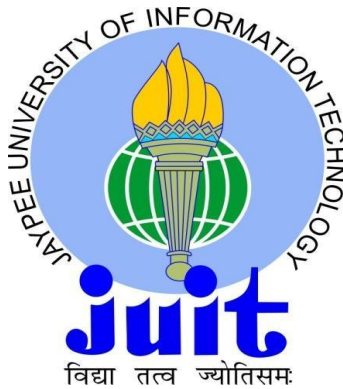


IMPLEMENTATION OF VARIOUS CLUSTERING TECHNIQUES

Enrollment No: 101308

Name: SaumyaMahajan

Supervisor Name:Dr. Pardeep Kumar



May 2014

Submitted in partial fulfillment of the Degree of
Bachelor of Technology

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING,
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,
WAKNAGHAT

TABLE OF CONTENTS

TOPIC	PAGE
Certificate	ii
Acknowledgement	iii
Abstract.....	iv
List of Figures.....	v
Chapter 1: Introduction	
1.1 What is Data Mining? Why is it important?	1
1.2 Problem Statement and Motivation.....	3
Chapter 2: Literature Survey	
2.1 What is Cluster Analysis?.....	4
2.2 Components of a Clustering Task.....	5
2.3 Requirements of Clustering.....	6
2.4 Types of Data in Cluster Analysis.....	8
2.5 A Categorization of Major Clustering Methods.....	11
2.6 Applications of Clustering.....	15
Chapter 3: Design and implementation	
3.1 DBSACN Algorithm.....	18
3.2 K-Means Algorithm.....	22
3.3 Code.....	25
Chapter 4: Result	
4.1 DBSCAN.....	36
4.2 K-Means.....	38
Chapter 5: Conclusion.....	41
References.....	42

CERTIFICATE

This is to certify that the work titled “**Implementation of Various Clustering Techniques**” submitted by “**Saumya Mahajan**” in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Wagnaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

.....

Dr. Pardeep Kumar

Assistant Professor (Senior Grade)

Date:

ACKNOWLEDGEMENT

On the very outset of this report, I would like to extend my sincere and heartfelt obligation towards all the personages who have helped me in this endeavor. Without their active guidance, help, cooperation & encouragement, I would not have made headway in the project.

I would like to extend my sincere gratitude and deep regards to our project mentor Dr. Pardeep Kumar, for his exemplary guidance, monitoring and constant encouragement throughout the course of this project.

Sincere thanks to Brig (Retd.) S.P.Ghrera, HOD, CSE & ICT Department, for being cooperative to the students of the department and providing relevant guidance in their endeavors.

I also extend our gratitude to Jaypee University of Information Technology for giving us this opportunity.

.....

Saumya Mahajan

Date:

ABSTRACT

Clustering is one of the important streams in data mining useful for discovering groups and identifying interesting distributions in the underlying data.

This project aims in analyzing and comparing the partitional and density based clustering algorithms namely K-Means and DBSCAN. The comparison is done based on the extent to which each of these algorithms identify the clusters and their pros and cons.

K-Means is a partitional clustering technique that helps to identify k clusters from a given set of n data points in d -dimensional space. It starts with k random centers and refines it at each step arriving to k clusters.

DBSCAN discovers clusters of arbitrary shape relying on a density based notion of clusters. Given ϵ as the input parameter, unlike k -means clustering, it tries to find out all possible clusters by classifying each point as core, border or noise. DBSCAN can be expensive as computation of nearest neighbors requires computing all pair wise proximities. Our implementation would provide a comparative study of K-Means against DBSCAN algorithm.

LIST OF FIGURES

Figure No.	Title	Page No.
1.1	Data Mining- searching for knowledge in your data	2
2.1	Stages in Clustering	5
2.2	Categorization of Clustering Algorithms	11
3.1	Clusters of Arbitrary Shape using DBSCAN	18
3.2	Border and Core Points	19
3.3	Point p directly density-reachable from point q	19
3.4	Point p density- reachable from point q	20
3.5	Density Connectivity	20
3.6	Clustering of a set of objects based on the k-means method	22
4.1	Selecting an algorithm	35
4.2	DBSCAN asking input parameters	36
4.3	DBSCAN showing clusters	37
4.4	K-Means asking input parameters	38
4.5	Entering Points	39
4.6	K-Means showing clusters	40

CHAPTER 1

INTRODUCTION

In recent years, the dramatic rise in the use of the web and the improvement in communications in general have transformed our society into one that strongly depends on information. The huge amount of data that is generated by this communication process contains important information that accumulates daily in databases and is not easy to extract. The field of **Data Mining** developed as a means of extracting information and knowledge from databases to discover patterns or concepts that are not evident.

As stated in [1], machine learning provides the technical basis of data mining by extracting information from the raw data in the databases. The process usually consists of the following: transforming the data to a suitable format, cleaning it, and inferring or making conclusions regarding the data.

Machine learning is divided into two primary sub-fields: supervised learning and unsupervised learning [2]. Within the category of unsupervised learning, one of the primary tools is clustering. This project report attempts to cover the main algorithms used for clustering, with brief and simple description of each.

1.1 What Is Data Mining? Why Is It Important?

Data mining has attracted a great deal of attention in the information industry and in society as a whole in recent years, due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information and knowledge. The information and knowledge gained can be used for applications ranging from market analysis, fraud detection, and customer retention, to production control and science exploration [3].

Data mining can be viewed as a result of the natural evolution of information technology. The database system industry has witnessed an evolutionary path in the development of the following functionalities: *data collection and database creation*, *data management* (including data storage and retrieval, and database transaction processing), and *advanced data analysis* (involving data warehousing and data mining). For instance, the early development of data collection and database creation mechanisms served as a prerequisite for later development of effective mechanisms for data storage and retrieval, and query and transaction processing. With numerous database systems offering query and transaction processing as common practice, advanced data analysis has naturally become the next target.

Many people treat data mining as a synonym for another popularly used term, ***Knowledge Discovery from Data***, or *KDD*. Alternatively, others view data mining as simply an essential step in the process of knowledge discovery.



Figure 1.1Data mining—searching for knowledge (interesting patterns) in your data.[3]

1.2 Problem Statement and Motivation

The abundance of data, coupled with the need for powerful data analysis tools, has been described as a *data rich but information poor situation*[4]. The fast-growing, tremendous amount of data, collected and stored in large and numerous data repositories, has far exceeded our human ability for comprehension without powerful tools. As a result, data collected in large data repositories become “data tombs”—data archives that are seldom visited. Consequently, important decisions are often made based not on the information-rich data stored in data repositories, but rather on a decision maker’s intuition, simply because the decision maker does not have the tools to extract the valuable knowledge embedded in the vast amounts of data. In addition, consider expert system technologies, which typically rely on users or domain experts to manually input knowledge into knowledge bases. Unfortunately, this procedure is prone to biases and errors, and is extremely time-consuming and costly. Data mining tools perform data analysis and may uncover important data patterns, contributing greatly to business strategies, knowledge bases, and scientific and medical research. The widening gap between data and information calls for a systematic development of *data mining tools* that will turn data tombs into “golden nuggets” of knowledge. Thus data analysis tools are required for in-depth analysis, such as data classification, clustering, and the characterization of data.

CHAPTER 2

LITERATURE SURVEY

2.1 What is Cluster Analysis?

The process of grouping a set of physical or abstract objects into classes of similar objects is called *clustering*. A *cluster* is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters [5]. A cluster of data objects can be treated collectively as one group and so may be considered as a form of data compression. Although classification is an effective means for distinguishing groups or classes of objects, it requires the often costly collection and labeling of a large set of training tuples or patterns, which the classifier used to model each group. It is often more desirable to proceed in the reverse direction: First partition the set of data into groups based on data similarity and then assign labels to the relatively small number of groups. Additional advantages of such a clustering – based process are that it is adaptable to changes and helps single out useful features that distinguish different groups.

Clustering is also called **data segmentation** in some applications because clustering partitions large data sets into groups according to their *similarity*. Clustering can also be used for **outlier detection**, where outliers (values that are “far away” from any cluster) may be more interesting than common cases.

In machine learning, clustering is an example of **unsupervised learning**. Unlike classification, clustering and unsupervised learning do not rely on predefined classes and class-labeled training examples. For this reason, clustering is a form of learning by observation, rather than learning by examples. In data mining, efforts have focused on finding methods for efficient and effective cluster analysis in large databases.

A good clustering method will produce high quality clusters with

- High intra-class similarity
- Low inter-class similarity

The quality of a clustering result depends on both the similarity measure used by the method and its implementation. The quality of a clustering method is also measured by its ability to discover some or all of the hidden patterns.

2.2 Components of a Clustering Task[6]

Typical pattern clustering activity involves the following steps:

1. Pattern representation
2. Definition of a pattern
3. Clustering or grouping
4. Data abstraction (if needed)
5. Assessment of output(if needed)

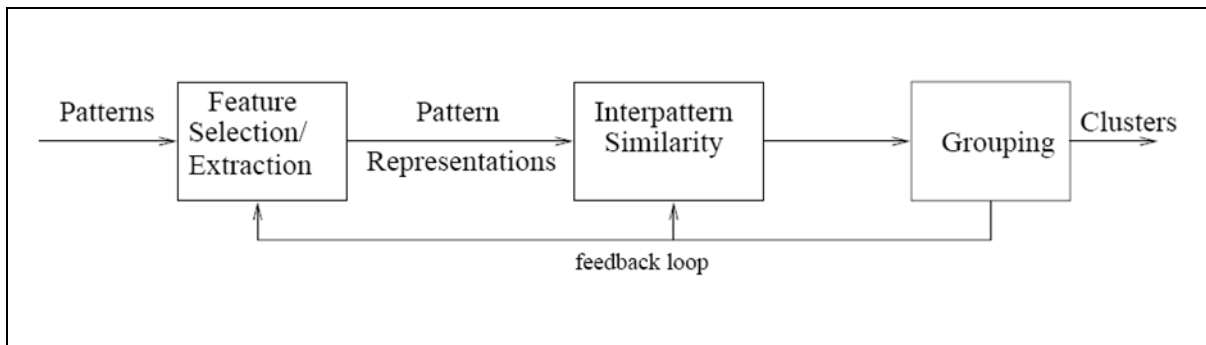


Figure 2.1 Stages in Clustering [6]

Figure 2.1 depicts a typical sequencing of the first three of these steps, including a feedback path where the grouping process output could affect subsequent feature extraction and similarity computations.

Pattern representation refers to the number of classes, the number of available patterns, and the number, type, and scale of the features available to the clustering algorithm. Some of this information may not be controllable by the practitioner.

Feature selection is the process of identifying the most effective subset of the original features to use in clustering.

Feature extraction is the use of one or more transformations of the input features to produce new salient features. Either or both of these techniques can be used to obtain an appropriate set of features to use in clustering.

Pattern proximity is usually measured by a distance function defined on pairs of patterns. A variety of distance measures are in use in the various communities. A simple distance measure like Euclidean distance can often be used to reflect dissimilarity between two patterns, whereas other similarity measures can be used to characterize the conceptual similarity between patterns.

The **grouping** step can be performed in a number of ways. The output clustering can be hard (a partition of the data into groups) or fuzzy (where each pattern has a variable degree of membership in each of the output clusters).

Data abstraction is the process of extracting a simple and compact representation of a data set. Here, simplicity is either from the perspective of automatic analysis or it is human-oriented. In the clustering context, a typical data abstraction is a compact description of each cluster, usually in terms of cluster prototypes or representative patterns such as the centroid.

2.3 Requirements of Clustering

Clustering is a challenging field of research in which its potential applications pose their own special requirements. The following are typical requirements of clustering in data mining [3]:

Scalability: Many clustering algorithms work well on small data sets containing fewer than several hundred data objects; however, a large database may contain millions of objects. Clustering on a *sample* of a given large data set may lead to biased results. Highly scalable clustering algorithms are needed.

Ability to deal with different types of attributes: Many algorithms are designed to cluster interval-based (numerical) data. However, applications may require clustering other types of data, such as binary, categorical (nominal), and ordinal data, or mixtures of these data types.

Discovery of clusters with arbitrary shape: Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures. Algorithms based on such distance measures tend to find spherical clusters with similar size and density. However, a cluster could be of any shape. It is important to develop algorithms that can detect clusters of arbitrary shape.

Minimal requirements for domain knowledge to determine input parameters: Many clustering algorithms require users to input certain parameters in cluster analysis (such as the number of desired clusters). The clustering results can be quite sensitive to input parameters. Parameters are often difficult to determine, especially for data sets containing high-dimensional objects. This not only burdens users, but it also makes the quality of clustering difficult to control.

Ability to deal with noisy data: Most real-world databases contain outliers or missing, unknown, or erroneous data. Some clustering algorithms are sensitive to such data and may lead to clusters of poor quality.

Incremental clustering and insensitivity to the order of input records: Some clustering algorithms cannot incorporate newly inserted data (i.e., database updates) into existing clustering structures and, instead, must determine a new clustering from scratch. Some clustering algorithms are sensitive to the order of input data. That is, given a set of data objects, such an algorithm may return dramatically different clusterings depending on the order of presentation of the input objects. It is important to develop incremental clustering algorithms and algorithms that are insensitive to the order of input.

High dimensionality: A database or a data warehouse can contain several dimensions or attributes. Many clustering algorithms are good at handling low-dimensional data, involving only two to three dimensions. Human eyes are good at judging the quality of clustering for up to three dimensions. Finding clusters of data objects in high dimensional space is challenging, especially considering that such data can be sparse and highly skewed.

Constraint-based clustering: Real-world applications may need to perform clustering under various kinds of constraints. Suppose that your job is to choose the locations for a given number of new automatic banking machines (ATMs) in a city. To decide upon this, you may cluster households while considering constraints such as the city’s rivers and highway networks, and the type and number of customers per cluster. A challenging task is to find groups of data with good clustering behavior that satisfy specified constraints.

Interpretability and usability: Users expect clustering results to be interpretable, comprehensible, and usable. That is, clustering may need to be tied to specific semantic interpretations and applications. It is important to study how an application goal may influence the selection of clustering features and methods.

2.4 Types of Data in Cluster Analysis

In this section, we study the types of data that often occur in cluster analysis and how to preprocess them for such an analysis. The various data objects can be described by *interval-scaled* variables; by *binary* variables; by *categorical*, and *ordinal* variables; or combinations of these variable types. Then we compute objects similarity based on which we compute clusters of objects.

2.4.1 Interval-Scaled Variables

Interval-scaled variables are continuous measurements of a roughly linear scale. Typical examples include weight and height, latitude and longitude coordinates and weather temperature. The distance measures that are commonly used for computing the similarity of objects described by such variables include the *Euclidean*, *Manhattan*, and *Minkowski distances* [7].

The most popular distance measure is ***Euclidean distance***, which is defined as

$$d(i, j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{in} - x_{jn})^2}, \quad (2.1)$$

Another well-known metric is ***Manhattan distance***, defined as

$$d(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{in} - x_{jn}|. \quad (2.2)$$

Minkowski distance is a generalization of both Euclidean distance and Manhattan distance. It is defined as

$$d(i, j) = (|x_{i1} - x_{j1}|^p + |x_{i2} - x_{j2}|^p + \dots + |x_{in} - x_{jn}|^p)^{1/p}, \quad (2.3)$$

where, $i = (x_{i1}, x_{i2}, \dots, x_{in})$ and $j = (x_{j1}, x_{j2}, \dots, x_{jn})$ are two n dimensional data objects.

2.4.2 Binary Variables

A binary variable has only two states: 0 or 1, where 0 means that the variable is absent, and 1 means that it is present. One such example could be the attribute gender having the states male and female.

A binary variable is *symmetric* if both of its states are equally valuable and carry the same weight; that is, there is no preference on which outcome should be coded as 0 or 1. Its distance measure, defined in Equation (2.4) can be used to assess the dissimilarity between objects i and j .

$$d(i, j) = \frac{r + s}{q + r + s + t} \quad (2.4)$$

A binary variable is *asymmetric* if the outcomes of the states are not equally important. By convention, we shall code the most important outcome, which is usually the rarest one, by 1 and the other by 0.

$$d(i, j) = \frac{r + s}{q + r + s} \quad (2.5)$$

Table 2.1 A contingency table for binary variable

		object j		
		1	0	sum
object i	1	Q	r	q + r
	0	S	t	s + t
	Sum	q + s	r + t	p

2.4.3 Categorical Variables

A categorical variable is a generalization of the binary variable in that it can take on more than two states. For example, map color is a categorical variable that may have, say, five states: red, yellow, green, pink, and blue. The dissimilarity between two objects i and j can be computed based on the ratio of mismatches:

$$d(i, j) = \frac{p - m}{p} \quad (2.6)$$

where m is the number of matches and p is the total number of variables.

2.4.4 Ordinal variables

An *ordinal variable* resembles a categorical variable, except that the M states of the ordinal value are ordered in a meaningful sequence. Ordinal variables are very useful for registering subjective assessments of qualities that cannot be measured objectively. For example, the relative ranking in a particular sport (e.g., gold, silver, bronze) is often more essential than the actual values of a particular measure. The treatment of ordinal variables is quite similar to that of interval-scaled variables when computing the dissimilarity between objects. Suppose that f is a variable from a set of ordinal variables describing objects. The dissimilarity computation with respect to f involves the following steps:

1. The value of f for the i th object is x_{if} , and f has M_f ordered states, representing the ranking $1, \dots, M_f$. Replace each x_{if} by its corresponding rank, $r_{if} \in \{1, \dots, M_f\}$.
2. Since each ordinal variable can have a different number of states, it is often necessary to map the range of each variable onto $[0.0, 1.0]$ so that each variable has equal weight. This can be achieved by replacing the rank r_{if} of the i th object in the f th variable by

$$z_{if} = \frac{r_{if} - 1}{M_f - 1} \quad (2.7)$$

3. Dissimilarity can then be computed using any of the distance measures described in Section 2.4.1 for interval-scaled variables, using z_{if} to represent the f value for the i th object.

2.4.5 Vector Objects

In some applications, such as information retrieval, text document clustering, and biological taxonomy, we need to compare and cluster complex objects (such as documents) containing a large number of symbolic entities (such as keywords and phrases). To measure the distance between complex objects, it is often desirable to abandon traditional metric distance computation and introduce a nonmetric similarity function.

2.5 A Categorization of Major clustering Methods

Many clustering algorithms exist in the literature. It is difficult to provide a crisp categorization of clustering methods because these categories may overlap, so that a method may have features from several categories. Nevertheless, it is useful to present a relatively organized picture of the different clustering methods. In general, the major clustering methods can be classified into the following categories.

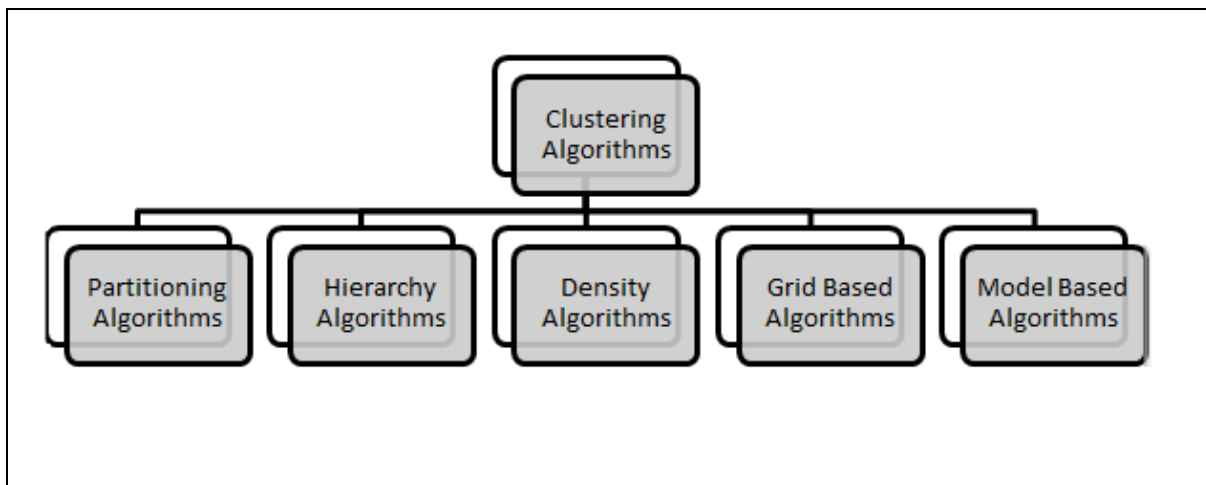


Figure 2.2 Categorization of Clustering Algorithms

1. Partitioning Methods

Given a database of n objects or data tuples, a partitioning method constructs k partitions of the data, where each partition represents a cluster and $k < n$. That is, it classifies the data into k groups, which together satisfy the following requirements:

- (1) each group must contain at least one object, and

(2) each object must belong to exactly one group.

Given k , the number of partitions to construct, a partitioning method creates an initial partitioning. It then uses an iterative relocation technique that attempts to improve the partitioning by moving objects from one group to another [8]. The general criterion of a good partitioning is that objects in the same cluster are “close” or related to each other, whereas objects of different clusters are “far apart” or very different. There are various kinds of other criteria for judging the quality of partitions.

To achieve global optimality in partitioning-based clustering would require the exhaustive enumeration of all of the possible partitions. Instead, most applications adopt one of a few popular heuristic methods, such as (1) the *k-means* algorithm, where each cluster is represented by the mean value of the objects in the cluster, and (2) the *k-medoids* algorithm, where each cluster is represented by one of the objects located near the center of the cluster. These heuristic clustering methods work well for finding spherical-shaped clusters in small to medium-sized databases. To find clusters with complex shapes and for clustering very large data sets, partitioning-based methods need to be extended.

2. Hierarchical Methods

A hierarchical method creates a hierarchical decomposition of the given set of data objects. A hierarchical method can be classified as being either *agglomerative* or *divisive*, based on how the hierarchical decomposition is formed. The *agglomerative approach*, also called the *bottom-up* approach, starts with each object forming a separate group. It successively merges the objects or groups that are close to one another, until all of the groups are merged into one (the topmost level of the hierarchy), or until a termination condition holds. The *divisive approach*, also called the *top-down* approach, starts with all of the objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until eventually each object is in one cluster, or until a termination condition holds.

Hierarchical methods suffer from the fact that once a step (merge or split) is done, it can never be undone. This rigidity is useful in that it leads to smaller computation costs by not having to worry about a combinatorial number of different choices. However, such techniques cannot correct erroneous decisions. There are two approaches to improving the

quality of hierarchical clustering: (1) perform careful analysis of object “linkages” at each hierarchical partitioning, or (2) integrate hierarchical agglomeration and other approaches by first using a hierarchical agglomerative algorithm to group objects into *micro clusters*, and then performing *macro clustering* on the micro clusters using another clustering method such as iterative relocation.

3. Density-based Methods

Most partitioning methods cluster objects based on the distance between objects. Such methods can find only spherical-shaped clusters and encounter difficulty at discovering clusters of arbitrary shapes. Other clustering methods have been developed based on the notion of *density*. Their general idea is to continue growing the given cluster as long as the density (number of objects or data points) in the “neighborhood” exceeds some threshold; that is, for each data point within a given cluster, the neighborhood of a given radius has to contain at least a minimum number of points. Such a method can be used to filter out noise (outliers) and discover clusters of arbitrary shape.

DBSCAN and its extension, OPTICS, are typical density-based methods that grow clusters according to a density-based connectivity analysis. DENCLUE is a method that clusters objects based on the analysis of the value distributions of density functions.

4. Grid-based Methods

Grid-based methods quantize the object space into a finite number of cells that form a grid structure. All of the clustering operations are performed on the grid structure (i.e., on the quantized space). The main advantage of this approach is its fast processing time, which is typically independent of the number of data objects and dependent only on the number of cells in each dimension in the quantized space.

STING is a typical example of a grid-based method. WaveCluster applies wavelet transformation for clustering analysis and is both grid-based and density-based.

5. Model-based Methods

Model-based methods hypothesize a model for each of the clusters and find the best fit of the data to the given model. A model-based algorithm may locate clusters by constructing a density function that reflects the spatial distribution of the data points. It also leads to a way of automatically determining the number of clusters based on standard statistics, taking “noise” or outliers into account and thus yielding robust clustering methods.

EM is an algorithm that performs expectation-maximization analysis based on statistical modeling. COBWEB is a conceptual learning algorithm that performs probability analysis and takes *concepts* as a model for clusters. SOM (or self-organizing feature map) is a neural network-based algorithm that clusters by mapping high dimensional data into a 2-D or 3-D feature map, which is also useful for data visualization.

The choice of clustering algorithm depends both on the type of data available and on the particular purpose of the application. If cluster analysis is used as a descriptive or exploratory tool, it is possible to try several algorithms on the same data to see what the data may disclose.

Some clustering algorithms integrate the ideas of several clustering methods, so that it is sometimes difficult to classify a given algorithm as uniquely belonging to only one clustering method category. Furthermore, some applications may have clustering criteria that require the integration of several clustering techniques. Aside from the above categories of clustering methods, there are two classes of clustering tasks that require special attention. One is *clustering high-dimensional data*, and the other is *constraint-based clustering*.

Clustering high-dimensional data is a particularly important task in cluster analysis because many applications require the analysis of objects containing a large number of features or dimensions. For example, text documents may contain thousands of terms or keywords as features, and DNA microarray data may provide information on the expression levels of thousands of genes under hundreds of conditions. Clustering high-dimensional data is challenging due to the curse of dimensionality. Many dimensions may

not be relevant. As the number of dimensions increases, the data become increasingly sparse so that the distance measurement between pairs of points become meaningless and the average density of points anywhere in the data is likely to be low. Therefore, a different clustering methodology needs to be developed for high-dimensional data. CLIQUE and PROCLUS are two influential *subspace clustering methods*, which search for clusters in subspaces (or subsets of dimensions) of the data, rather than over the entire data space. Frequent pattern-based clustering, another clustering methodology, extracts *distinct frequent patterns* among subsets of dimensions that occur frequently. It uses such patterns to group objects and generate meaningful clusters. pCluster is an example of frequent pattern-based clustering that groups objects based on their pattern similarity.

Constraint-based clustering is a clustering approach that performs clustering by incorporation of user-specified or application-oriented constraints. A constraint expresses a user's expectation or describes "properties" of the desired clustering results, and provides an effective means for communicating with the clustering process. Various kinds of constraints can be specified, either by a user or as per application requirements. Focus of discussion is generally on *spatial clustering* with the existence of obstacles and clustering under user-specified constraints. In addition, *semi-supervised clustering* is described, which employs, for example, pairwise constraints (such as pairs of instances labeled as belonging to the same or different clusters) in order to improve the quality of the resulting clustering.

2.6 Applications of Clustering

1. Business and Marketing

- *Market research*: Cluster analysis is widely used in market research when working with multivariate data from surveys and test panels. Market researchers use cluster analysis to partition the general population of consumers into market segments and to better understand the relationships between different groups of consumers/potential customers, and for use in market segmentation, Product positioning, New product development and Selecting test markets.

- *Grouping of shopping items:* Clustering can be used to group all the shopping items available on the web into a set of unique products. For example, all the items on eBay can be grouped into unique products.

2. World Wide Web[11]

- *Social network analysis:* In the study of social networks, clustering may be used to recognize communities within large groups of people.
- *Search result grouping:* In the process of intelligent grouping of the files and websites, clustering may be used to create a more relevant set of search results compared to normal search engines like Google. There are currently a number of web based clustering tools such as Clusty.

3. Computer Science

- *Software evolution:* Clustering is useful in software evolution as it helps to reduce legacy properties in code by reforming functionality that has become dispersed. It is a form of restructuring and hence is a way of directly preventative maintenance.
- *Image segmentation:* Clustering can be used to divide a digital image into distinct regions for border detection or object recognition.
- *Recommender systems:* Recommender systems are designed to recommend new items based on a user's tastes. They sometimes use clustering algorithms to predict a user's preferences based on the preferences of other users in the user's cluster.

4. Biology

- *Plant and animal ecology:* cluster analysis is used to describe and to make spatial and temporal comparisons of communities (assemblages) of organisms in heterogeneous environments; it is also used in plant systematics to generate artificial phylogenies or clusters of organisms (individuals) at the species, genus or higher level that share a number of attributes.
- *Human genetic clustering:* The similarity of genetic data is used in clustering to infer population structures.

5. Data Mining

- As a Stand- alone tool to get insight into data distribution
- As a preprocessing step for other algorithms

6. Clustering Algorithm in Academics[10]

The ability to monitor the progress of students' academic performance has been the critical issue for the academic community of higher learning. Clustering algorithm can be used to monitor the students' academic performance. Based on the students' score they are grouped into different-different clusters (using k-means), where each clusters denoting the different level of performance. By knowing the number of students' in each cluster we can know the average performance of a class as a whole.

7. Clustering Algorithm in Wireless Sensor Network's based Application

Clustering Algorithm can be used effectively in Wireless Sensor Network's based application. One application where it can be used is in Landmine detection.

8. Outlier Detection

Applications of outlier detection include the detection of credit card fraud and the monitoring of criminal activities in electronic commerce. Exceptional cases in credit card transactions, such as very expensive and frequent purchases, may be of interest as possible fraudulent activity.

CHAPTER 3

DESIGN AND IMPLEMENTATION

3.1 DBSCAN Clustering Algorithm

The DBSCAN algorithm can identify clusters in large spatial data sets by looking at the [9] local density of database elements, using only one input parameter. Furthermore, the user gets a suggestion on which parameter value that would be suitable. Therefore, minimal knowledge of the domain is required. The DBSCAN can also determine what information should be classified as noise or outliers. In spite of this, its working process is quick and scales very well with the size of the database – almost linearly.

By using the density distribution of nodes in the database, DBSCAN can categorize these nodes into separate clusters that define the different classes. DBSCAN can find clusters of arbitrary shape, as can be seen in figure 3.1. However, clusters that lie close to each other tend to belong to the same class.

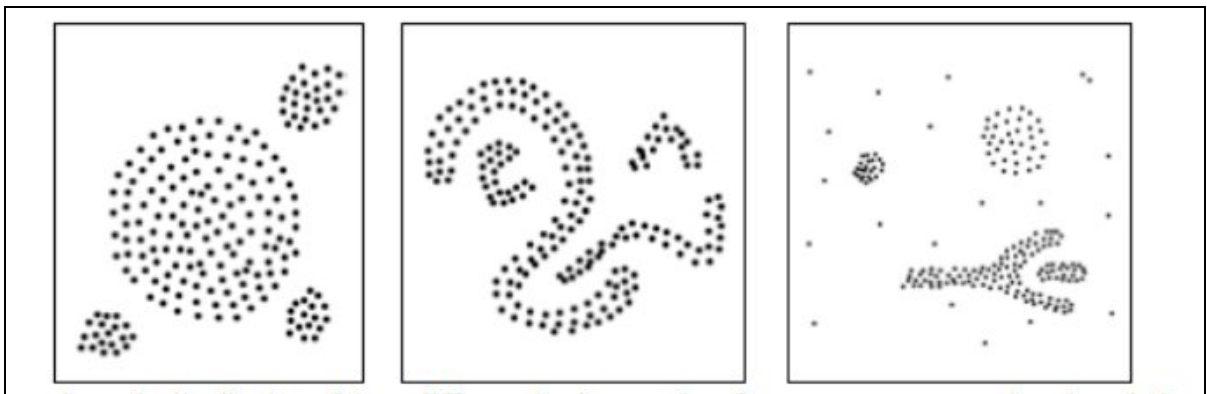


Figure 3.1 Clusters of arbitrary shapes using DBSCAN [8]

The following section will describe further how the DBSCAN algorithm works. Its computing process is based on six rules or definitions, creating two lemmas.

Definition 1: (*The Eps-neighborhood of a point*)

$$N_{Eps}(p) = \{q \in D \mid \text{dist}(p,q) < Eps\} \quad (3.1)$$

For a point to belong to a cluster it needs to have at least one other point that lies closer to it than the distance Eps .

Definition 2: (*Directly density-reachable*)

There are two kinds of points belonging to a cluster; there are border points and core points, as can be seen in figure 3.2.

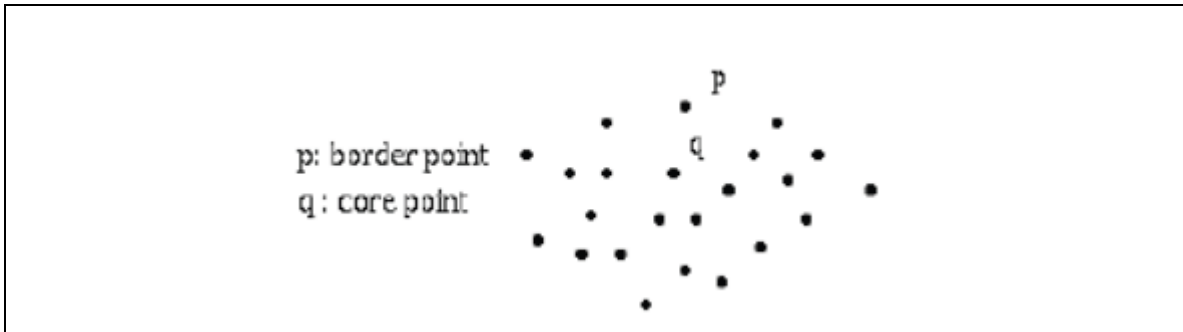


Figure 3.2 Border and Core Points

“The Eps -neighborhood of a border point tends to have significantly less points than the Eps neighborhood of a core point”. The border points will still be a part of the cluster and in order to include these points, they must belong to the Eps -neighborhood of a core point q as seen in figure 3.3

$$1) p \in N_{Eps}(q)$$

In order for point q to be a core point it needs to have a minimum number of points within its Eps -neighborhood.

$$2) |N_{Eps}(q)| \geq \text{MinPts (core point condition)} \quad (3.2)$$

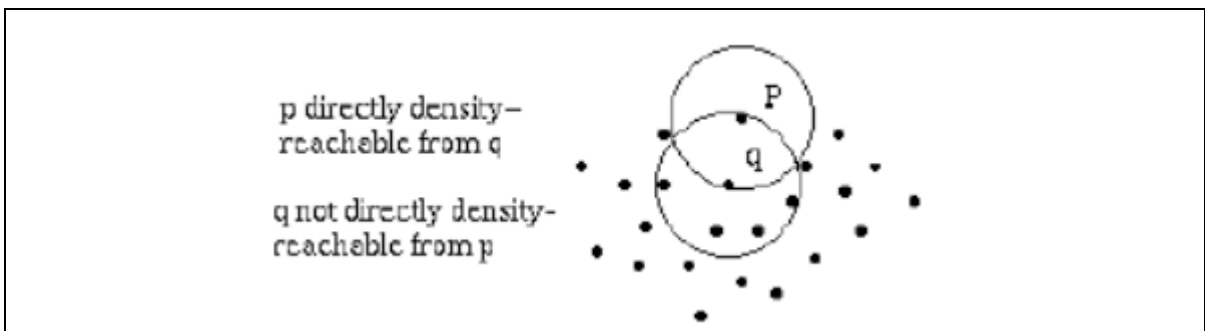


Figure 3.3 Point p is directly density-reachable from point q but not vice versa

Definition 3: (*Density-reachable*)

“A point p is *density-reachable* from a point q with respect to Eps and $MinPts$ if there is a chain of points p_1, \dots, p_n , $p_1 = q$, $p_n = p$ such that p_{i+1} is *directly density-reachable* from p_i .” Figure 3.4 shows an illustration of a density-reachable point.

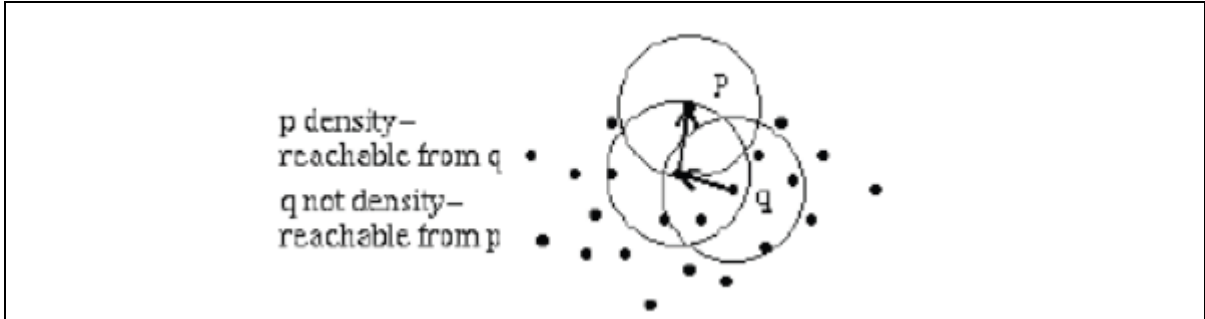


Figure 3.4 Point p is density reachable from point q and not vice versa

Definition 4: (*Density-connected*)

There are cases when two border points will belong to the same cluster but where the two border points don't share a specific core point. In these situations the points will not be *density-reachable* from each other. There must however be a core point o from which they are both *density-reachable*. Figure 3.5 shows how density connectivity works.

“A point p is *density-connected* to a point q with respect to Eps and $MinPts$ if there is a point o such that both, p and q are density-reachable from o with respect to Eps and $MinPts$.”

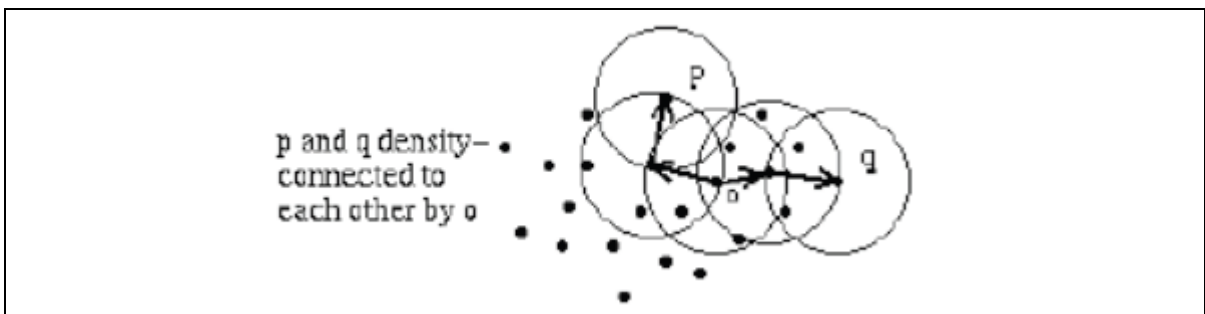


Figure 3.5 Density connectivity

Definition 5: (*cluster*)

If point p is a part of a cluster C and point q is *density-reachable* from point p with respect to a given distance and a minimum number of points within that distance, then q is also a part of cluster C .

1) " $\forall p, q$: if $p \in C$ and q is *density-reachable* from p with respect to Eps and $MinPts$, then $q \in C$.

Two points belongs to the same cluster C , is the same as saying that p is density-connected to q with respect to the given distance and the number of points within that given distance.

2) " $\forall p, q \in C$: p is *density-connected* to q with respect to Eps and $MinPts$.

Definition 6: (*noise*)

Noise is the set of points, in the database, that don't belong to any of the clusters.

Lemma 1:

A cluster can be formed from any of its core points and will always have the same shape.

Lemma 2:

Let p be a core point in cluster C with a given minimum distance (Eps) and a minimum number of points within that distance ($MinPts$). If the set O is *density-reachable* from p with respect to the same Eps and $MinPts$, then C is equal to the set O .

To find a cluster, DBSCAN starts with an arbitrary point p and retrieves all points density-reachable from p with respect to Eps and $MinPts$.

If p is a core point, this procedure yields a cluster with respect to Eps and $MinPts$ (see Lemma 2).

If p is a border point then no points are density-reachable from p and DBSCAN visits the next point of the database."

3.2 K-Means Clustering Algorithm

One of the most popular heuristics for solving the k-means problem is based on a simple iterative scheme for finding a locally optimal solution. This algorithm is often called the k-means algorithm. This algorithm is also referred to as the Lloyd's algorithm [1].

The k-means algorithm partitions the dataset into 'k' subsets such that all records, from now on referred to as points, in a given subset belong to the same center. Also the points in a given subset are closer to that center than to any other center.

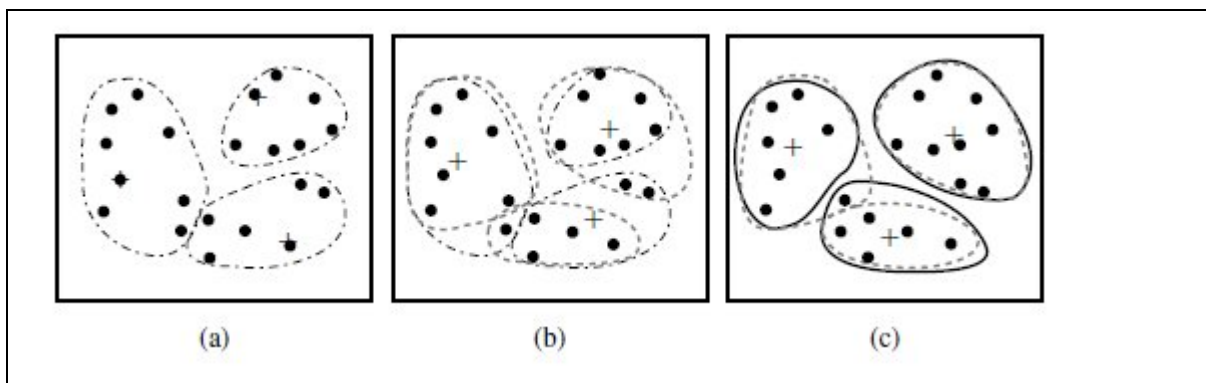


Figure 3.6 Clustering of a set of objects based on the k-means method (The mean of each cluster is marked by a “+”) [3]

The algorithm keeps track of the centroids of the subsets, and proceeds in simple iterations. The initial partitioning is randomly generated, that is, we randomly initialize the centroids to some points in the region of the space. In each iteration step, a new set of centroids is generated using the existing set of centroids following two very simple steps. Let us denote the set of centroids after the i^{th} iteration by $C^{(i)}$. The following operations are performed in the steps:

1. Partition the points based on the centroids $C^{(i)}$, that is, find the centroids to which each of the points in the dataset belongs. The points are partitioned based on the Euclidean distance from the centroids.

2. Set a new centroid $c^{(i+1)} \in C^{(i+1)}$ to be the mean of all the points that are closest to $c^{(i)} \in C^{(i)}$. The new location of the centroid in a particular partition is referred to as the new location of the old centroid.

The algorithm is said to have converged when recomputing the partitions does not result in a change in the partitioning. In the terminology that we are using, the algorithm has converged completely when $C^{(i)}$ and $C^{(i-1)}$ are identical. For configurations where no point is equidistant to more than one center, the above convergence condition can always be reached. This convergence property along with its simplicity adds to the attractiveness of the k-means algorithm.

The k-means needs to perform a large number of "nearest-neighbor" queries for the points in the dataset. If the data is 'd' dimensional and there are 'N' points in the dataset, the cost of a single iteration is $O(kdN)$. As one would have to run several iterations, it is generally not feasible to run the naïve k-means algorithm for large number of points.

Sometimes the convergence of the centroids (i.e. $C^{(i)}$ and $C^{(i+1)}$ being identical) takes several iterations. Also in the last several iterations, the centroids move very little. As running the expensive iterations so many more times might not be efficient, we need a measure of convergence of the centroids so that we stop the iterations when the convergence criteria is met. Distortion is the most widely accepted measure. Clustering error measures the same criterion and is sometimes used instead of distortion. In fact k-means algorithm is designed to optimize distortion. Placing the cluster center at the mean of all the points minimizes the distortion for the points in the cluster. Also when another cluster center is closer to a point than its current cluster center, moving the cluster from its current cluster to the other can reduce the distortion further. The above two steps are precisely the steps done by the k-means cluster. Thus k-means reduces distortion in every step locally. The k-Means algorithm terminates at a solution that is locally optimal for the distortion function. Hence, a natural choice as a convergence criterion is distortion. Among other measures of convergence used by other researchers, we can measure the sum of Euclidean distance of the new centroids from the old centroids as in [9]. In this thesis we always use clustering error/distortion as the convergence criterion for all variants of k-means algorithm.

Definition 1: *Clustering error* is the sum of the squared Euclidean distances from points to the centers of the partitions to which they belong.

Mathematically, given a clustering ϕ , we denote by $\phi(x)$ the centroid this clustering associates with an arbitrary point x (so for k-means, $\phi(x)$ is simply the center closest to x). We then define a measure of quality for ϕ :

$$distortion_{\phi} = \frac{1}{N} \sum_x |x - \phi(x)|^2 \quad (3.3)$$

Where $|a|$ is used to denote the norm of a vector ‘a’. The lesser the difference in distortion over successive iterations, the more the centroids have converged. Distortion is therefore used as a measure of goodness of the partitioning.

In spite of its simplicity, k-means often converges to local optima. The quality of the solution obtained depends heavily on the initial set of centroids, which is the only non-deterministic step in the algorithm. Note that although the starting centers can be selected arbitrarily, k-means is fully deterministic, given the starting centers. A bad choice of initial centers can have a great impact on both performance and distortion. Also a good choice of initial centroids would reduce the number of iterations that are required for the solution to converge. Many algorithms have tried to improve the quality of the k-means solution by suggesting different ways of sampling the initial centers, but none has been able to avoid the problem of the solution converging to a local optimum.

3.3 Code

3.3.1 Code for DBSCAN

dbscan.java

```
import java.util.*;
public class dbscan
{
    public static int e=GUI.tdistance;
    public static int minpt =GUI.minpoints;
    public static Vector<List> resultList = new Vector<List>();
    public static Vector<Point> pointList = Utility.getList();
    public static Vector<Point> Neighbours ;
    public static Vector<List> applyDbscan()
    {
        resultList.clear();
        pointList.clear();
        Utility.VisitList.clear();
        pointList=Utility.getList();
        int index2 =0;
        while (pointList.size()>index2){
            Point p =pointList.get(index2);
            if(!Utility.isVisited(p)){
                Utility.Visited(p);
                Neighbours =Utility.getNeighbours(p);
                if (Neighbours.size()>=minpt)
                {
                    int ind=0;
                    while(Neighbours.size()>ind)
                    {
                        Point r = Neighbours.get(ind);
                        if(!Utility.isVisited(r)){
                            Utility.Visited(r);
                            Vector<Point> Neighbours2 = Utility.getNeighbours(r);
                            if (Neighbours2.size() >= minpt){
                                Neighbours=Utility.Merge(Neighbours, Neighbours2);
                            }
                        } ind++;
                    }
                }
                System.out.println("N"+Neighbours.size());
                resultList.add(Neighbours);}
            }index2++;
        }return resultList;
    }
}
```

Point.java

```
public class Point
{
private double x;
private double y;
Point(double a, double b)
{
x=a;
y=b;
}

public double getX ()
{
return x;
}

public double getY ()
{
return y;
}
}
```

Utility.java

```
import java.util.Iterator;
import java.util.Vector;
public class Utility{
public static Vector<Point> VisitList = new Vector<Point>();
public static double getDistance (Point p, Point q)
{
double dx = p.getX()-q.getX();
double dy = p.getY()-q.getY();
double distance = Math.sqrt (dx * dx + dy * dy);
return distance;
}

/**
neighbourhood points of any point p
**/

public static Vector<Point> getNeighbours(Point p)
```



```

{
Vector<Point> neigh =new Vector<Point>();
Iterator<Point> points = dbscan.pointList.iterator();
while(points.hasNext()){
Point q = points.next();
if(getDistance(p,q)<= dbscan.e){
neigh.add(q);
}
}
return neigh;
}

public static void Visited(Point d){
VisitList.add(d);
}

public static boolean isVisited(Point c)
{
if (VisitList.contains(c))
{
return true;
}
else
{
return false;
}
}

public static Vector<Point> Merge(Vector<Point> a,Vector<Point> b)
{

Iterator<Point> it5 = b.iterator();
while(it5.hasNext()){
Point t = it5.next();
if (!a.contains(t) ){
a.add(t);
}
}
return a;
}

// Returns PointsList to DBscan.java

public static Vector<Point> getList() {

Vector<Point> newList =new Vector<Point>();

```

```

newList.clear();
newList.addAll(GUI.hset);
return newList;
}

public static Boolean equalPoints(Point m , Point n) {
if((m.getX()==n.getX())&&(m.getY()==n.getY()))
return true;
else
return false;
}
}
}

```

3.3.2 Code for K-Means

KMeans.java

```

import java.util.ArrayList;
import java.util.List;
import java.util.Vector;

public class KMeans
{ public static Vector<Point> pointList;
public static Vector<List> resultList = new Vector<List>();
  private static final int NUM_CLUSTERS = 2; // Total clusters.
  private static int TOTAL_DATA ; // Total data points.

  private static double SAMPLES[][] = new double[100][100] ;

  private static ArrayList<Data> dataSet = new ArrayList<Data>();
  private static ArrayList<Centroid> centroids = new ArrayList<Centroid>();

  private static void initialize()
  {
    System.out.println("Centroids initialized at:");
    centroids.add(new Centroid(SAMPLES[0][0], SAMPLES[0][1])); // lowest set.
    centroids.add(new Centroid(SAMPLES[TOTAL_DATA-1][0],
SAMPLES[TOTAL_DATA-1][1])); // highest set.
    System.out.println(" (" + centroids.get(0).X() + ", " + centroids.get(0).Y() + ")");
    System.out.println(" (" + centroids.get(1).X() + ", " + centroids.get(1).Y() + ")");
  }
}

```

```

    System.out.print("\n");
    return;
}

private static void kMeanCluster()
{
    final double bigNumber = Math.pow(10, 10); // some big number that's sure to be
larger than our data range.
    double minimum = bigNumber; // The minimum value to beat.
    double distance = 0.0; // The current minimum value.
    int sampleNumber = 0;
    int cluster = 0;
    boolean isStillMoving = true;
    Data newData = null;

    // Add in new data, one at a time, recalculating centroids with each new one.
    while(dataSet.size() < TOTAL_DATA)
    {
        newData = new Data(SAMPLES[sampleNumber][0],
SAMPLES[sampleNumber][1]);
        dataSet.add(newData);
        minimum = bigNumber;
        for(int i = 0; i < NUM_CLUSTERS; i++)
        {
            distance = dist(newData, centroids.get(i));
            if(distance < minimum){
                minimum = distance;
                cluster = i;
            }
        }
        newData.cluster(cluster);

        // calculate new centroids.
        for(int i = 0; i < NUM_CLUSTERS; i++)
        {
            int totalX = 0;
            int totalY = 0;
            int totalInCluster = 0;
            for(int j = 0; j < dataSet.size(); j++)
            {
                if(dataSet.get(j).cluster() == i){
                    totalX += dataSet.get(j).X();
                    totalY += dataSet.get(j).Y();
                    totalInCluster++;
                }
            }
        }
    }
}

```

```

        if(totalInCluster > 0){
            centroids.get(i).X(totalX / totalInCluster);
            centroids.get(i).Y(totalY / totalInCluster);
        }
    }
    sampleNumber++;
}

// Now, keep shifting centroids until equilibrium occurs.
while(isStillMoving)
{
    // calculate new centroids.
    for(int i = 0; i < NUM_CLUSTERS; i++)
    {
        int totalX = 0;
        int totalY = 0;
        int totalInCluster = 0;
        for(int j = 0; j < dataSet.size(); j++)
        {
            if(dataSet.get(j).cluster() == i){
                totalX += dataSet.get(j).X();
                totalY += dataSet.get(j).Y();
                totalInCluster++;
            }
        }
        if(totalInCluster > 0){
            centroids.get(i).X(totalX / totalInCluster);
            centroids.get(i).Y(totalY / totalInCluster);
        }
    }

    // Assign all data to the new centroids
    isStillMoving = false;

    for(int i = 0; i < dataSet.size(); i++)
    {
        Data tempData = dataSet.get(i);
        minimum = bigNumber;
        for(int j = 0; j < NUM_CLUSTERS; j++)
        {
            distance = dist(tempData, centroids.get(j));
            if(distance < minimum){
                minimum = distance;
                cluster = j;
            }
        }
    }
}

```

```

        tempData.cluster(cluster);
        if(tempData.cluster() != cluster){
            tempData.cluster(cluster);
            isStillMoving = true;
        }
    }
}
return;
}

/**
 * // Calculate Euclidean distance.
 * @param d - Data object.
 * @param c - Centroid object.
 * @return - double value.
 */
private static double dist(Data d, Centroid c)
{
    return Math.sqrt(Math.pow((c.Y() - d.Y()), 2) + Math.pow((c.X() - d.X()), 2));
}

private static class Data
{
    private double mX = 0;
    private double mY = 0;
    private int mCluster = 0;

    public Data()
    {
        return;
    }

    public Data(double x, double y)
    {
        this.X(x);
        this.Y(y);
        return;
    }

    public void X(double x)
    {
        this.mX = x;
        return;
    }

    public double X()

```

```

    {
        return this.mX;
    }

    public void Y(double y)
    {
        this.mY = y;
        return;
    }

    public double Y()
    {
        return this.mY;
    }

    public void cluster(int clusterNumber)
    {
        this.mCluster = clusterNumber;
        return;
    }

    public int cluster()
    {
        return this.mCluster;
    }
}

private static class Centroid
{
    private double mX = 0.0;
    private double mY = 0.0;

    public Centroid()
    {
        return;
    }

    public Centroid(double newX, double newY)
    {
        this.mX = newX;
        this.mY = newY;
        return;
    }

    public void X(double newX)
    {

```

```

        this.mX = newX;
        return;
    }

    public double X()
    {
        return this.mX;
    }

    public void Y(double newY)
    {
        this.mY = newY;
        return;
    }

    public double Y()
    {
        return this.mY;
    }
}

public static Vector<List> applyKMeans()
{
    pointList= Utility.getList();
    TOTAL_DATA= pointList.size();
    for(int i= 0; i<TOTAL_DATA; i++)
    {
        Point p= pointList.get(i);
        SAMPLES[i][0]=p.getX();
        SAMPLES[i][1]= p.getY();
    }
    initialize();
    kMeanCluster();

    List<Point> cluster[] = (ArrayList<Point>[]) new ArrayList[2];
    cluster[0] = new ArrayList<Point>();
    cluster[1]= new ArrayList<Point>();
    // Print out clustering results.
    for(int i = 0; i < NUM_CLUSTERS; i++)
    {
        System.out.println("Cluster " + i + " includes:");
        for(int j = 0; j < TOTAL_DATA; j++)
        {
            if(dataSet.get(j).cluster() == i){
                Point p = new Point(dataSet.get(j).X(),dataSet.get(j).Y());
                cluster[i].add(p);
            }
        }
    }
}

```

```
        System.out.println(" (" + dataSet.get(j).X() + ", " + dataSet.get(j).Y() + ")");
    }
} // j
System.out.println();
} // i

// Print out centroid results.
System.out.println("Centroids finalized at:");
for(int i = 0; i < NUM_CLUSTERS; i++)
{
    System.out.println(" (" + centroids.get(i).X() + ", " + centroids.get(i).Y());
}
System.out.print("\n");
resultList.add(cluster[0]);
resultList.add(cluster[1]);
return resultList;
}
}
```


CHAPTER 4

RESULT

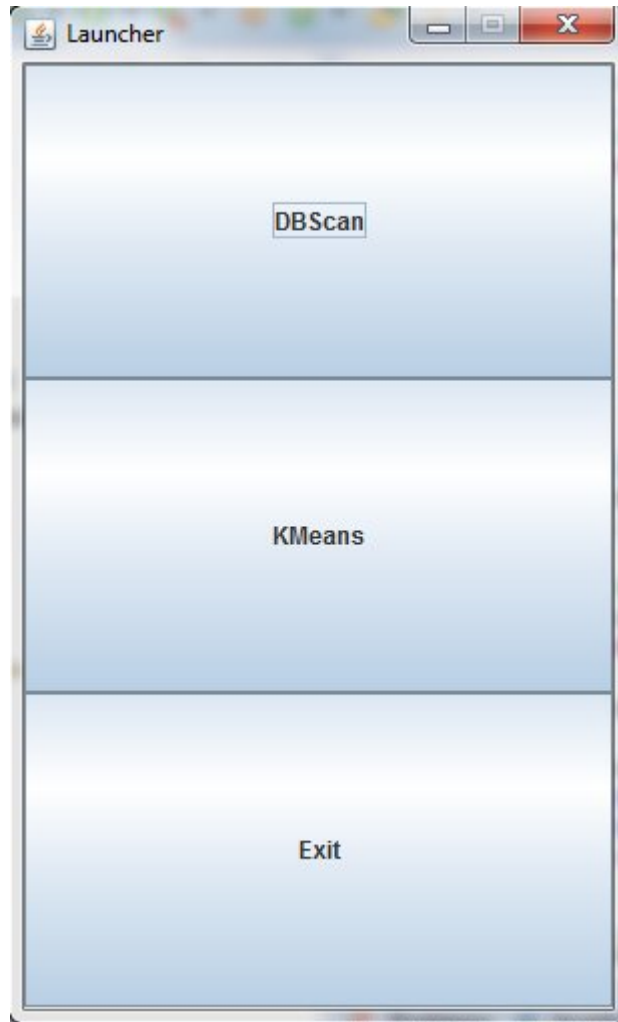
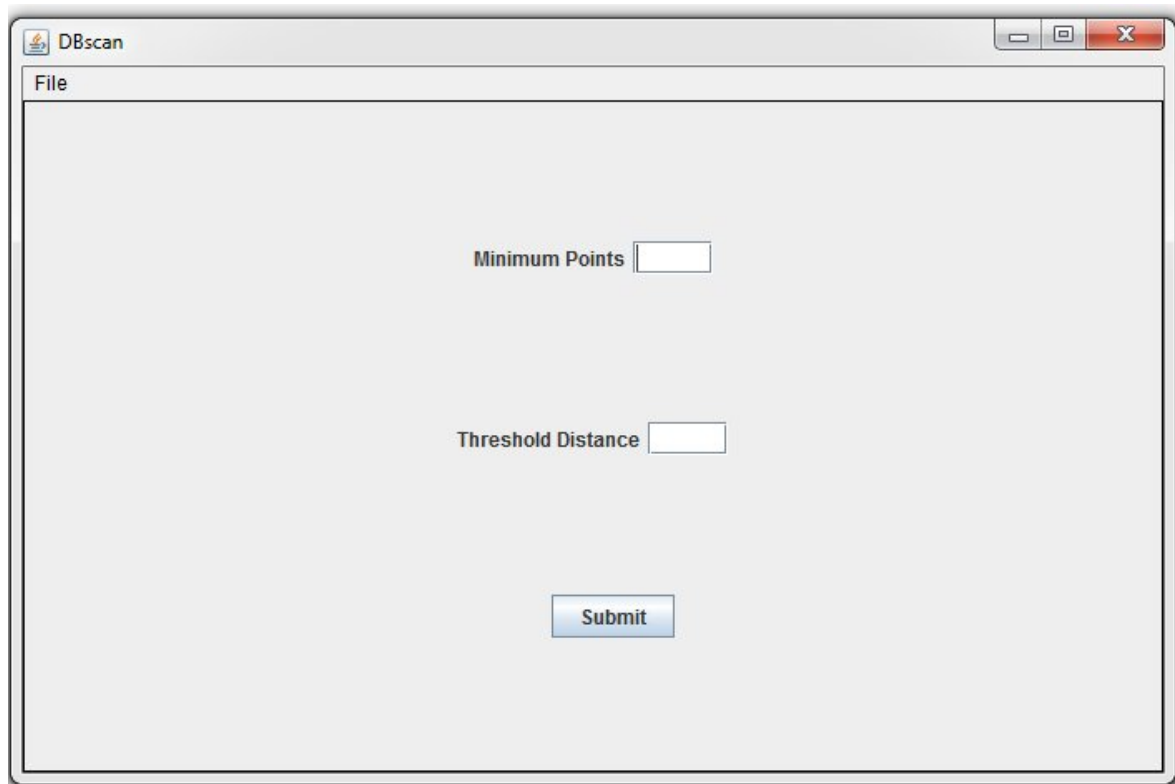


Figure 4.1 Launcher enabling to select an algorithm

4.1 Results for DBSCAN



The image shows a window titled "DBscan" with a "File" menu. The main area contains two input fields: "Minimum Points" and "Threshold Distance", each followed by a text box. Below these fields is a "Submit" button.

Figure 4.2 DBSCAN asking input parameters

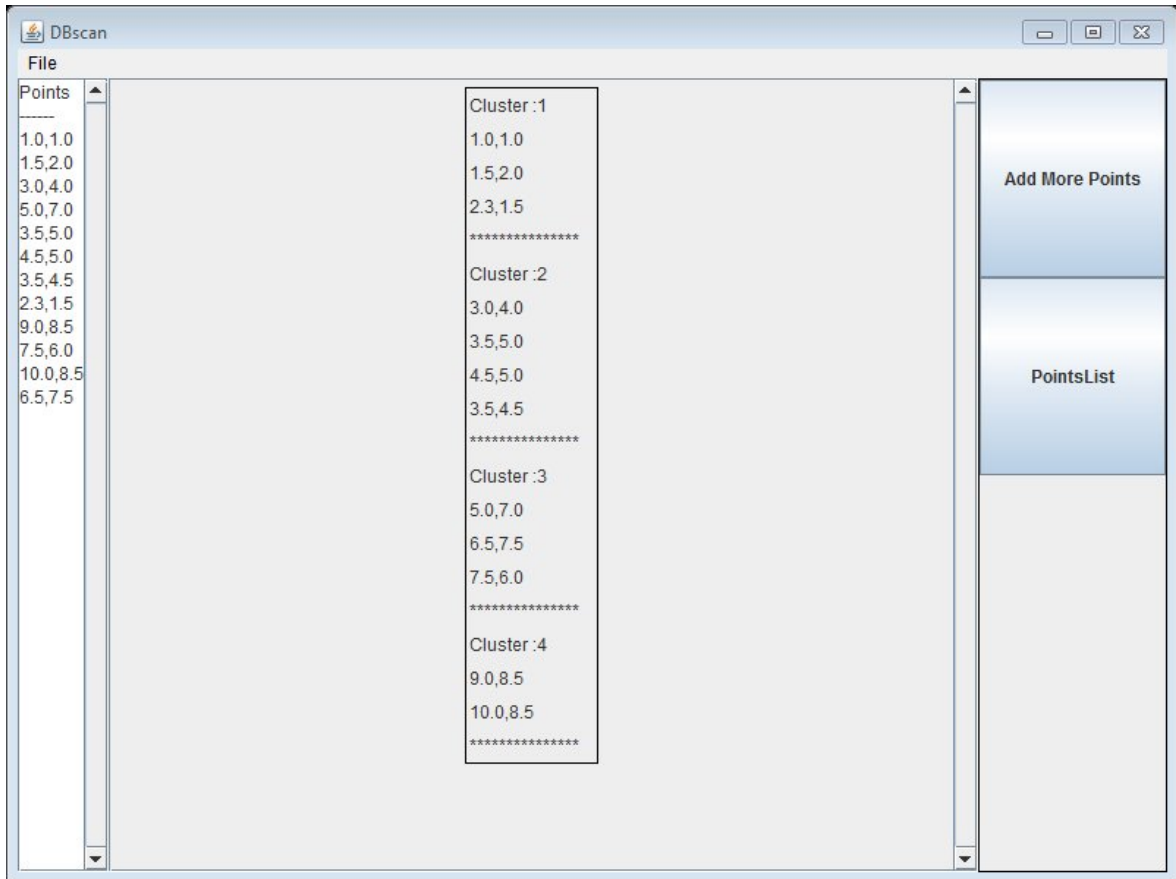


Figure 4.3DBSCAN showing Clusters

4.2 Results for K-Means

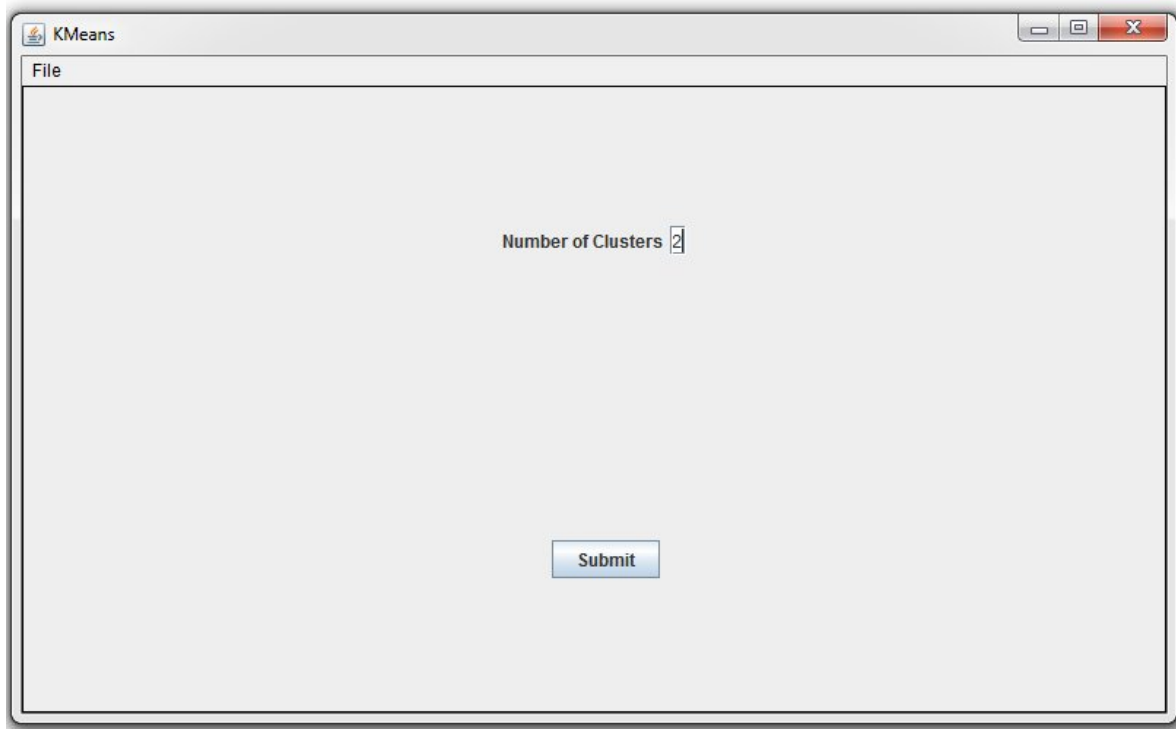


Figure 4.4 k-means asking input parameter



Figure 4.5 Entering Points

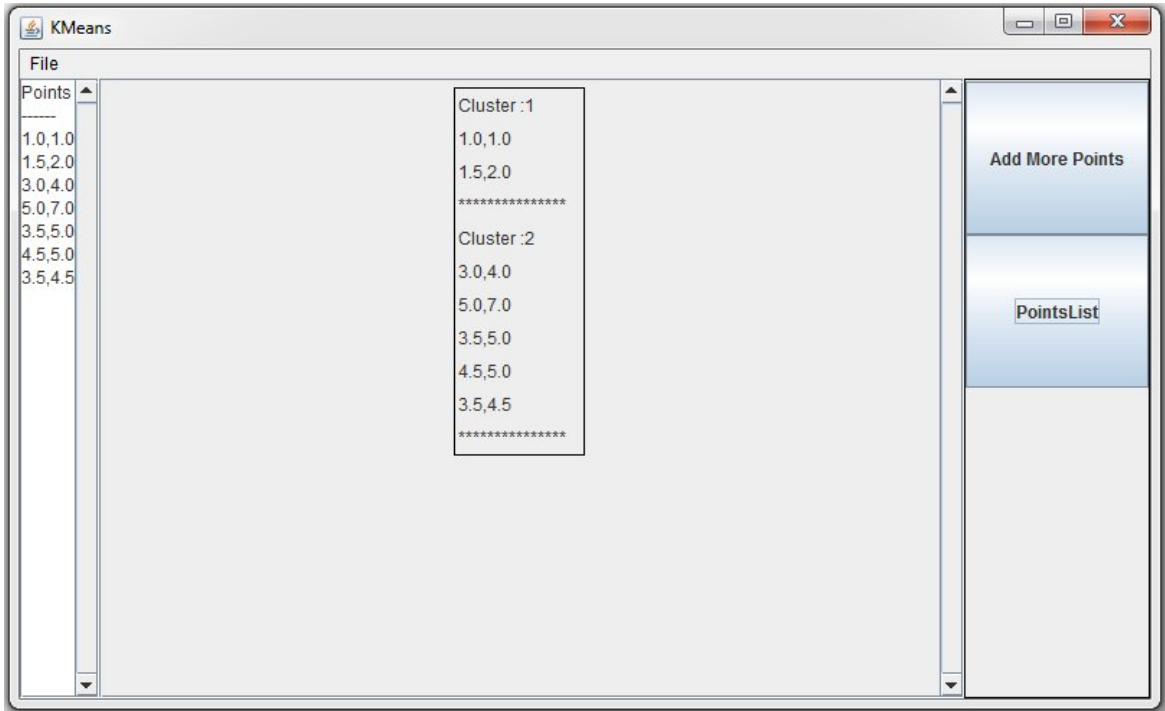


Figure 4.6 k-means showing clusters

CHAPTER 5

CONCLUSION

The report provides an insight about the formal study of algorithms and methods of grouping objects. An object is described either by a set of measurements or by relationships between the object and other objects. A comparative analysis of clustering methods is useful since empirical evidence seems to be the only practical guide in the selection of clustering methods.

This report presents a comparative study of the clustering technique K-means and DBSCAN. This study indicate that DBSCAN is better than K-means in discovering non-convex clusters and it is as good or better than K-means in extracting convex clusters. It can also detect noise. However, DBSCAN algorithm also takes much more CPU time for large data sets. When the clusters are of arbitrary shape, a density based clustering algorithm like DBSCAN is preferable. On the other hand, when the clusters are of hyper spherical or convex shape and well separated and the data set is large, K-means may be preferable as it is faster.

REFERENCES

- [1] I. Witten, E. Frank and Mark Hall, *Data Mining*, 3rd ed. San Francisco: Morgan Kaufmann Publishers, 2011
- [2] Glenn Fung, *A Comprehensive Overview of Basic Clustering Algorithms*, 2001
- [3] Jiawei Han and Micheline Kamber, *Data Mining*, 2nd ed. San Francisco: Morgan Kaufmann Publishers, 2006
- [4] Yujie Zheng, “*Clustering Methods in Data Mining with its Applications in High Education*” International Conference on Education Technology and Computer (ICETC2012), IPCSIT vol.43, 2012
- [5] Anil K. Jain and Richard C. Dubes, *Algorithms for Clustering Data*, New Jersey: Prentice Hall, 1988
- [6] A.K. Jain, M.N. Murty and P.J.Flynn, “*Data Clustering: A review*”.
- [7] Lior Rokach and Oded Maimon, *Clustering Methods*, Department of Industrial Engineering, Tel-Aviv University
- [8] J.MacQueen. *Some methods for classification and analysis of multivariate observations*. In L.M. LeCam and J. Neyman, editors, Proceedings of the Fifth Berkley Symposium on Mathematical Statistics and Probability, vol.1, pages 281-297, Berkley, CA, 1967.
- [9] Henrik Backlund, Anders Hedblom, Niklas Neijman, *A Density-Based Spatial Clustering of Application with Noise*, Linkopings University, 2011
- [10] O.J. Oyelade, O.O. Oladipupo and I.C. Obagbuwa, *Application of k-means clustering algorithm for prediction of students' academic performance*
- [11] Ting Liu, Charles Rosenberg and H.A. Rowley, “Clustering Billions of Images with Large Scale Nearest Neighbor Search”