for j=1:n
         OP=D(j)*P/k;
     end
end

%fitness evaluation
for i=1:k
  for j=1:n
     f1=X(i)-D(j)/n;
  end
end

f=f1/k;

maxv=max(D);
minv=min(D);

disp('CLUSTER center matrix='); disp(D); %cluster center matrix
disp('MIN value ='); disp(min(minv)); %minimum value
disp('MAX value ='); disp(max(maxv)); %maximum value
disp('MAX Purity overall ='); disp(max(OP)); %maximum purity-overall
max purity value
disp('Fitness ='); disp(f); %purity

toc

gscatter(x1,y1,gIdx);
```

KPSO:

```
clear
clc
n = 500;
bird_setp  = 100;
dim = 4;

c2 =1.2;
c1 = 1.2;
w =0.9;
fitness=0*ones(n,bird_setp);
xdat = load('kmean_poker.dat');

                                       %    initialize the parameter %



R1 = rand(dim, n);
R2 = rand(dim, n);
current_fitness =0*ones(n,1);



                                 % Initializing swarm and velocities
and position %


current_position = 10*(rand(dim, n)-.5);
velocity = .3*randn(dim, n) ;
local_best_position  = current_position ;




                                 %     Evaluate initial population
%

for i = 1:n
    current_fitness(i) = Live_fn(current_position(:,i));
end


local_best_fitness  = current_fitness ;
[global_best_fitness,g] = min(local_best_fitness) ;

for i=1:n
    globl_best_position(:,i) = local_best_position(:,g) ;
end
                                               %  VELOCITY UPDATE  %
```

```
velocity = w *velocity + c1*(R1.*(local_best_position-
current_position)) + c2*(R2.*(globl_best_position-current_position));


                                              %   SWARMUPDATE    %



current_position = current_position + velocity ;


                                              %  evaluate anew swarm
%



%% Main Loop
tic
iter = 0 ;         % Iterationscounter
while  ( iter < bird_setp )
iter = iter + 1;

for i = 1:n,
current_fitness(i) = Live_fn(current_position(:,i)) ;

end


for i = 1 : n
        if current_fitness(i) < local_best_fitness(i)
           local_best_fitness(i)  = current_fitness(i);
           local_best_position(:,i) = current_position(:,i)   ;
        end
 end


 [current_global_best_fitness,g] = min(local_best_fitness);


if current_global_best_fitness < global_best_fitness
   global_best_fitness = current_global_best_fitness;

    for i=1:n
        globl_best_position(:,i) = local_best_position(:,g);
    end

end


 velocity = w *velocity + c1*(R1.*(local_best_position-
current_position)) + c2*(R2.*(globl_best_position-current_position));
 current_position = current_position + velocity;
```

```
x=current_position(1,:);
y=current_position(2,:);

clf
    plot(x, y , 'h')
    axis([-5 5 -5 5]);

pause(.2)


end
cp=current_position';% end of while loop its mean the end of all step
that the birds move it


              [Jbest_min,I] = min(current_fitness) % minimum fitness
               current_position(:,I) % best solution
disp(cp);

toc
```

KPSOK:

```
X=load('kpso_poker.dat');
k=4;
k1=500;
X1=load('poker.dat');

rng(1);
[gIdx,c]=kmeans(X,k);

[n,m]=size(X);

if ~isscalar(k)
    c=k;
    k=size(c,1);
else
    c=X(ceil(rand(k,1)*n),:);
end

g0=ones(n,1);
gIdx=zeros(n,1);
D=zeros(n,k);
P=zeros(n,k);
OP=zeros(1,k);

tic


while any(g0~=gIdx)

    g0=gIdx;

    for t=1:k
        d=zeros(n,1);

        for s=1:m
            d=d+(X(:,s)-c(t,s)).^2;
        end
        D(:,t)=sqrt(d);
        x=D(:,1);
        y=D(:,2);
        x1=X1(:,2);
        y1=X1(:,11);

       clf

    end

    [z,gIdx]=min(D,[],2);

    for t=1:k
```

```
        c(t,:)=mean(X(gIdx==t,:));
    end
end

for j=1:n
     p=max(D);
     P=p/D(j);
end

for m=1:k
     for j=1:n
         OP=D(j)*P/k;
     end
end

%fitness evaluation
for i=1:k
  for j=1:n
     f1=X(i)-D(j)/n;
  end
end

f=f1/k;

maxv=max(D);
minv=min(D);

disp('CLUSTER center matrix='); disp(D); %cluster center matrix
disp('MIN value ='); disp(min(minv)); %minimum value
disp('MAX value ='); disp(max(maxv)); %maximum value
disp('MAX Purity overall ='); disp(max(OP)); %maximum purity-overall
max purity value
disp('Fitness ='); disp(f); %purity

toc


gscatter(x1,y1,gIdx);
```

HEART DISESES DATA SET

K-MEANS:

```
X=load('heart.dat');
k=4;
k1=297;
rng(1);

[gIdx,c]=kmeans(X,k);

[n,m]=size(X);

if ~isscalar(k)
    c=k;
    k=size(c,1);
else
    c=X(ceil(rand(k,1)*n),:);
end

g0=ones(n,1);
gIdx=zeros(n,1);
D=zeros(n,k);
P=zeros(n,k);
OP=zeros(1,k);

tic


while any(g0~=gIdx)

    g0=gIdx;

    for t=1:k
        d=zeros(n,1);

        for s=1:m
            d=d+(X(:,s)-c(t,s)).^2;
        end
        D(:,t)=sqrt(d);

        x=D(:,1);
        y=D(:,2);
        x1=X(:,4);
        y1=X(:,5);

       clf
```

```
      %   plot(x, y , 'o', x1, y1, '*')
      %   axis([-50 50 -50 50]);

      %   pause(.10)
    end

    [z,gIdx]=min(D,[],2);

    for t=1:k
        c(t,:)=mean(X(gIdx==t,:));
    end
end

for j=1:n
     p=max(D);
     P=p/D(j);
end

for m=1:k
     for j=1:n
         OP=D(j)*P/k;
     end
end

%fitness evaluation
for i=1:k
  for j=1:n
     f1=X(i)-D(j)/n;
  end
end

f=f1/k;

maxv=max(D);
minv=min(D);

disp('CLUSTER center matrix='); disp(D); %cluster center matrix
disp('MIN value ='); disp(min(minv)); %minimum value
disp('MAX value ='); disp(max(maxv)); %maximum value
disp('MAX Purity overall ='); disp(max(OP)); %maximum purity-overall
max purity value
disp('Fitness ='); disp(f); %purity


toc
gscatter(x1,y1,gIdx);
```

KPSO:

```
n = 297;
bird_setp  = 100;
dim = 4;

c2 =1.49;
c1 = 1.49;
w =0.72;
fitness=0*ones(n,bird_setp);
xdat = load('kmean_heart_c4.dat');



R1 = rand(dim, n);
R2 = rand(dim, n);
current_fitness =0*ones(n,1);

current_position = xdat';
velocity = 0*ones(dim, n) ;
local_best_position  = current_position ;




for i = 1:n
    current_fitness(i) = Live_fn(current_position(:,i));
end


local_best_fitness  = current_fitness ;
[global_best_fitness,g] = min(local_best_fitness) ;

for i=1:n
    globl_best_position(:,i) = local_best_position(:,g) ;
end

+++++++++u
velocity = w *velocity + c1*(R1.*(local_best_position-
current_position)) + c2*(R2.*(globl_best_position-current_position));

current_position = current_position + velocity ;

tic

iter = 0 ;
while  ( iter < bird_setp )
iter = iter + 1;

for i = 1:n
current_fitness(i) = Live_fn(current_position(:,i)) ;
```

```
end


for i = 1 : n
        if current_fitness(i) < local_best_fitness(i)
           local_best_fitness(i)  = current_fitness(i);
           local_best_position(:,i) = current_position(:,i)   ;
        end
 end


 [current_global_best_fitness,g] = min(local_best_fitness);


if current_global_best_fitness < global_best_fitness
   global_best_fitness = current_global_best_fitness;

    for i=1:n
        globl_best_position(:,i) = local_best_position(:,g);
    end

end


 velocity = w *velocity + c1*(R1.*(local_best_position-
current_position)) + c2*(R2.*(globl_best_position-current_position));
 current_position = current_position + velocity;

 x=current_position(1,:);
 y=current_position(2,:);

 clf
   plot(x, y , 'h')
  axis([-9 9 -9 9]);

 pause(.1)


end

cp=current_position';

for j=1:n
     p=max(cp);
     P=p/cp(j);
end

for m=1:dim
     for j=1:n
         OP=cp(j)*P/dim;
     end
end
```

```
%fitness evaluation
for i=1:dim
  for j=1:n
     f1=xdat(i)-cp(j)/n;
  end
end

f=f1/dim;

[Jbest_min,I] = min(current_fitness);
[Jbest_max,I] = max(current_fitness);

disp(max(OP));
disp(f1);
disp(cp);

toc
```

KPSOK:

```
X=load('kpso_heart_c4.dat');
k=4;
k1=297;
X1=load('heart.dat');

rng(1);
[gIdx,c]=kmeans(X,k);

[n,m]=size(X);

if ~isscalar(k)
    c=k;
    k=size(c,1);
else
    c=X(ceil(rand(k,1)*n),:);
end

g0=ones(n,1);
gIdx=zeros(n,1);
D=zeros(n,k);
P=zeros(n,k);
OP=zeros(1,k);

tic


while any(g0~=gIdx)

    g0=gIdx;
    for t=1:k
        d=zeros(n,1);

        for s=1:m
            d=d+(X(:,s)-c(t,s)).^2;
        end
        D(:,t)=sqrt(d);
        x=D(:,1);
        y=D(:,2);
        x1=X1(:,4);
        y1=X1(:,5);

       clf

    end

    [z,gIdx]=min(D,[],2);

    for t=1:k
        c(t,:)=mean(X(gIdx==t,:));
    end
end

for j=1:n
     p=max(D);
     P=p/D(j);
end

for m=1:k
     for j=1:n
         OP=D(j)*P/k;
     end
end

%fitness evaluation
for i=1:k
  for j=1:n
     f1=X(i)-D(j)/n;
  end
end

f=f1/k;

maxv=max(D);
minv=min(D);

disp('CLUSTER center matrix='); disp(D); %cluster center matrix
disp('MIN value ='); disp(min(minv)); %minimum value
disp('MAX value ='); disp(max(maxv)); %maximum value
disp('MAX Purity overall ='); disp(max(OP)); %maximum purity-overall
max purity value
disp('Fitness ='); disp(f); %purity

toc
```

```
gscatter(x1,y1,gIdx);
```

LIVE FUNCTION:

```
function fposition=Live_fn(x)

    p=0;q=0;
    for k=1:5
    p=p+k*cos((k+1)*x(1)+k);
    q=q+k*cos((k+1)*x(2)+k);
    end

fposition=p*q+(x(1)+1.42513)^2+(x(2)+.80032)^2;
```

8. CONCLUSIONS AND FUTURE ENHANCEMENTS

Data clustering is the process of dividing data elements into classes or clusters so that items in the same class are as similar as possible, and items in different classes are as dissimilar as possible. Depending on the nature of the data and the purpose for which clustering is being used, different measures of similarity may be used to place items into classes, where the similarity measure controls how the clusters are formed.

The main objective of the project is to find the efficient optimization and similarities for Particle Swarm Optimization and K-Means. The performance was measured by the fitness function value and the CPU time. A general thumb rule is that a clustering with lower fitness function value, lower proximity error and lower CPU time is preferable. . The effectiveness of these algorithms was tested on 6 datasets, Heart Disease, Online News Popularity, Seeds, Prostate Cancer, Wholesale Customers Data, GPS Trajectories are taken.

Experimental results show that the proposed hybrid algorithm of K-Means with PSO i.e. KPSOK gives better results than K-Means, PSO and various combinations.

REFERENCES

1) C. Y. Chen and F. Ye, Particle swarm optimization and its application to clustering analysis, Proceedings of the Int. Conf. on Networking, Sensing and Control, Taipei: Taiwan, 2004, 789-794.
2) J. Kennedy and R. C. Eberhart, Particle swarm optimization, Proceedings of the IEEE International Joint Conference on Neural Networks, 4 (1995), 1942-1948.
3) L. Kaufman and P. Rousseeuw, Finding groups in Data: Introduction to cluster analysis, John Wily & Sons Inc., New York, 1990.
4) R. Eberhart and Y. Shi (1998)."A Modified Particle Swarm Optimizer", Proceedings of IEEE International Conference on Evolutionary Computation (ICEC'98), pp.69-73, Anchorage.
5) Hartigan, J. A. 1975. Clustering Algorithms. JohnWiley and Sons, Inc., New York, NY.
6) UCI Repository for Machine Learning Databases retrieved from the World Wide Web: http://www.ics.uci.edu/~mlearn/MLRepositor.y.htm.
7) www.matlabcentral.com
8) www.dynamic-optimization.org
9) www.google.com
10) www.slideshare.com
11) www.wikipedia.com

GLOSSARY

PSO	Particle Swarm Optimization
KPSO	K-Means Particle Swarm Optimization
KPSO-K	K-Means Particle Swarm Optimization-K-Means
GA	Genetic Algorithm
OFV	Object Function Value

Team members photo with guide :