

Learning Resource Centre-JUIT



SP06085

# **PREDICTION TOOL BASED ON CLUSTERING**

**By**

**TUSHAR GUPTA – 061303**

**KHUSHBOO BHATIA – 061423**



**MAY-2010**

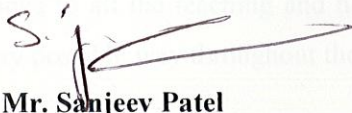
**Submitted in partial fulfillment of the Degree of Bachelor of  
Technology in Computer Science & Engineering**

**DEPARTMENT OF COMPUTER SCIENCE AND  
ENGINEERING AND INFORMATION TECHNOLOGY  
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY-  
WAKNAGHAT**



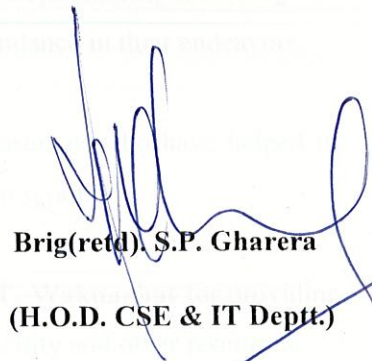
## CERTIFICATE

This is to certify that the work entitled, "Prediction Tool Based on Clustering" submitted by **Tushar Gupta (061303)** and **Khushboo Bhatia (061423)** in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering of Jaypee University of Information Technology 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.



**Mr. Sanjeev Patel**

**(Project Guide)**



**Brig(retd). S.P. Gharera**

**(H.O.D. CSE & IT Deptt.)**

## ACKNOWLEDGMENT

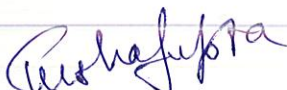
We extend our warm and sincere thanks to our project supervisor **Mr. Sanjeev Patel** who has been a staunch supporter and motivator of this project. Right from the inception of this project work, our supervisor **Mr. Sanjeev Patel** guided us till the very end in the true sense of the word. He always came up with innovative ways and creative terms, thus, also helping us to instill and enhance the quality of creative thinking within us.

Sincere thanks to Brig.(retd). **S.P. Gharera, HOD, CSE & IT Department**, for being co-operative to the students of the department and providing relevant guidance in their endeavors.

Thanks to all the teaching and non-teaching staff of the CSE Department who have helped in every possible way throughout the 4 years of the B.Tech. academic program.

We would also like to express our gratitude to this alma mater **JUIT, Wagnaghat** for providing proper resources as and when required such as an all time internet facility and other resources.

Hence, without giving a warm thanks to all of them who made this project work a reality, our work would be incomplete.

  
**TUSHAR GUPTA**

061303

  
**KHUSHBOO BHATIA**

061423



## TABLE OF CONTENTS

Certificate.....	i
Acknowledgement.....	ii
Table of Contents.....	iii

### Chapter 1: Introduction

1.1 Objective .....	1
1.2 General Description .....	1
1.3 Scope of the project .....	3
1.4 Modules and Timeline .....	4

### Chapter 2: Literature Survey

2.1 Non-Continuous and Continuous Attributes .....	6
2.1.1 Non - Continuous Attributes .....	6
2.1.1.1 Nominal Attributes .....	6
2.1.1.2 Ordinal Attributes .....	7
2.1.1.3 Binary Attributes .....	7
2.1.2 Continuous Attributes .....	7
2.2 Descretization of Attributes and Degree of Dependency .....	8
2.2.1 Descretization of Attributes .....	8
2.2.2 Degree of Dependency .....	9
2.2.2.1 Non - Continuous Attributes .....	10
2.2.2.2 Continuous Attributes .....	10
2.2.2.2.1 Clustering .....	11
2.2.2.2.2 Degree of Dependency using Clustering .....	28
2.3 Baye's Theorem.....	29

## **Chapter 3: Design and Implementation**

3.1 Degree of Dependency of Non-Continuous Attributes	
3.1.1 Modular Description .....	32
3.2 Clustering	
3.2.1 Modular Description .....	33
3.2.2 Algorithm .....	35
3.3 Degree of Dependency of Continuous Attributes	
3.3.1 Modular Description .....	39
3.3.2 Algorithm.....	39
3.4 Flowchart .....	40
3.5 Code	
3.5.1 Degree of Dependency of Non-Continuous Attributes.....	41
3.5.2 Clustering.....	45
3.5.3 Degree of Dependency of Continuous Attributes.....	54

## **Chapter 4: Results and Technology Used**

4.1 Degree of Dependency of Non-Continuous Attributes.....	68
4.2 Clustering.....	69
4.3 Degree of Dependency of Continuous Attributes.....	71
4.4 Technology and Resources Used.....	72

## **Chapter 5: Future Work and Conclusion**

## **Bibliography**

## CHAPTER – 1

### INTRODUCTION

#### 1.1 OBJECTIVE

To develop a tool for predicting the results for various domains. The tool can take both nominal and continuous attributes as inputs. The prediction for continuous attributes requires clustering, in order to convert it into a nominal form. For clustering we have used DBSCAN algorithm.

#### 1.2 GENERAL DESCRIPTION

Data analysis underlies many computing applications, either in a design phase or as part of their on-line operations. Data analysis procedures can be dichotomized as either exploratory or confirmatory, based on the availability of appropriate models for the data source, but a key element in both types of procedures (whether for hypothesis formation or decision-making) is the grouping, or classification of measurements based on either (i) goodness-of-fit to a postulated model, or (ii) natural groupings (clustering) revealed through analysis.

It is important to understand the difference between clustering (unsupervised classification) and discriminant analysis (supervised classification). In supervised classification, we are provided with a collection of *labeled* (pre classified) patterns; the problem is to label a newly encountered, yet unlabeled, pattern. Typically, the given labeled (*training*) patterns are used to learn the descriptions of classes which in turn are used to label a new pattern. A training set is a set of data used in various areas of information science to discover potentially predictive relationships. Training sets are used in artificial intelligence, machine learning, genetic programming, intelligent systems, and statistics. In all these fields, a training set has much the same role and is often used in conjunction with a test set. The algorithms for unsupervised discretization make no consideration for

the class attribute while the supervised discretization algorithms consider the interdependence between class labels and the attribute values. Clustering is the unsupervised classification of patterns (observations, data items) into groups (clusters). Cluster analysis is the organization of a collection of patterns into clusters based on similarity. Intuitively, patterns within a valid cluster are more similar to each other than they are to a pattern belonging to a different cluster. For this reason, clustering is a form of learning by observation, rather than learning by examples. The problem is to group a given collection of unlabeled patterns into meaningful clusters. In a sense, labels are associated with clusters also, but these category labels are *data driven*; that is, they are obtained solely from the data.

The process of Knowledge Discovery in Databases (KDD) commonly involves a number of sub-processes such as data selection, data preprocessing, data transformation, data mining, and the interpretation and visualization of the mined patterns. Some of these sub-processes are iterative in nature and sometimes some loop amongst themselves. Among these the data preprocessing and data transformation are steps of much interest because these steps prepare the data according to the input specification of the data mining algorithms.

The necessary tasks in preprocessing and transformation are data integration to create single data matrix from data tables obtained from various resources, creating attributes, e excluding irrelevant attributes, discretization of the values of attributes. Discretization transforms the infinitely many continuous values of an attribute into a finite and a significantly small numbers of intervals. For continuous attributes , in order to find degree of dependency , we first cluster them into groups of similar data objects and then treat the clusters as nominal attributes , thus , finding their degree of dependency. The majority of the Data Mining algorithms are applied to data described by discrete or nominal attributes. In order to apply these algorithms effectively to any dataset the continuous attribute need to be transformed to discretized ones.



### 1.3 SCOPE OF THE PROJECT

Numerous applications require the management of data. Increasingly large amounts of data are obtained from various sources .

Therefore, automated knowledge discovery becomes more and more important.

Clustering algorithms are attractive for the task of class identification. However, due to the application to large databases arise the following requirements for clustering algorithms:

- (1) Minimal requirements of domain knowledge to determine the input parameters, because appropriate values are often not known in advance when dealing with large databases.
- (2) Discovery of clusters with arbitrary shape, because the shape of clusters in spatial databases may be spherical, drawn-out, linear, elongated etc.
- (3) Good efficiency on large databases, i.e. on databases of significantly more than just a few thousand objects.
- (4) Capability of discovery of knowledge for diverse domains.

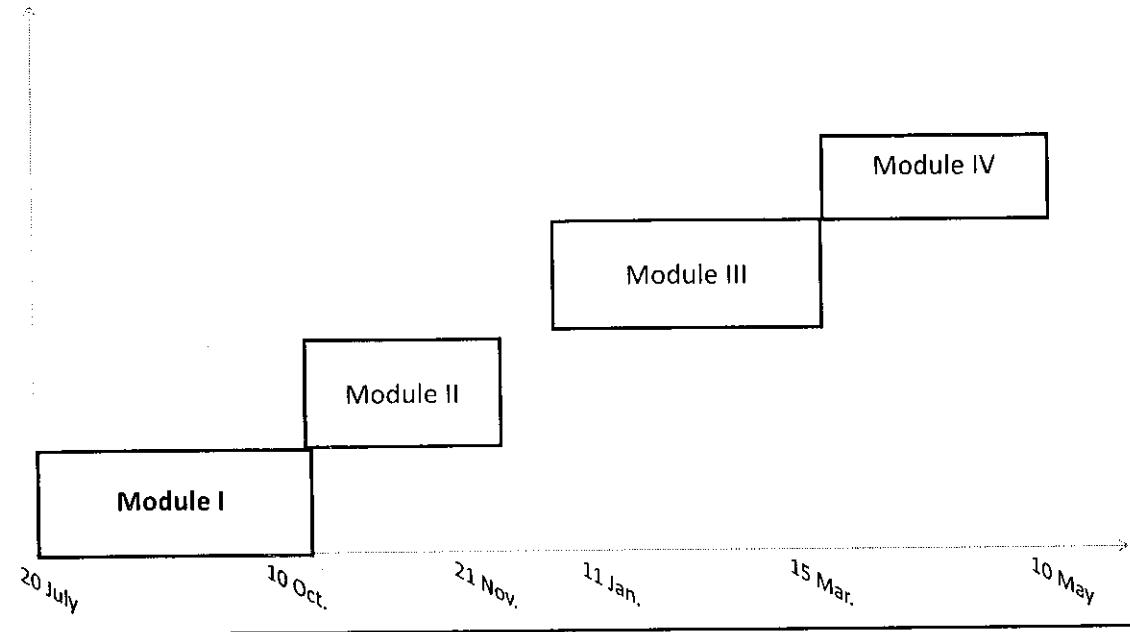
Hence , our project i.e. "A PREDICTION TOOL BASED ON DBSCAN " aims at prediction from information provided from different domains and producing consistent results for highly scalable data , since the amount of information available is huge .

## 1.4 MODULES AND TIMELINE

### MODULES:

MODULES	DESCRIPTION
MODULE - I	Finding degree of dependency for non-continuous attributes.
MODULE - II	The second module comprises of working on clustering by providing indices to the clusters.
MODULE - III	Our aim is to find the degree of dependency of continuous attributes.
MODULE - IV	After successful completion of Module -- I, II, and III , the final prediction of results will take place.

**TIMELINE:**





## **CHAPTER – 2**

### **LITERATURE SURVEY**

#### **2.1 NON – CONTINUOUS AND CONTINUOUS ATTRIBUTES**

##### **2.1.1 NON – CONTINUOUS ATTRIBUTES**

###### **2.1.1.1 NOMINAL VARIABLES**

- Nominal variable is a variable with values whose order is insignificant (synonym : categorical variable) i.e. a variable for which values represent the names of things, with no order implied.
- Nominal variable is a generalization of a binary variable in that it can take on more than two states.
- Nominal variables classify data into categories. This process involves labeling categories and then counting frequencies of occurrence.
- Let the number of states of a nominal variable be  $M$ . The states can be denoted by letters , symbols, or a set of integers, such as 1, 2, ...,  $M$ . Notice that such integers are used just for data handling and do not represent any specific ordering.
- E.g., profession, ID numbers, eye color, zip codes.

### **2.1.1.2 ORDINAL ATTRIBUTES**

- A discrete ordinal variable resembles a nominal variable except that the  $M$  states of the ordinal value are ordered in a certain meaningful sequence.
- E.g., rankings (e.g., army, professions), grades, height in {tall, medium, short}

### **2.1.1.3 BINARY ATTRIBUTES**

- A binary variable has only two states: 0 and 1.
- 0 shows absence of certain feature and 1 shows the presence.
- E.g., medical test (positive vs. negative)

### **2.1.2 CONTINUOUS ATTRIBUTES**

- A continuous variable is one for which, within the limits the variable ranges, any value is possible. Thus, they can take any value over a defined range.
- Has real numbers as attribute values.
- Examples: temperature, height, or weight, etc.
- Practically, real values can only be measured and represented using a finite number of digits.
- Continuous attributes are typically represented as floating-point variables.

- Example :

Consider the following measurements:

- times to run a marathon
- temperatures recorded at intervals during a day
- weight of each bunch of grapes sold at a supermarket yesterday.

Time, temperature and weight are all examples of numerical data, but there is not a restricted set of values that they can take. Whereas you can have 2 or 3 children in a family but not 2.5, with temperature it is possible to have not only 22 °C and 23 °C but also 22.1 °C, 22.25 °C, 22.97 °C as well. This type of variable is restricted only by the accuracy with which the measurement can be made. Such variables are known as 'continuous variables'.

- Continuous variables often relate to *measured* items.

## **2.2 DESCRETIZATION OF ATTRIBUTES AND DEGREE OF DEPENDENCY**

### **2.2.1 DESCRETIZATION OF ATTRIBUTES**

The majority of the Data Mining algorithms are applied to data described by discrete or nominal attributes. The large number of attribute values in database slows down the process of better discovery of the discrete intervals thus making inductive learning ineffective. Discretization techniques can be used to reduce the number of values for a given continuous attribute, by dividing the range of attributes into intervals or by applying clustering techniques. Discrete attributes have only a finite or count ably infinite set of values.



## 2.2.2 DEGREE OF DEPENDENCY

- A set of attributes D depends totally on a set of attributes C, denoted  $C \Rightarrow D$ , if all attribute values from D are uniquely determined by values of attributes from C.
- By calculating the change in dependency when an attribute is removed from the set of considered conditional attributes, a measure of the significance of the attribute can be obtained.
- The higher the change in dependency, the more significant the attribute is. If the significance is 0, then the attribute is dispensable.
- By calculating the change in dependency when an attribute is removed from the set of considered conditional attributes, a measure of the significance of the attribute can be obtained.
- The higher the change in dependency, the more significant the attribute is. If the significance is 0, then the attribute is dispensable.

### **2.2.2.1 DEGREE OF DEPENDENCY OF NON - CONTINUOUS ATTRIBUTES**

Nominal variable is a variable for which values represent the names of things, with no order implied.

For example : Attributes for weather can be clear, sunny, cloudy or rainy which are nominal and that of temperature can vary along any range and these will be continuous.

Nominal variables classify data into categories. This process involves labeling categories and then counting frequencies of occurrence.

The classes with maximum frequency of occurrences are then recorded and degree of dependency is calculated based on them.

For example : A researcher might wish to compare essay grades between male and female students. Tabulations would be compiled using the categories "male" and "female." Sex would be a nominal variable. Note that the categories themselves are not quantified. Maleness or femaleness are not numerical in nature, rather the frequencies of each category results in data that is quantified -- 11 males and 9 females.

### **2.2.2.2 DEGREE OF DEPENDENCY OF CONTINUOUS ATTRIBUTES**

The majority of the Data Mining algorithms are applied to data described by discrete or nominal attributes. The large number of attribute values in database slows down the process of better discovery of the discrete intervals thus making inductive learning ineffective. Therefore, one of the main goals of any discretization process is to significantly reduce the number of intervals for the values of the continuous attribute under consideration. In order to apply data mining algorithms effectively to any dataset the continuous attribute need to be transformed to discretized ones.

A continuous variable is one for which, within the limits the variable ranges, any value is possible. Thus, they can take any value over a defined range.



Thus, in order to find degree of dependency for continuous attributes , one needs to convert them into nominal attributes and then find their degree of dependency.



To convert continuous attributes to nominal variables we've used clustering (DBSCAN) technique.

#### **2.2.2.2.1 CLUSTERING**

The process of grouping of 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. 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 uses 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(e.g. using clustering), 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.

Cluster analysis is an important human activity. By automated clustering, we can identify dense and sparse regions in object space and, therefore, discover overall distribution patterns and interesting correlations among data attributes. Cluster analysis has been widely used in numerous



applications, including market research, pattern recognition, data analysis and image processing. In business, clustering can help marketers discover distinct groups in their customer bases and characterize customer groups based on purchasing patterns. In biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality, and gain insight into structures inherent in populations. Clustering may also help in the identification of groups of houses in a city according to house type, value, and geographic location as well as the identification of groups of automobile insurance policy holders with a high average claim cost. It can also be used to classify documents on the Web for information discovery.

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. Applications of outlier detection include the detection of credit card fraud and the monitoring of criminal activities in electronic commerce. For example, exceptional cases in credit card transactions, such as very expensive and frequent purchases, may be of interest as possible fraudulent activity. As a data mining function, cluster analysis can be used as a stand-alone tool to gain insight into the distribution of data, to observe the characteristics of each cluster, and to focus on a particular set of clusters for further analysis. Alternatively, it may serve as a preprocessing step for other algorithms, such as characterization, attribute subset selection, and classification, which would then operate on selected clusters and the selected attributes or features.

Clustering is the **unsupervised classification** of patterns (observations, data items) into groups (clusters). Clustering is useful in several exploratory pattern-analysis, grouping, decision-making, and machine-learning situations, including data mining, document retrieval, image segmentation, and pattern classification. However, in many such problems, there is little prior information (e.g., statistical models) available about the data, and the decision-maker must make as few assumptions about the data as possible. It is under these restrictions that clustering methodology is particularly appropriate for the exploration of interrelationships among the data points to make an assessment (perhaps preliminary) of their structure. In the case of clustering, the problem is to group a given collection of unlabeled patterns into meaningful clusters. In a

sense, labels are associated with clusters also, but these category labels are *data driven*; that is, they are obtained solely from the data.

A good clustering method will produce high quality clusters with

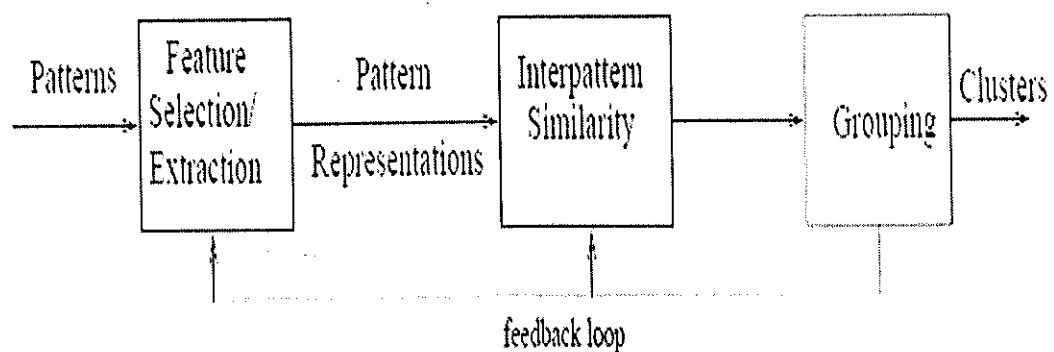
- high intra-class similarity
- low inter-class similarity

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.

### Components of a Clustering Task

Typical pattern clustering activity involves the following :

- (1) pattern representation (optionally including feature extraction and/or selection),
- (2) definition of a pattern proximity measure appropriate to the data domain,
- (3) clustering or grouping,
- (4) data abstraction (if needed), and
- (5) assessment of output (if needed).



Stages in Clustering

Figure 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 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 (or clusterings) 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 (so that a machine can perform further processing efficiently) or it is human-oriented (so that the representation obtained is easy to comprehend and intuitively appealing). In the clustering context, a typical data abstraction is a compact description of each cluster, usually in terms of cluster prototypes or representative patterns.



### **Requirements of Clustering in Data Mining :**

- Scalability
- Ability to deal with different types of attributes
- Discovery of clusters with arbitrary shape
- Minimal requirements for domain knowledge to determine input parameters
- Able to deal with noise and outliers
- Insensitive to order of input records
- High dimensionality
- Incorporation of user-specified constraints
- Interpretability and usability

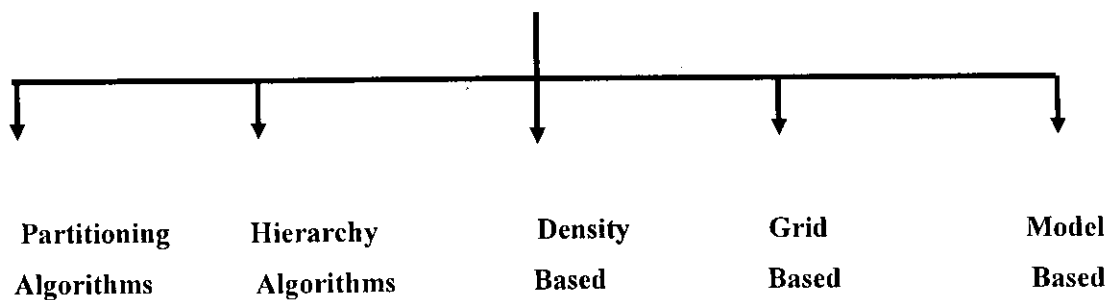
### **Typical applications :**

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

### General Applications of Clustering :

- Pattern Recognition
- Spatial Data Analysis
  - detect spatial clusters and explain them in spatial data mining
- Image Processing
- Economic Science (especially market research)
- WWW
  - Document classification

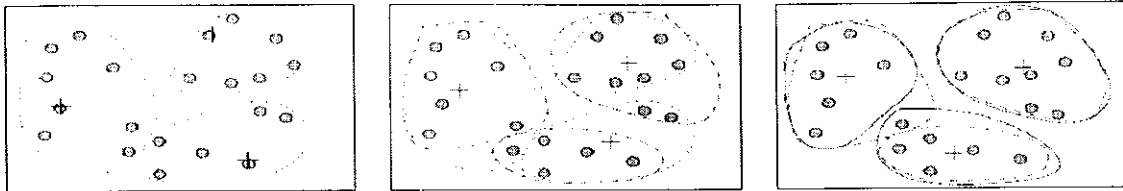
### Major Clustering Approaches



- **Partitioning algorithms:** Construct various partitions and then evaluate them by some criterion. Construct a partition of a database  $D$  of  $n$  objects into a set of  $k$  clusters. Given a  $k$ , find a partition of  $k$  clusters that optimizes the chosen partitioning criterion. Global optimal: exhaustively enumerate all partitions.

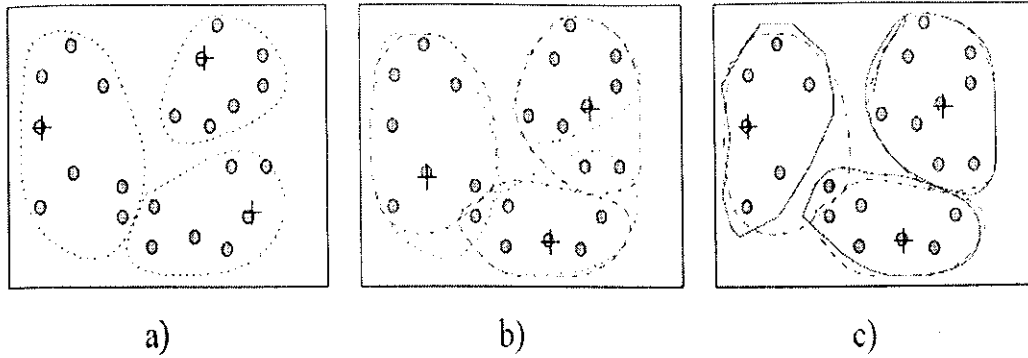
Heuristic methods: *k-means* and *k-medoids* algorithms

***k-means*:** Each cluster is represented by the center of the cluster.



**Clustering of set of points using k-means method**

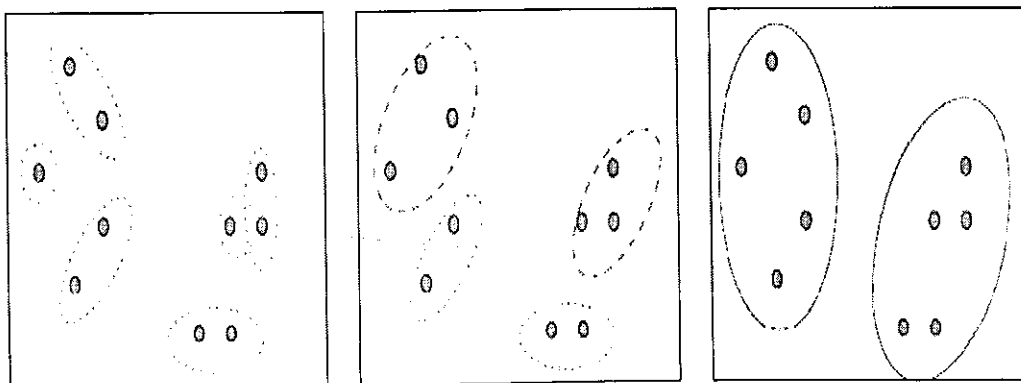
- ***k-medoids* or PAM (Partition around medoids):** Basically, the algorithm finds the center of a cluster and takes the element closest to the center as "the Medoid" means most centrally located object in a cluster. Starts from an initial set of medoids and iteratively replaces one of the medoids by one of the non-medoids if it improves the total distance of the resulting clustering. It is performed based on the principle of minimizing the sum of the dissimilarities between each object. *PAM* works effectively for small data sets, but does not scale well for large data sets.



### Clustering of set of points based on k-medoids algorithm

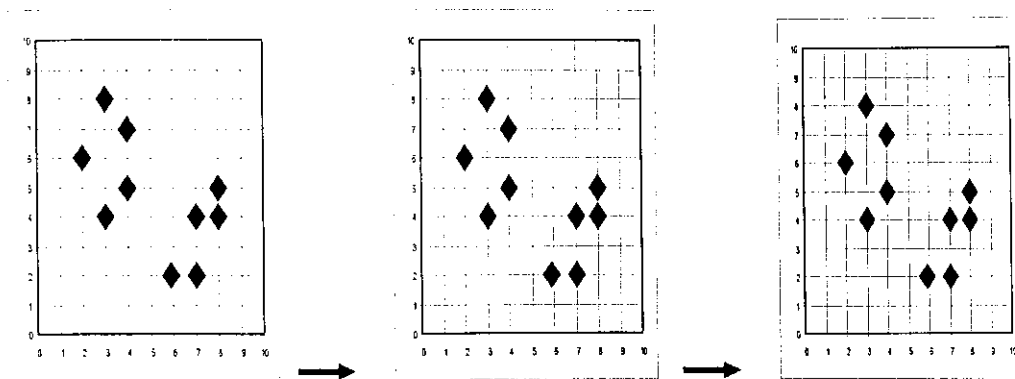
- **Hierarchy algorithms:** It creates a hierarchical decomposition of the given set of data objects. It can be classified into either **agglomerative** or **divisive**.

**Agglomerative approach(bottom-up):** set each object as a individual cluster or group and merges the objects or groups close to one another, until all of the groups are merged into one(the topmost level of the hierarchy).



### Clustering of a set of points based on the “Agglomerative Nesting” method

**Divisive approach(top down):** starts with all objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until a termination condition holds.



### DIANA(Divisive Approach)

- **Density-based:** based on connectivity and density functions. To discover the clusters with arbitrary shapes density based clustering methods have been proposed. Clusters are regarded as the dense regions of the objects, separated by regions of low density.

#### Major features:

- Discover clusters of arbitrary shape
- Handle noise
- Need density parameters



**Two parameters:**

- : Maximum radius of the neighborhood
- **MinPts**: Minimum number of points in an neighborhood of that point.

$$\varepsilon \text{ neighborhood of that point: } N_{\varepsilon}(p) : \{q \in D \mid \text{dist}(p, q) \leq \varepsilon\}$$

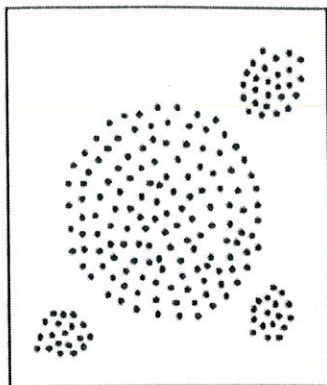
#### **Example: DBSCAN Algorithm –**

Clustering algorithms are attractive for the task of class identification. However, the application to large databases rises the following requirements for clustering algorithms:

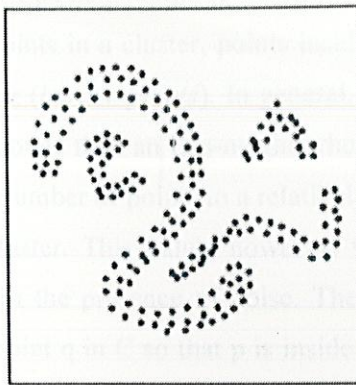
- (1) Minimal requirements of domain knowledge to determine the input parameters, because appropriate values are often not known in advance when dealing with large databases.
- (2) Discovery of clusters with arbitrary shape, because the shape of clusters in spatial databases may be spherical, drawn-out, linear, elongated etc.
- (3) Good efficiency on large databases, i.e. on databases of significantly more than just a few thousand objects.

DBSCAN refers to **Density Based Spatial Clustering of applications with Noise**. It Relies on a *density-based* notion of cluster: A *cluster* is defined as a maximal set of density-connected points. It was able to discover clusters of arbitrary shape in spatial databases with noise.

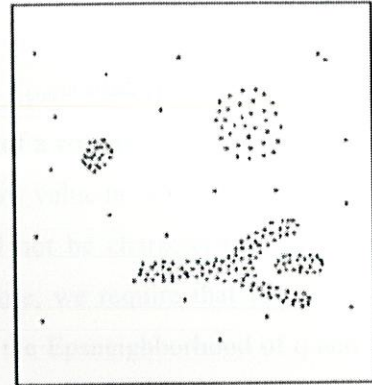
When looking at the sample sets of points depicted in figure, we can easily and unambiguously detect clusters of points and noise points not belonging to any of those clusters.



database 1



database 2



database 3

The main reason why we recognize the clusters is that within each cluster we have a typical density of points which is considerably higher than outside of the cluster. Furthermore, the density within the areas of noise is lower than the density in any of the clusters.

In the following, we try to formalize this intuitive notion of “clusters” and “noise” in a database  $D$  of points of some  $k$ -dimensional space  $S$ . Note that both, our notion of clusters and our algorithm DBSCAN, apply as well to 2D or 3D Euclidean space as to some high dimensional feature space. The key idea is that for each point of a cluster the neighborhood of a given radius has to contain at least a minimum number of points, i.e. the density in the neighborhood has to exceed some threshold. The shape of a neighborhood is determined by the choice of a distance function for two points  $p$  and  $q$ , denoted by  $dist(p, q)$ . For instance, when using the Manhattan distance in 2D space, the shape of the neighborhood is rectangular. Note, that our approach works with any distance function so that an appropriate function can be chosen for some given application. For the purpose of proper visualization, all examples will be in 2D space using the Euclidean distance.

**Definition 1:** (Eps-neighborhood of a point) The *Epsneighborhood* of a point  $p$ , denoted by  $NEps(p)$ , is defined by

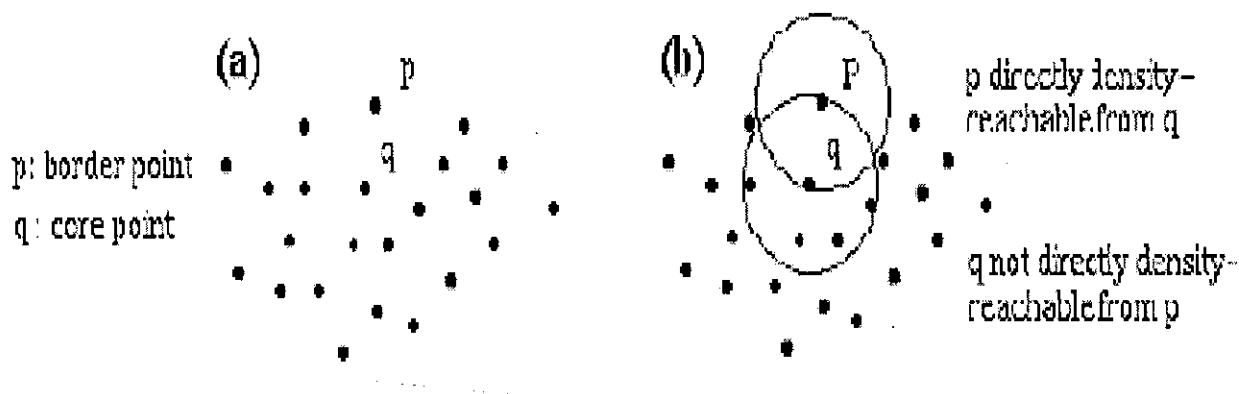
$$NEps(p) = \{q \in D \mid dist(p, q) < Eps\}.$$

A naive approach could require for each point in a cluster that there are at least a minimum number (*MinPts*) of points in an Eps-neighborhood of that point. However, this approach fails because there are two kinds of points in a cluster, points inside of the cluster (*core points*) and points on the border of the cluster (*border points*). In general, an Epsneighborhood of a border point contains significantly less points than an Eps-neighborhood of a core point. Therefore, we would have to set the minimum number of points to a relatively low value in order to include all points belonging to the same cluster. This value, however, will not be characteristic for the respective cluster - particularly in the presence of noise. Therefore, we require that for every point  $p$  in a cluster  $C$  there is a point  $q$  in  $C$  so that  $p$  is inside of the Epsneighborhood of  $q$  and  $Eps(q)$  contains at least *MinPts* points.

**Definition 2:** (directly density-reachable) A point  $p$  is *directly density-reachable* from a point  $q$  wrt.  $Eps$ , *MinPts* if

- 1)  $p \in NEps(q)$  and
- 2)  $|NEps(q)| \geq MinPts$  (core point condition)

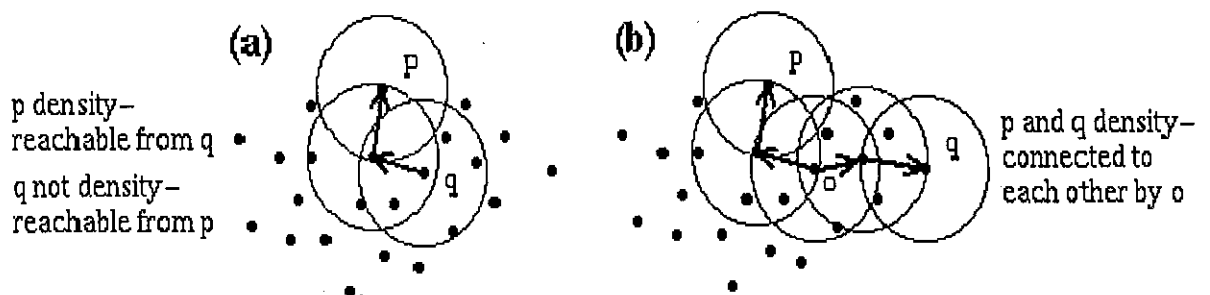
Obviously, directly density-reachable is symmetric for pairs of core points. In general, however, it is not symmetric if one core point and one border point are involved. Figure shows the asymmetric case.



**Definition 3:** (density-reachable) A point  $p$  is *densityreachable* from a point  $q$  wrt.  $Eps$  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  $p_i$ . Density-reachability is a canonical extension of direct density-reachability. This relation is transitive, but it is not symmetric. Figure depicts the relations of some sample points and, in particular, the asymmetric case. Although not symmetric in general, it is obvious that density-reachability is symmetric for core points. Two border points of the same cluster  $C$  are possibly not density reachable from each other because the core point condition might not hold for both of them. However, there must be a core point in  $C$  from which both border points of  $C$  are density-reachable. Therefore, we introduce the notion of density-connectivity which covers this relation of border points.

**Definition 4:** (density-connected) A point  $p$  is *densityconnected* to a point  $q$  wrt.  $Eps$  and  $MinPts$  if there is a point  $o$  such that both,  $p$  and  $q$  are density-reachable from  $o$  wrt.  $Eps$  and  $MinPts$ . Density-connectivity is a symmetric relation. For density reachable points, the relation of density-connectivity is also reflexive.

Now, we are able to define our density-based notion of a cluster. Intuitively, a cluster is defined to be a set of densityconnected points which is maximal wrt. density-reachability. Noise will be defined relative to a given set of clusters. Noise is simply the set of points in  $D$  not belonging to any of its clusters.



Density-reachability and Density connectivity

**Definition 5:** (cluster) Let  $D$  be a database of points. A *cluster*  $C$  wrt.  $Eps$  and  $MinPts$  is a non-empty subset of  $D$  satisfying the following conditions:

1)  $p, q$ : if  $p \in C$  and  $q$  is density-reachable from  $p$  wrt.  $Eps$  and  $MinPts$ , then  $q \in C$ .  
(Maximality)

2)  $\forall p, q \in C$ :  $p$  is density-connected to  $q$  wrt.  $Eps$  and  $MinPts$ . (Connectivity)

**Definition 6:** (noise) Let  $C_1, \dots, C_k$  be the clusters of the database  $D$  wrt. parameters  $Eps_i$  and  $MinPts_i$ ,  $i = 1, \dots, k$ . Then we define the *noise* as the set of points in the database  $D$  not belonging to any cluster  $C_i$ , i.e.

$$noise = \{p \in D \mid \forall i: p \notin C_i\}.$$

Note that a cluster  $C$  wrt.  $Eps$  and  $MinPts$  contains at least  $MinPts$  points because of the following reasons. Since  $C$  contains at least one point  $p$ ,  $p$  must be density-connected to itself via some point  $o$  (which may be equal to  $p$ ). Thus, at least  $o$  has to satisfy the core point condition and, consequently, the  $Eps$ -Neighborhood of  $o$  contains at least  $MinPts$  points.

The following lemmata are important for validating the correctness of our clustering algorithm. Intuitively, they state the following. Given the parameters  $Eps$  and  $MinPts$ , we can discover a cluster in a two-step approach. First, choose an arbitrary point from the database satisfying the core point condition as a seed. Second, retrieve all points that are density-reachable from the seed obtaining the cluster containing the seed.



**Lemma 1:** Let  $p$  be a point in  $D$  and  $|NEps(p)| \in MinPts$ . Then the set  $O = \{o \mid o \in D \text{ and } o \text{ is density-reachable from } p \text{ wrt. } Eps \text{ and } MinPts\}$  is a cluster wrt.  $Eps$  and  $MinPts$ . It is not obvious that a cluster  $C$  wrt.  $Eps$  and  $MinPts$  are uniquely determined by *any* of its core points. However, each point in  $C$  is density-reachable from any of the core points of  $C$  and, therefore, a cluster  $C$  contains exactly the points which are density-reachable from an arbitrary core point of  $C$ .

**Lemma 2:** Let  $C$  be a cluster wrt.  $Eps$  and  $MinPts$  and let  $p$  be any point in  $C$  with  $|NEps(p)| \in MinPts$ . Then  $C$  equals to the set  $O = \{o \mid o \text{ is density-reachable from } p \text{ wrt. } Eps \text{ and } MinPts\}$ .

- **Grid-based:** based on a multiple-level granularity structure.
  - No distance computations.
  - Clustering is performed on summaries and not individual objects; complexity is usually  $O(\# \text{-populated-grid-cells})$  and not  $O(\# \text{objects})$ .
  - Easy to determine which clusters are neighboring
  - Shapes are limited to union of grid-cells

**Basic Grid-based Algorithm:**

- Define a set of grid-cells
- Assign objects to the appropriate grid cell and compute the density of each cell.
- Eliminate cells, whose density is below a certain threshold  $t$ .
- Form clusters from contiguous (adjacent) groups of dense cells (usually minimizing a given objective function).

- All the cluster boundaries are either horizontal or vertical, and no diagonal boundary is detected.

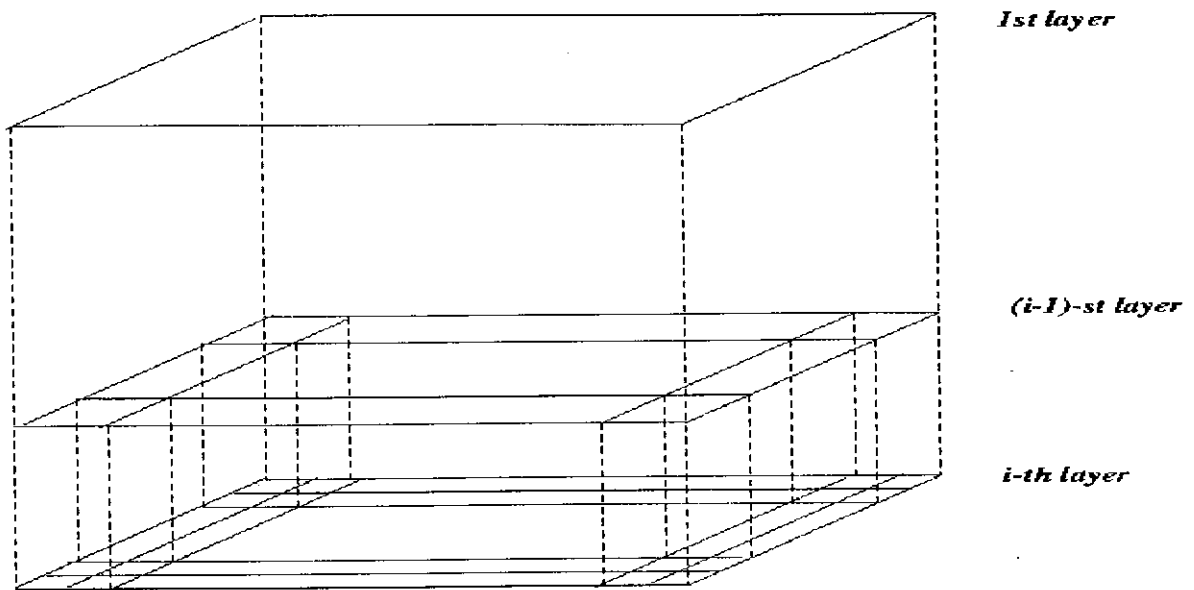


Figure: STING

- **Model-based:** A model is hypothesized for each of the clusters and the idea is to find the best fit of that model to each other.

#### **2.2.2.2.2 DEGREE OF DEPENDENCY USING CLUSTERING**

For continuous attributes , in order to find degree of dependency , we first cluster them into groups of similar data objects and then treat the clusters as nominal attributes , thus , finding their degree of dependency. The majority of the Data Mining algorithms are applied to data described by discrete or nominal attributes. The large number of attribute values in database slows down the process of better discovery of the discrete intervals thus making inductive learning ineffective. Therefore, one of the main goals of any discretization process is to significantly reduce the number of intervals for the values of the continuous attribute under consideration. In order to apply data mining algorithms effectively to any dataset the continuous attribute need to be transformed to discretized ones.

This approach requires integration of two approaches i.e. clustering and finding degree of dependency.

A continuous variable is one for which, within the limits the variable ranges, any value is possible. Thus, they can take any value over a defined range. Thus, in order to find degree of dependency for continuous attributes, one needs to convert them into nominal attributes and then find their degree of dependency. To convert continuous attributes to nominal variables we've used clustering (DBSCAN) technique.

## 2.3 BAYE'S THEOREM

In probability theory, **Bayes' theorem** shows the relation between one conditional probability and its inverse. The key idea is that the probability of event A (e.g., having breast cancer) given event B (having a positive mammogram) depends not only on the relationship between A and B (i.e., the accuracy of mammograms) but on the absolute probability (occurrence) of A not concerning B (i.e., the incidence of breast cancer in general), and the absolute probability of B not concerning A (i.e. the probability of a positive mammogram). For instance, if mammograms are known to be 95% accurate, this could be due to 5% false positives, 5% false negatives (missed cases), or a random mix of false positives and false negatives. Bayes' theorem allows one to calculate the exact probability of having breast cancer, given a positive mammogram for any of these three cases, because the probability of B (a positive mammogram) will be different for each of these cases. It is worth noting that if 5% of mammograms result in a positive result, then the probability that an individual with a positive result actually has cancer is rather small, since the probability of cancer is closer to 1%. The probability of a positive result is then five times more likely than the probability of the cancer itself. This shows the value of correctly understanding and applying Bayes' theorem.

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}.$$

Suppose there is a school with 60% boys and 40% girls as students. The female students wear trousers or skirts in equal numbers; the boys all wear trousers. An observer sees a (random) student from a distance; all the observer can see is that this student is wearing trousers. What is the probability this student is a girl? The correct answer can be computed using Bayes' theorem.

The event  $A$  is that the student observed is a girl, and the event  $B$  is that the student observed is wearing trousers. To compute  $P(A|B)$ , we first need to know:

- $P(A)$ , or the probability that the student is a girl regardless of any other information. Since the observer sees a random student, meaning that all students have the same probability of being observed, and the fraction of girls among the students is 40%, this probability equals 0.4.
- $P(B|A)$ , or the probability of the student wearing trousers given that the student is a girl. As they are as likely to wear skirts as trousers, this is 0.5.
- $P(B)$ , or the probability of a (randomly selected) student wearing trousers regardless of any other information. Since half of the girls and all of the boys are wearing trousers, this is  $0.5 \times 0.4 + 1 \times 0.6 = 0.8$ .

Given all this information, the probability of the observer having spotted a girl given that the observed student is wearing trousers can be computed by substituting these values in the formula:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{0.5 \times 0.4}{0.8} = 0.25.$$

Another, essentially equivalent way of obtaining the same result is as follows. Assume, for concreteness, that there are 100 students, 60 boys and 40 girls. Among these, 60 boys and 20 girls wear trousers. All together there are 80 trouser-wearers, of which 20 are girls. Therefore the chance that a random trouser-wearer is a girl equals  $20/80 = 0.25$ . Put in terms of Bayes' theorem, the probability of a student being a girl is  $40/100$ , the probability that any given girl will wear trousers is  $1/2$ . The product of these two is  $20/100$ , but we know the student is wearing trousers, so you remove the 20 non trouser wearing students and receive a probability of  $(20/100)/(80/100)$ , or  $20/80$ .

It is often helpful when calculating conditional probabilities to create a simple table containing the number of occurrences of each outcome, or the relative frequencies of each outcome, for each of the independent variables. The table below illustrates the use of this method for the above girl-or-boy example



	Girls	Boys	Total
<b>Trousers</b>	20	60	80
<b>Skirts</b>	20	0	20
<b>Total</b>	40	60	100

## **CHAPTER – 3**

### **DESIGN AND IMPLEMENTATION**

#### **3.1 DEGREE OF DEPENDENCY OF NON – CONTINUOUS ATTRIBUTES**

##### **3.1.1 MODULAR DESCRIPTION**

In the implementation part of calculating degree of dependency , we have in this module implemented for nominal attributes . Nominal variable is a variable for which values represent the names of things, with no order implied.

For example : Attributes for weather can be clear, sunny, cloudy or rainy which are nominal and that of temperature can vary along any range and these will be continuous.

Nominal variables classify data into categories. This process involves labeling categories and then counting frequencies of occurrence. The classes with maximum frequency of occurrences are then recorded and degree of dependency is calculated based on them.

## 3.2 CLUSTERING

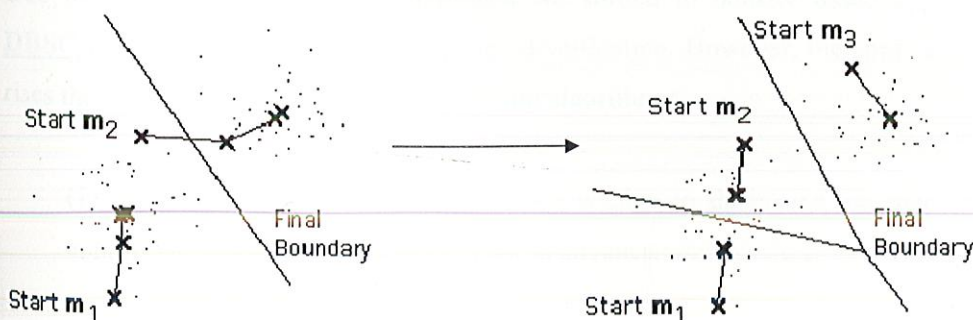
### 3.2.1 MODULAR DESCRIPTION

#### First Attempt :

We first implemented clustering by means of using partitioning algorithm i.e. k-means. K-means partitioning algorithm is a centroid-based technique. The partitioning method classifies the data into  $k$ -groups where each partition represents a cluster and  $k < n$ . That is, it classifies data into  $k$ -groups which together satisfy the following requirements:

1. Each group must contain atleast one object
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 iterative relocation technique that attempts to improve the partitioning by moving objects from one group to another. 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.



These partitioning clustering methods work well for finding spherical shaped clusters in small to medium partition sized databases.

**Weakness :**

- Applicable only when *mean* is defined
- Need to specify *k*, the *number* of clusters, in advance
- Unable to handle noisy data and *outliers*
- K mean sensitive to outliers, extremely large value may distort the data distribution
- Not suitable to discover clusters with *non-convex shapes*

**Second Attempt:**

Due the limitations of k-means approach we shifted to density based approach ,i.e. , we used **DBSCAN** algorithm in 1-D for clustering. identification. However, the application to large databases rises the following requirements for clustering algorithms:

- (1) Minimal requirements of domain knowledge to determine the input parameters, because appropriate values are often not known in advance when dealing with large databases.

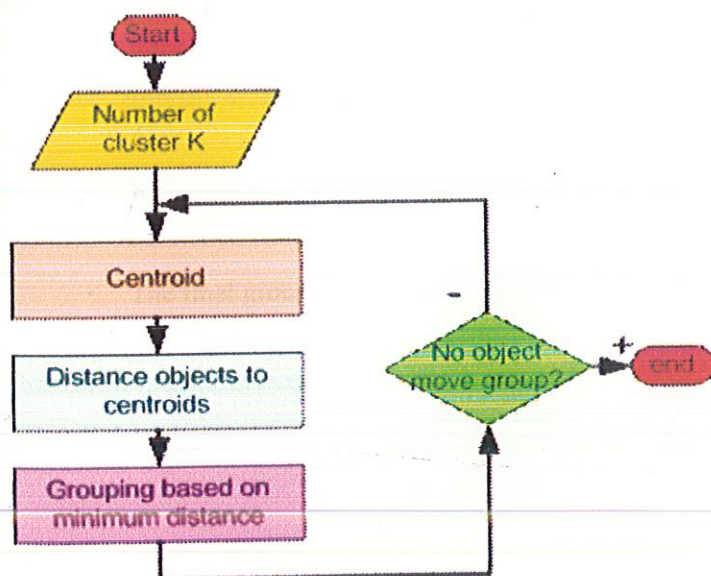
(2) Discovery of clusters with arbitrary shape, because the shape of clusters in spatial databases may be spherical, drawn-out, linear, elongated etc.

(3) Good efficiency on large databases, i.e. on databases of significantly more than just a few thousand objects.

### 3.2.2 ALGORITHM

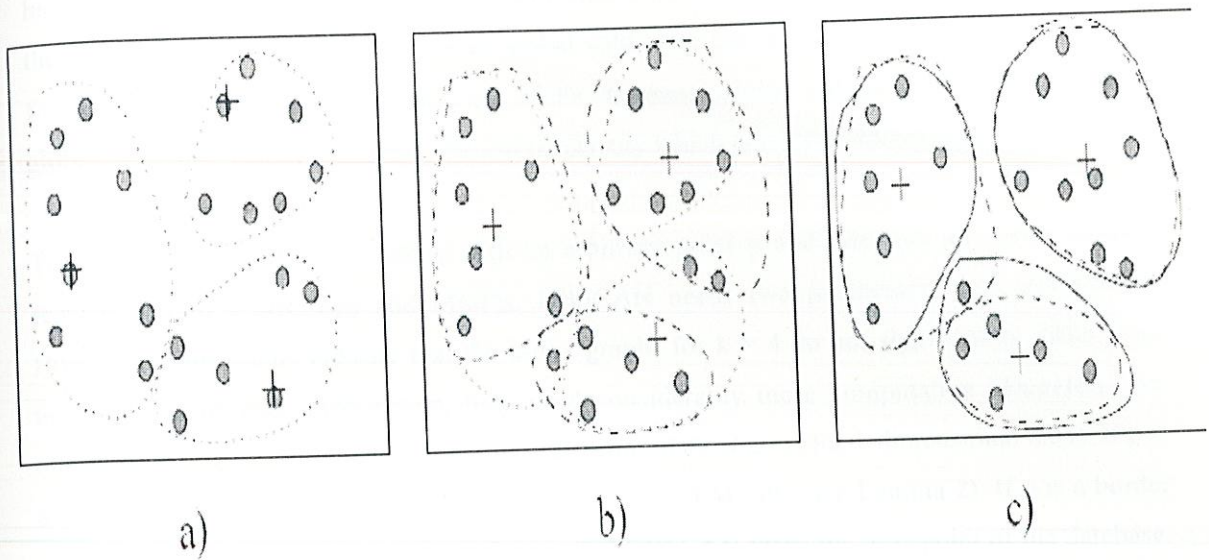
#### First Attempt:

The implementation of **k-means** required various steps :





- First we take 'n' no. of points from user
- We allocate any 'k' no. of points as centroids
- 'K' is less then or equal to 'n'
- We calculate the distance of each 'n' points from each 'k' centroids
- We save the distance data in a matrix
- We compare the distance of a point to each centroid
- And we allocate '1' if the distance is minimum, else '0' in another matrix
- We again calculate the new values of centroids considering only points with value in matrix equal to '1' in respective rows
- We repeat the process till the value of coordinates of centroid become constant
- The final groups of points are the required clusters



**Clustering of set of points using k-means method**

### **Second Attempt:**

In order to implement **DBSCAN**, the following algorithm was used:

Ideally, we would have to know the appropriate parameters  $Eps$  and  $MinPts$  of each cluster and at least one point from the respective cluster. Then, we could retrieve all points that are density-reachable from the given point using the correct parameters. But there is no easy way to get this information in advance for all clusters of the database. However, there is simple and effective

heuristic to determine the parameters Eps and MinPts of the "thinnest", i.e. least dense, cluster in the database. Therefore, DBSCAN uses global values for Eps and MinPts, i.e. the same values for all clusters. The density parameters of the "thinnest" cluster are good candidates for these global parameter values specifying the lowest density which is not considered to be noise.

To find a cluster, DBSCAN starts with an arbitrary point  $p$  and retrieves all points density-reachable from  $p$  wrt. Eps and MinPts. DBSCAN needs two parameters, Eps and MinPts. However, experiments indicate that the  $k$ -dist graphs for  $k > 4$  do not significantly differ from the 4-dist graph and, furthermore, they need considerably more computation. Therefore, we eliminate the parameter MinPts by setting it to 4 for all databases (for 1-dimensional data). If  $p$  is a core point, this procedure yields a cluster wrt. Eps and MinPts (see Lemma 2). If  $p$  is a border point, no points are density-reachable from  $p$  and DBSCAN visits the next point of the database. Since we use global values for Eps and MinPts, DBSCAN may merge two clusters according to definition 5 into one cluster, if two clusters of different density are "close" to each other. Let the distance between two sets of points  $S1$  and  $S2$  be defined as

$$\text{dist}(S1, S2) = \min \{ \text{dist}(p, q) \mid p \in S1, q \in S2 \}$$

Then, two sets of points having at least the density of the thinnest cluster will be separated from each other only if the distance between the two sets is larger than Eps. Consequently, a recursive call of DBSCAN may be necessary for the detected clusters with a higher value for MinPts. This is, however, no disadvantage because the recursive application of DBSCAN yields an elegant and very efficient basic algorithm. Furthermore, the recursive clustering of the points of a cluster is only necessary under conditions that can be easily detected.

- Arbitrary select a point  $p$
- Retrieve all points density-reachable from  $p$  wrt Eps and MinPts.

- If  $p$  is a core point(points inside of the cluster), a cluster is formed.
- If  $p$  is not a core point, no points are density-reachable from  $p$  and DBSCAN visits the next point of the database.
- Continue the process until all of the points have been processed

### 3.3 DEGREE OF DEPENDENCY FOR CONTINUOUS ATTRIBUTES

#### 3.3.1 MODULAR DESIGN

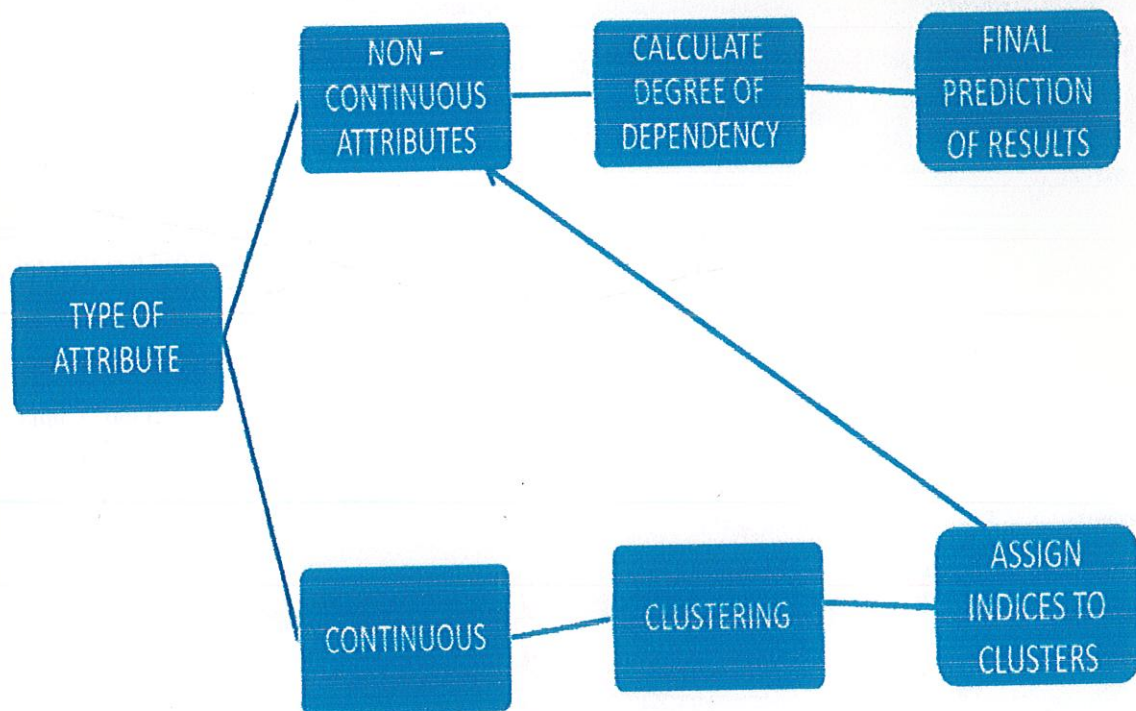
A continuous variable is one for which, within the limits the variable ranges, any value is possible. Thus, they can take any value over a defined range. Thus, in order to find degree of dependency for continuous attributes , one needs to convert them into nominal attributes and then find their degree of dependency. To convert continuous attributes to nominal variables we've used clustering.

#### 3.3.2 ALGORITHM

- Take continuous values as input
- Pass these continuous values to the clustering algorithm
- The clusters of continuous values are formed
- Create indices of these clusters
- Use these indices as non – continuous values

- Pass these indices to the algorithm for calculating the degree of dependency for non – continuous attributes
- The degree of dependency for continuous attributes takes place.

### 3.4 FLOWCHART



### 3.5 CODE

#### 3.5.1 DEGREE OF DEPENDENCY FOR NON - CONTINUOUS ATTRIBUTES

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<iostream.h>
#include<malloc.h>
#include<fstream.h>
#include<string.h>

int n=0,i=0,j=0,flag=0,t=0,dod=0;
int **fin;
char **cond;
char **deci;
char **temp;
char condition[10],decision[10];

FILE *infile3,*infile4;

void main()
{
    infile3 = fopen("dod.in","r");
    infile4 = fopen("dependency.in","r");
    fscanf(infile3,"%d",&n);
```



```

fscanf(infile3,"%s",&condition);
fscanf(infile3,"%s",&decision);


cond=(char **)malloc(n * sizeof(char *));
for(j=0;j<n;j++)
    cond[j]=(char *)malloc(10 * sizeof(char));


deci=(char **)malloc(n * sizeof(char *));
for(j=0;j<n;j++)
    deci[j]=(char *)malloc(10 * sizeof(char));


for(i=0;i<n;i++)
{
    fscanf(infile4,"%s",&cond[i]);
    fscanf(infile4,"%s",&deci[i]);
}


cout<<"\n\t\t\t\tTABLE\n";
cout<<"\t"<<condition<<"(cond. attr.)";
cout<<"\t\t"<<decision<<"(deci. attr.)";
for(i=0;i<n;i++)
{
    cout<<"\n\t\t"<<cond[i];
    cout<<"\t\t\t\t"<<deci[i];
}


temp=(char **)malloc(n * sizeof(char *));
for(j=0;j<n;j++)
    temp[j]=(char *)malloc(10 * sizeof(char));
for(i=0;i<n;i++)

```

```

        strcpy(temp[i], " ");

for(i=0; i<n; i++)
{
    for(j=0; j<n; j++)
    {
        if(strcmp(cond[i], temp[j]) == 0)
            flag = 1;
    }
    if(flag == 0)
    {
        strcpy(temp[t], cond[i]);
        t++;
    }
    flag = 0;
}

fin = (int **) malloc(n * sizeof(int *));
for(j=0; j<n; j++)
    fin[j] = (int *) malloc(2 * sizeof(int));

for(i=0; i<n; i++)
{
    for(j=0; j<2; j++)
        fin[i][j] = 0;
}

for(i=0; i<t; i++)
{
    for(j=0; j<n; j++)
    {

```

```

        if(strcmp(temp[i],cond[j])==0)
        {
            if(strcmp(deci[j],"yes")==0)
                fin[i][0]++;
            else if(strcmp(deci[j],"no")==0)
                fin[i][1]++;
        }
    }

    for(i=0;i<t;i++)
    {
        //cout<<"\n"<<dod;
        //cout<<"\n"<<fin[i][0]<<"\t"<<fin[i][1];
        if(fin[i][0]>=fin[i][1])
            dod=dod+fin[i][0];
        else if(fin[i][0]<fin[i][1])
            dod=dod+fin[i][1];
    }

    cout<<"\n\nDegree of dependency of decision attribute("<<decision<<") over conditional
    attribute("<<condition<<"):\t"<<dod<<"/"<<n;
}

```

### 3.5.2 CLUSTERING

#### k – Means:

```
#include <iostream.h>
#include <stdio.h>
#include <math.h>

void enter_data();
void distance(float z1[],float z2[],float z3[],float z4[]);
int i=0,j=0,k=0,n=0,a=0,flag=0;
float x[4]={0.0},y[4]={0.0},xy[4][4]={0.0},xy1[4][4]={0.0},temp=0.0;

void main()
{
    enter_data();
    cout<<"\nEnter the no. of centroids:\t";
    cin>>a;
    //float cenx[]={0.0},ceny[]={0.0},cenx_temp[]={0.0},ceny_temp[]={0.0};
    float *cenx,*ceny,*cenx_temp,*ceny_temp;
    cenx=new float[a];
    ceny=new float[a];
    cenx_temp=new float[a];
    ceny_temp=new float[a];
    for(i=0;i<a;i++)
    {
        cenx[i]=x[i];
        ceny[i]=y[i];
    }
}
```

```

    }
    distance(cenx,ceny,cenx_temp,ceny_temp);
}

```

```

void enter_data()

```

```

{
    i=0;
    //cout<<"Enter the no of points you want to enter:\t";
    //cin>>n;
    while(i<4)

    {
        cout<<"\nEnter the co-ordinates of the point.";
        cout<<"\nEnter the x co-ordinate:\t";
        cin>>x[i];
        cout<<"\nEnter the y co-ordinate:\t";
        cin>>y[i];
        i++;
    }
}

```

```

void distance(float cenx[],float ceny[],float cenx_temp[],float ceny_temp[])

```

```

{
    //finding the distance from centroid 1 & 2 from all 4 points
    for(i=0;i<a;i++)
        for(j=0;j<4;j++)
            xy[i][j]=sqrt(((cenx[i]-x[j])*(cenx[i]-x[j]))+((ceny[i]-y[j])*(ceny[i]-y[j]))));

    //assignment of 1 if minimum distance from resp. centroid else 0

```

```

for(j=0;j<4;j++)

{
    temp=xy[0][j];
    for(i=0;i<a;i++)

        {
            if(temp>=xy[i][j])
                temp=xy[i][j];
        }

    for(i=0;i<a;i++)

        {
            if(xy[i][j]==temp)
                xyl[i][j]=1;
            else
                xyl[i][j]=0;
        }
}

//for new centroid
for(i=0;i<a;i++)

{
    cenx_temp[i]=cenx[i];
    ceny_temp[i]=ceny[i];
    cenx[i]=0;
    ceny[i]=0;
}

```



```

for(i=0;i<a;i++)

{
    k=0;
    for(j=0;j<4;j++)

    {
        if(xy1[i][j]==1)

        {
            k++;
            cenx[i]=cenx[i]+x[j];
            ceny[i]=ceny[i]+y[j];
        }
    }

    if(k!=0)

    {
        cenx[i]=cenx[i]/k;
        ceny[i]=ceny[i]/k;
    }
}

//checking if centroids coordinates have changed or not
for(i=0;i<a;i++)

{
    if((cenx_temp[i]==cenx[i])&&(ceny_temp[i]==ceny[i]))
        flag=1;
}

```

```

        else
            flag=0;
    }

    if(flag==1)
    {
        for(i=0;i<a;i++)
        {
            cout<<"\n\nThe      coordinate      of      centroid      no."<<i+1<<"
are:\t{"<<cenx[i]<<","<<ceny[i]<<"}";
            cout<<"\nThe points in cluster no."<<i+1<<" is/are:\t";
            for(int j=0;j<4;j++)
            {
                if(xy1[i][j]==1)
                    cout<<{"<<x[j]<<","<<y[j]<<"}\t";
            }
        }
    }

    Else

        distance(cenx,centy,cenx_temp,centy_temp);

}

```

## DBSCAN:

```
#include<stdio.h>
#include<stdlib.h>
#include<iostream.h>
#include<malloc.h>

void main()
{
    int n=0,i=0,j=0,k=0,low=0,high=0,minpts=0,p=0;

    float eps=0.0;

    cout<<"\nPlease enter the value of eps(float value):\t";
    cin>>eps;
    cout<<"\nPlease enter the value of minpoints(integer value):\t";
    cin>>minpts;
    cout<<"\nPlease enter the number of data points in the dataset:\t";
    cin>>n;

    float *arr;
    arr=new float[n];
    for(i=0;i<n;i++)

    {
        cout<<"\nEnter the data for "<<i+1<<" location:\t";
        cin>>arr[i];
    }

    for(i=0;i<n;i++)
```

```

{
    if(high<arr[i])
        high=arr[i];

    if(low>arr[i])
        low=arr[i];
}

p=((high-low)/eps)+1;
int *count;
count=new int[p];

float **clus;
clus=(float **)malloc(p * sizeof(float *));
for(i=0;i<p;i++)
    clus[i]=(float *)malloc(p * sizeof(float));

for(i=0;i<p;i++)

{
    count[i]=0;
    for(j=0;j<p;j++)
        clus[i][j]=0.0;
}

for(i=0;i<n;i++)

{
    for(j=0;j<p;j++)

```

```

{
    if((arr[i]>=(low+(j*eps)))&&(arr[i]<=(low+((j+1)*eps))))

    {
        clus[j][count[j]]=arr[i];
        count[j]++;
    }
}

for(i=0;i<p;i++)

    cout<<"\nThe no of elements in partition "<<i+1<<" is/are:\t"<<count[i];

j=1;

for(i=0;i<p;i++)

{
    if(count[i]>=minpts)
    {
        cout<<"\n\nNumber of elements in cluster "<<j<<" are:\t"<<count[i];
        cout<<"\nWhich are:\t";
        for(k=0;k<count[i];k++)

            cout<<clus[i][k]<<"\t";

        j++;
    }
}

cout<<"\n\n\nNumber of clusters formed according to eps & minpoints values are:\t"<<j-1<<"\n\n";

```

```
cout<<"\n\nNumber of elements part of Noise is/are:\t";
```

```
for(i=0;i<p;i++)
```

```
{
```

```
    if((count[i]<minpts)&&(count[i]>0))
```

```
    {
```

```
        for(k=0;k<count[i];k++)
```

```
            cout<<clus[i][k]<<"\t";
```

```
    }
```

```
}
```



### 3.5.3 DEGREE OF DEPENDENCY FOR CONTINUOUS ATTRIBUTES

```
FILE *infile1,*infile2;
infile1 = fopen("userdatadod.in","r");
infile2 = fopen("datadod.in","r");

fscanf(infile1,"%f",&eps);
fscanf(infile1,"%d",&minpts);
fscanf(infile1,"%d",&maxpts);
fscanf(infile1,"%d",&maxlth);
fscanf(infile1,"%d",&n);
fscanf(infile1,"%s",&condition);
fscanf(infile1,"%s",&decision);

eps=eps*2;
float *arr;
arr=new float[n];

char *output;
output=new char[n];

for(i=0;i<n;i++)
{
    fscanf(infile2,"%f",&arr[i]);
    fscanf(infile2,"%c",&output[i]);
}

for(i=0;i<n;i++)
{
    if(high<arr[i])
        high=arr[i];
}
```

```

        if(low>arr[i])
            low=arr[i];
    }

    p=((high-low)/eps)+1;

    int *count;
    count=new int[p];

    float **clus;
    clus=(float **)malloc(p * sizeof(float *));
    for(i=0;i<p;i++)
        clus[i]=(float *)malloc(p * sizeof(float));

    for(i=0;i<p;i++)
    {
        count[i]=0;
        for(j=0;j<p;j++)
            clus[i][j]=0.0;
    }

    for(i=0;i<n;i++)
    {
        for(j=0;j<p;j++)
        {
            if((arr[i]>=(low+(j*eps)))&&(arr[i]<=(low+((j+1)*eps))))
            {
                clus[j][count[j]]=arr[i];
                count[j]++;
            }
        }
    }

```

```

    }
}

float **inter;
inter=(float **)malloc((p+10) * sizeof(float *));
for(i=0;i<(p+10);i++)
inter[i]=(float *)malloc((p+10) * sizeof(float));

for(i=0;i<(p+10);i++)
{
    for(j=0;j<(p+10);j++)
        inter[i][j]=0.0;
}

int t0=0;
l=0;

for(i=0;i<p;i++)
{
    if(count[i]>=minpts)
    {
        for(k=0;k<count[i];k++)
        {
            inter[l][k]=clus[i][k];
        }
        if(count[i]>t0) //to find max no of elements
            t0=count[i];
        l++; //l moving as i
    }
}

```

```

j=1;
for(i=0;i<p;i++)
{
    if(count[i]>=minpts)
    {
        //cout<<"\n\nNumber of elements in cluster "<<j<<" are:\t"<<count[i];
        //cout<<"\nWhich are:\t";
        //for(k=0;k<count[i];k++)
        //cout<<clus[i][k]<<"\t";
        j++;
    }
}

interval=j-1;
cout<<"No. of intervals are:\t"<<interval;
cout<<"\n";

int no_of_elements=0;
float temp_low=0;
float temp_high=0;
float range=0;
int no_of_elements_1=0;
int no_of_elements_2=0;
int g;
float temp_low1=0;
float temp_high1=0;
float temp_low2=0;
float temp_high2=0;
float max=0;
float min=0;

```

```

float h;
float d1;
float d2;
float temp_min=0;
int t1=0;

float *d;
d=new float[interval];
for(j=0;j<interval;j++)
    d[j]=10000;
//large value

float *in;
in=new float[t0];

float *in2;
in2=new float[t0];

for(i=0;i<t0;i++)
{
    in[i]=0.0;
    in2[i]=0.0;
}

int temp_interval=interval;
do
{
    temp_interval=interval;
    for(i=0;i<interval;i++)
    {
        for(j=0;j<t0;j++)

```

```

        in[j]=inter[i][j];

no_of_elements=0;
for(j=0;j<t0;j++)
{
    if(in[j]!=0.0)
        no_of_elements++;
}

temp_low=in[0];
temp_high=in[0];
for(j=0;j<t0;j++)
{
    if(temp_low>in[j])
        temp_low=in[j];
    if(temp_high<in[j])
        temp_high=in[j];
}
range=temp_high-temp_low;

if((no_of_elements>=(2*minpts))||(range>maxlth))    //split condition
{
    int mid=no_of_elements/2;

    if((no_of_elements%2)==0)    //even
    {
        for(j=0;j<mid;j++)
            inter[interval][j]=inter[i][mid+j];

        for(j=mid;j<t0;j++)
            inter[i][j]=0.0;
    }
}

```



```

    }

    else //odd
    {
        for(j=0;j<mid;j++)
            inter[interal][j]=inter[i][mid+j+1];

        for(j=mid;j<t0;j++)
            inter[i][j+1]=0.0;
    }

    interal++;
}

else if(no_of_elements<minpts)
{
    for(k=0;k<interval;k++)
    {
        if(k!=i)
        {
            for(j=0;j<t0;j++)
                in2[j]=inter[k][j];

            no_of_elements_1=0;
            no_of_elements_2=0;
            for(j=0;j<t0;j++)
            {
                if(in[j]!=0)
                    no_of_elements_1++;
                if(inter[k][j]!=0)
                    no_of_elements_2++;
            }
        }
    }
}

```

```
}
```

```
temp_low1=0;
```

```
temp_high1=0;
```

```
temp_low2=0;
```

```
temp_high2=0;
```

```
max=0;
```

```
min=0;
```

```
for(j=0;j<no_of_elements_1;j++)
```

```
{
```

```
    if(temp_low1>in[j])
```

```
        temp_low1=in[j];
```

```
    if(temp_high1<in[j])
```

```
        temp_high1=in[j];
```

```
}
```

```
for(j=0;j<no_of_elements_2;j++)
```

```
{
```

```
    if(temp_low2>inter[k][j])
```

```
        temp_low2=inter[k][j];
```

```
    if(temp_high2<inter[k][j])
```

```
        temp_high2=inter[k][j];
```

```
}
```

```
d1=temp_high1-temp_low2;
```

```
d2=temp_low1-temp_high2;
```

```
if(d1<0)
```

```
    d1=d1*(-1);
```

```
if(d2<0)
```

```

        d2=d2*(-1);

        if(d1<d2)
//smaller distance
            d[k]=d1;
        else
            d[k]=d2;
    }                //end if
}                //end for

if(i!=0)
    temp_min=d[0];
else
    temp_min=d[1];

for(k=0;k<interval;k++)    //to find minimum distance pair
{
    if(k!=i)
    {
        if(temp_min>d[k])
        {
            temp_min=d[k];
            t1=k;
        }
    }
}

no_of_elements_1=0;
no_of_elements_2=0;
for(j=0;j<t0;j++)
{

```

```

        if(in[j]!=0)
            no_of_elements_1++;
        if(inter[t1][j]!=0)
            no_of_elements_2++;
    }

    g=no_of_elements_1+no_of_elements_2;

    temp_low1=0;
    temp_high1=0;
    temp_low2=0;
    temp_high2=0;
    max=0;
    min=0;
    for(j=0;j<no_of_elements_1;j++)
    {
        if(temp_low1>in[j])
            temp_low1=in[j];

        if(temp_high1<in[j])
            temp_high1=in[j];
    }

    for(j=0;j<no_of_elements_2;j++)
    {
        if(temp_low2>inter[t1][j])
            temp_low2=inter[t1][j];

        if(temp_high2<inter[t1][j])
            temp_high2=inter[t1][j];
    }

```

```
if(temp_high1>temp_high2)
```

```
    max=temp_high1;
```

```
else
```

```
    max=temp_high2;
```

```
if(temp_low1<temp_low2)
```

```
    min=temp_low1;
```

```
else
```

```
    min=temp_low2;
```

```
h=max-min;
```

```
int t2=0;
```

```
if((g<minpts)&&(h<maxlth))
```

```
//merge
```

condition

```
{
```

```
    for(j=no_of_elements;j<t0;j++)
```

```
    {
```

```
        inter[i][j]=inter[t1][t2];
```

```
        t2++;
```

```
    }
```

```
    for(k=t1;k<interval-1;k++)
```

```
    {
```

```
        for(j=0;j<t0;j++)
```

```
        {
```

```
            inter[k][j]=inter[k+1][j];
```

```
        }
```

```
    }
```

```

        interval--;
    }
}

//end else if
} //end for
}

while(interval!=temp_interval); //no change in no. of intervals

for(i=0;i<interval;i++)
{
    for(k=0;k<t0;k++)
        cout<<inter[i][k]<<"\t";
    cout<<"\n";
}

cout<<"\n\t\t\tTABLE\n";
cout<<"\t"<<condition<<"(cond. attr.)";
cout<<"\t\t\t"<<decision<<"(deci. attr.)";
for(i=0;i<n;i++)
{
    cout<<"\n\t\t"<<arr[i];
    cout<<"\t\t\t"<<output[i];
}

fin=(int **)malloc(n * sizeof(int *));
for(j=0;j<n;j++)
    fin[j]=(int *)malloc(2 * sizeof(int));

for(i=0;i<n;i++)
{
    for(j=0;j<2;j++)

```

```

        fin[i][j]=0;
    }

    int x=0;
    for(i=0;i<interval;i++)
    {
        for(j=0;j<t0;j++)
        {
            for(k=0;k<n;k++)
            {
                if(inter[i][j]==arr[k])
                {
                    if(output[k]=='y')
                        fin[i][0]++;
                    else if(output[k]=='n')
                        fin[i][1]++;
                    x++;
                }
            }
        }
    }

    for(i=0;i<interval;i++)
    {
        if(fin[i][0]>=fin[i][1])
            dod=dod+fin[i][0];
        else if(fin[i][0]<fin[i][1])
            dod=dod+fin[i][1];
    }

```

```

cout<<"\n\nDegree of dependency of decision attribute("<<decision<<")\n"

```



```
<<"over conditional attribute("<<condition<<").\t"<<dod<<"/"<<x;  
return 0;  
}
```

## CHAPTER – 4

### RESULTS

#### 4.1 DEGREE OF DEPENDENCY OF NON-CONTINUOUS ATTRIBUTES

(Inactive C:\DOCUMENTS\HUSH\DESKTOP\DOO\_11213)

Enter the decision data for 2 location:           yes

Enter the conditional data for 3 location:       rainy

Enter the decision data for 3 location:           no

Enter the conditional data for 4 location:       rainy

Enter the decision data for 4 location:           yes

Enter the conditional data for 5 location:       sunny

Enter the decision data for 5 location:           yes

TABLE	
outlook(cond. attr.)	play(deci. attr.)
over	yes
over	yes
rainy	no
rainy	yes
sunny	yes

Degree of dependency of decision attribute(play) over conditional attribute(outlook): 4/5

## 4.2 CLUSTERING

### k - Means:

■ (Inactive Z:\FINALY-1\K\_COD-1\K-MEAN\_K.EXE)

Enter the y co-ordinate: 1

Enter the co-ordinates of the point.  
Enter the x co-ordinate: 1

Enter the y co-ordinate: 2

Enter the co-ordinates of the point.  
Enter the x co-ordinate: 3

Enter the y co-ordinate: 4

Enter the co-ordinates of the point.  
Enter the x co-ordinate: 5

Enter the y co-ordinate: 4

Enter the no. of centroids: 2

The coordinate of centroid no.1 are: {1,1.5}  
The points in cluster no.1 is/are: {1,1} {1,2}

The coordinate of centroid no.2 are: {4,4}  
The points in cluster no.2 is/are: {3,4} {5,4}

## DBSCAN:

(Inactive Z:\FINALY-1\K\_COD-1\DBSCAN.EXE)

Enter the data for 16 location: 99

Enter the data for 17 location: 109

Enter the data for 18 location: 49

Enter the data for 19 location: 185

Enter the data for 20 location: 144

Number of elements in cluster 1 are: 5  
Which are: 1 5 7.8 8.4 3

Number of elements in cluster 2 are: 4  
Which are: 82 83 85.5 88

Number of clusters formed according to eps & minpoints values are: 2

Number of elements part of Noise is/are: 15 26 47 49.9  
49 63 68 99 109 144 185

### 4.3 DEGREE OF DEPENDENCY OF CONTINUOUS ATTRIBUTES

☐ (Inactive Z:\FINALE~1\EPSPDOD.EXE)



No. of intervals are: 2

Which are: 1.3 5.2 9.4

Which are: 55.7 58.8 52

TABLE

Humidity(cond. attr.)

Outcome(deci. attr.)

1.3

n

5.2

y

9.4

n

99.6

n

55.7

y

58.8

n

52

n

22

y

84

n

41

y

Degree of dependency of decision attribute(Outcome)  
over conditional attribute(Humidity): 4/6

#### **4.4 TECHNOLOGY AND RESOURCES USED**

Hardware :

- 1(+)GB RAM

Software :

- Compiler – Turbo C++4.5

Operating System :

- Windows Operating System

## CHAPTER – 5

### FUTURE WORK AND CONCLUSION

The prediction tool we've created is for 1-D applications. Due to its flexibility, high – scalability, and less dependence on domain knowledge, it can be used for prediction in various domains. For example: stock analysis, weather prediction and many such kind of domains. This project can be further extended to 2-D and 3-D applications like image segmentation, object recognition etc.. DBSCAN, the clustering technique we've used, is a model basically constructed for spatial data and hence, permits such extension.

There are several applications where decision making and exploratory pattern analysis have to be performed on large data sets. For example, in document retrieval, a set of relevant documents has to be found among several millions of documents of dimensionality of more than 1000. It is possible to handle these problems if some useful abstraction of the data is obtained and is used in decision making, rather than directly using the entire data set. By *data abstraction*, we can have a simple and compact representation of the data. This simplicity helps the machine in efficient processing or a human in comprehending the structure in data easily. Clustering algorithms are ideally suited for achieving data abstraction.

“The Prediction Tool Based on Clustering” is a tool designed to predict data in 1 – D, considering some limitations of data mining techniques and thus, merging them in a order which helps in overcoming the constraints. The following constraints have been taken care of:

- Scalability Issues
- Domain specificity
- Difficulties in Non – Convex shaped cluster formation

In summary, clustering is an interesting, useful, and challenging problem. It has great potential in applications like object recognition, image segmentation, and information filtering and retrieval. However, it is possible to exploit this potential only after making several design choices carefully. Such an attempt was made by us.



## BIBLIOGRAPHY:

- [1.] Decision making via degree of dependency (A.M. Kozae a. A.A. Abo Khadra b, T. Medhat b).
- [2.] Discretization Using Clustering and Rough Set Theory (Girish Kumar Singh, Sonajharia Minz)
- [3.] Truth from Trash : How learning Makes Sense, Chris Thornton, MIT Press, 2000.
- [4.] <http://people.revoledu.com/kardi/tutorial/kMean/index.html>
- [5.] A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise (By: Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu)
- [6.] Data Clustering: A Review (By: A.K. Jain, M.N. Murty & P.J. Flynn)
- [7.] <http://en.wikipedia.org/wiki/>
- [8.] <http://www.cs.ualberta.ca/~zaiane/courses/cmput690/slides/Chapter8/sld023.htm>
- [9.] <http://www.inf.unibz.it/dis/teaching/misc/project-dbscan.html>
- [10.] [http://home.dei.polimi.it/matteucc/Clustering/tutorial\\_html/](http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/)
- [11.] <http://www.statsoft.com/textbook/cluster-analysis/>
- [12.] Data Mining: Concepts and Techniques (By: Jiawei Han and Micheline Kamber)