DATA MINING ON HEALTH DATA

Submitted in partial fulfilment
of the requirements for the degree of

BACHELOR OF TECHNOLOGY
(CSE & / IT)
By

Tejpal Singh Sahni 071246

Amber pandey 071247

Under the Supervision of
Mr. Pardeep Kumar

MAY 2011

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY-WAKNAGHAT

CONTENTS

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

CERTIFICATE

This is to certify that the work entitled "Data Mining On Health Data" submitted in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science 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.

Name of Supervisor: PARDEEP KUMAR (Pardeep Kumar. 23/5/2011

ACKNOWLEDGEMENT

Computer Science is fast developing subject & new dimensions are being added from time to time. During the last decade many new dimensions have been introduced and many old once have been redefined. In light of new development and recent findings, we devote the task that was asked from us during our IV year at Jaypee University Of Information Technology to DATA MINING ON HEALTH DATA .

We would like to thank **Mr. Pardeep Kumar** who helped us during the task given to us. We thank him again for providing us with the necessary facilities to complete this task.

Tejpal Singh Sahni	Roll No. 071246
Amber Pandey	Roll No. 071247

B.Tech(Computer Science & Engineering)

Jaypee University Of Information Technology

CHAPTER 1: INTRODUCTION AND PROBLEM STATEMENT

1.1 INTRODUCTION

A primary motivation for Data mining (DM) is that a lot of data is inherently distributed .Merging of remote data at the central site to perform data mining will result in unnecessary communication overhead and algorithmic complexities .

For example, consider the NASA Earth Observing System Data and Information System (EOSDIS) which manages data from earth science research satellites and field measurement programs. It provides data archiving, distribution, and information management services and holds more than 1450 datasets that are stored and managed at many sites throughout the United States. It manages extraordinary rates and volumes of scientific data.

For example, Terra spacecraft produces 194 gigabytes (GB) per day; data downlink is at 150 Megabits/sec and the average amount of data collected per orbit is 18.36Megabits/sec14. A centralized data mining system may not be adequate in such a dynamic, distributed environment.

Indeed, the resources required to transfer and merge the data on a centralized site may become implausible at such a rapid rate of data arrival. Data mining techniques that minimize communication between sites are quite valuable.

DDM is data mining where the data and computation are spread over many independent sites. For some applications, the distributed setting is more natural than the centralized one because the data is inherently distributed. Typically, in a DDM environment, each site has its own data source and data mining algorithms operate on it producing local models .Each local model represents knowledge learned from the local data source, but could lack globally meaningful knowledge. Thus the sites need to communicate by message passing over a network, in order to keep track of the global information.

1.2 MOTIVATION

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 type 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

(ii) natural groupings (clustering) revealed through analysis.

Cluster analysis is the organization of a collection of patterns (usually represented as a vector of measurements, or a point in a multidimensional space) 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. An example of clustering is depicted in Figure 1. The input patterns are shown in Figure 1(a), and the desired clusters are shown in Figure 1(b). Here, points belonging to the same cluster are given the same label. The variety of techniques for representing data, measuring proximity (similarity) between data elements, and grouping data elements has produced a rich and often confusing assortment of clustering methods.

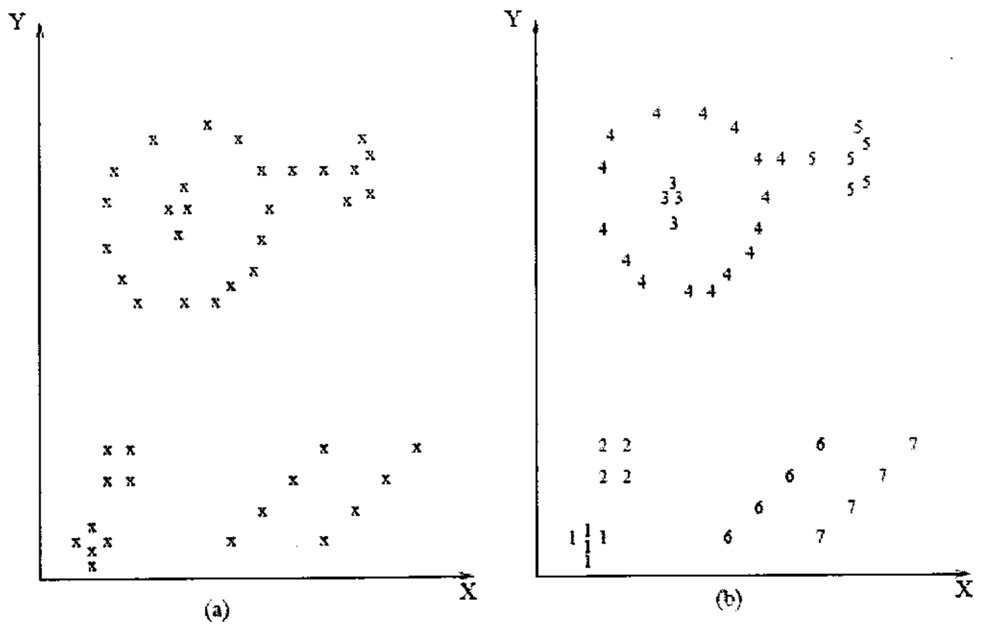


Figure 1. Data clustering.

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. 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.

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.

1.3 STATEMENT OF PROBLEM

To design a system that can help the victims of various health related data to find solution of their problem. We aim to design a system that can locate them to specialized health centre for their problem as well as can provide them measures to keep them secure and health our system will also warn them various associated diseases which can be their future problem due to present disease.

1) If a person enter his disease symptoms we can give him the probable disease from which he is suffering.
2) We can locate the spreading disease in certain location by doing clustering on attributes like location id and disease id which we collect from the health centers.

CHAPTER 2: BACKGROUND

2.1 : HISTORY

Data mining refers to extracting or mining knowledge from large amounts of data. Remember that the mining of gold from rocks or sand is referred to as gold mining rather than rock or sand mining. Thus, data mining should have been more appropriately named knowledge mining from data, which is unfortunately somewhat long. Knowledge mining, a shorter term, may not correct the emphasis on mining from large amounts of data. Nevertheless, mining is a vivid term , the characterizing the process that finds a small set of precious nuggets from a great deal of raw material (Figure). Thus, such a misnomer which carries both data and mining became a popular choice. There are many other terms carrying a similar or slightly different meaning to data mining, such as knowledge mining from databases, knowledge extraction, data/pattern analysis, data archaeology, and data dredging.

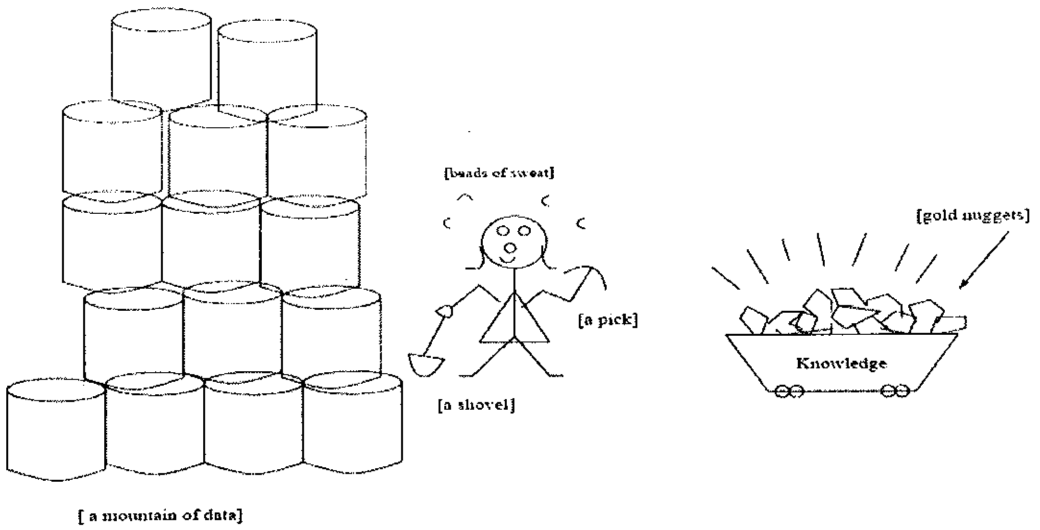


Figure : Data mining searching for knowledge (interesting patterns) in your data.

Many people treat data mining as a synonym for another popularly used term, Knowledge Discovery in Databases", or KDD. Alternatively, others view data mining as simply an essential step in the process of knowledge discovery in databases. Knowledge discovery as a process is depicted in Figure 1.2, and consists of an iterative sequence of the following steps:

- Data cleaning (to remove noise or irrelevant data),
- Data integration (where multiple data sources may be combined),
- Data selection (where data relevant to the analysis task are retrieved from the database),
- Data transformation (where data are transformed or consolidated into forms appropriate for mining by performing summary or aggregation operations, for instance),
- Data mining (an essential process where intelligent methods are applied in order to extract data patterns),
- Pattern evaluation (to identify the truly interesting patterns representing knowledge based on some interestingness measures), and
- Knowledge presentation (where visualization and knowledge representation techniques are used to present the mined knowledge to the user).

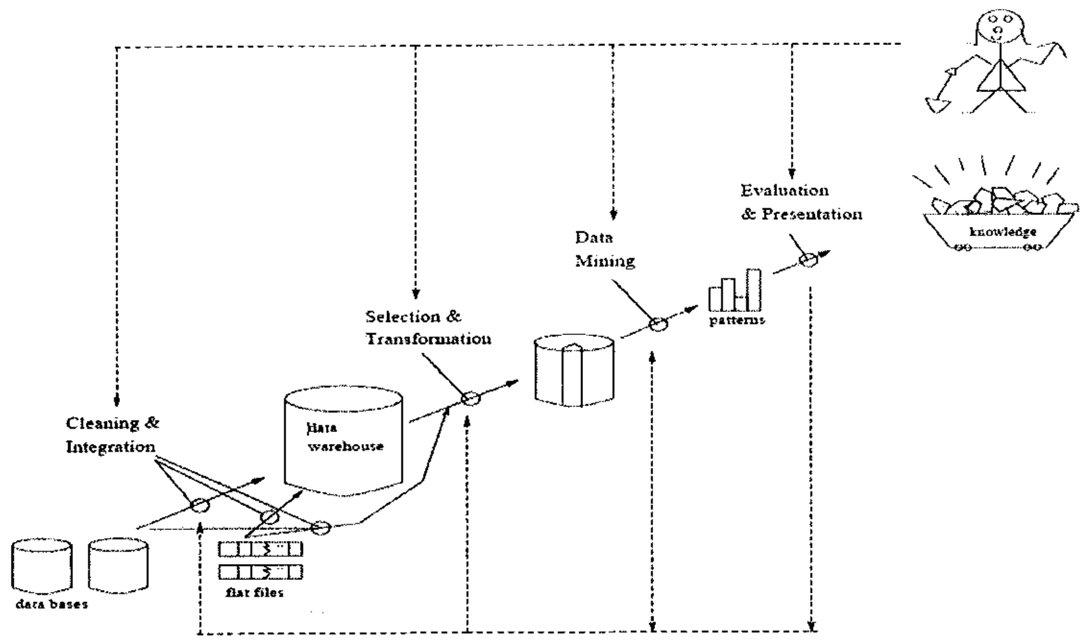


Figure 1 : Data mining as a process of knowledge discovery.

2.2 ARCHITECTURE OF TYPICAL DATA MINING SYSTEM

The architecture of a typical data mining system may have the following major components as shown in figure below:

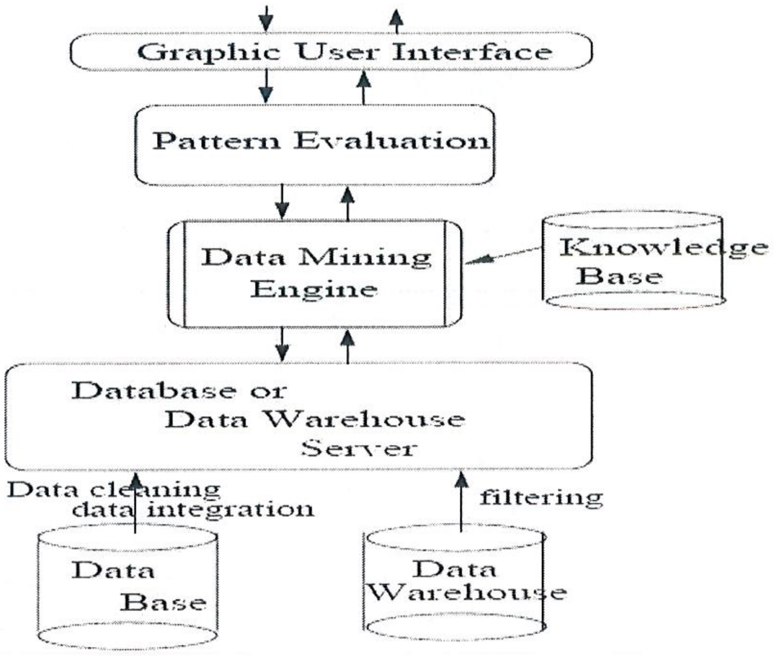


Figure : Architecture of a typical data mining system.

1. Database, data warehouse, or other information repository. This is one or a set of databases, data warehouses, spread sheets, or other kinds of information repositories. Data cleaning and data integration techniques may be performed on the data.

2. Database or data warehouse server. The database or data warehouse server is responsible for fetching the relevant data, based on the user's data mining request.

3. Knowledge base. This is the domain knowledge that is used to guide the search, or evaluate the interestingness of resulting patterns. Such knowledge can include concept hierarchies, used to organize attributes or attribute values into different levels of abstraction. Knowledge such as user beliefs, which can be used to assess a pattern's interestingness based on its unexpectedness, may

also be included. Other examples of domain knowledge are additional interestingness constraints or thresholds, and metadata (e.g., describing data from multiple heterogeneous sources).

4. Data mining engine. This is essential to the data mining system and ideally consists of a set of functional modules for tasks such as characterization, association analysis, classi_cation, evolution and deviation analysis.

5. Pattern evaluation module. This component typically employs interestingness measures and interacts with the data mining modules so as to focus the search towards interesting patterns. It may access interestingness thresholds stored in the knowledge base. Alternatively, the pattern evaluation module may be integrated with the mining module, depending on the implementation of the data mining method used. For efficient data mining, it is highly recommended to push the evaluation of pattern interestingness as deep as possible into the mining process so as to confine the search to only the interesting patterns.

6. Graphical user interface. This module communicates between users and the data mining system, allowing the user to interact with the system by specifying a data mining query or task, providing information to help focus the search, and performing exploratory data mining based on the intermediate data mining results. In addition, this component allows the user to browse database and data warehouse schemas or data structures, evaluate mined patterns, and visualize the patterns in different forms.

From a data warehouse perspective, data mining can be viewed as an advanced stage of on-line analytical processing (OLAP). However, data mining goes far beyond the narrow scope of summarization-style analytical processing of data warehouse systems by incorporating more advanced techniques for data understanding.

While there may be many \data mining systems" on the market, not all of them can perform true data mining. A data analysis system that does not handle large amounts of data can at most be categorized as a machine learning system, a statistical data analysis tool, or an experimental system prototype. A system that can only perform data or information retrieval, including finding aggregate values, or that performs deductive query answering in large databases should be more appropriately categorized as either a database system, an information retrieval system, or a deductive database system.

Data mining involves an integration of techniques from multiple disciplines such as database technology, statistics, machine learning, high performance computing, pattern recognition, neural networks, data visualization, information retrieval, image and signal processing, and spatial data analysis. We adopt a database perspective in our presentation of data mining in this book. That is, emphasis is placed on efficient and scalable data mining techniques for large databases. By performing data mining, interesting knowledge, regularities, or high-level information can be extracted from databases and viewed or browsed from different angles. The discovered knowledge can be applied to decision making, process control, information management, query processing, and so on. Therefore, data mining is considered as one of the most important frontiers in database systems and one of the most promising, new database applications in the information industry.

2.3 DATA WAREHOUSE

A data warehouse is a repository of information collected from multiple sources, stored under a united schema, and which usually resides at a single site. Data warehouses are constructed via a process of data cleansing, data transformation, data integration, data loading, and periodic data refreshing.

In order to facilitate decision making, the data in a data warehouse are organized around major subjects, such as customer, item, supplier, and activity. The data are stored to provide information from a historical perspective (such as from the past 5-10 years), and are typically summarized. For example, rather than storing the details of each sales transaction, the data warehouse may store a summary of the transactions per item type for each store, or, summarized to a higher level, for each sales region.

A data warehouse is usually modeled by a multidimensional database structure, where each dimension corresponds to an attribute or a set of attributes in the schema, and each cell stores the value of some aggregate measure, such as count or sales amount. The actual physical structure of a data warehouse may be a relational data store or a multidimensional data cube.

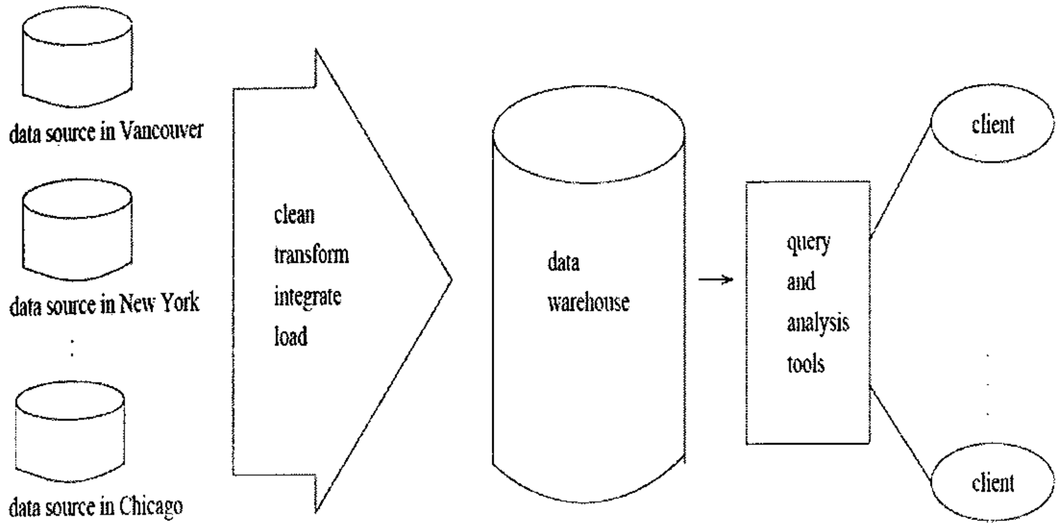


Figure : Architecture of a typical data warehouse.

2.4 CLASSIFICATION AND PREDICTION

Classification is the processing of finding a set of models (or functions) which describe and distinguish data classes or concepts, for the purposes of being able to use the model to predict the class of objects whose class label is unknown. The derived model is based on the analysis of a set of training data (i.e., data objects whose class label is known).

The derived model may be represented in various forms, such as classification (IF-THEN) rules, decision trees, mathematical formulae, or neural networks. A decision tree is a flow-chart-like tree structure, where each node denotes a test on an attribute value, each branch represents an outcome of the test, and tree leaves represent classes or class distributions. Decision trees can be easily converted to classification rules. A neural network is a collection of linear threshold units that can be trained to distinguish objects of different classes.

Classification can be used for predicting the class label of data objects. However, in many applications, one may like to predict some missing or unavailable data values rather than class labels. This is usually the case when the predicted values are numerical data, and is often specifically referred to as prediction. Although prediction may refer to both data value prediction and class label prediction, it is usually confined to data value prediction and thus is distinct from classification. Prediction also encompasses the identification of distribution trends based on the available data.

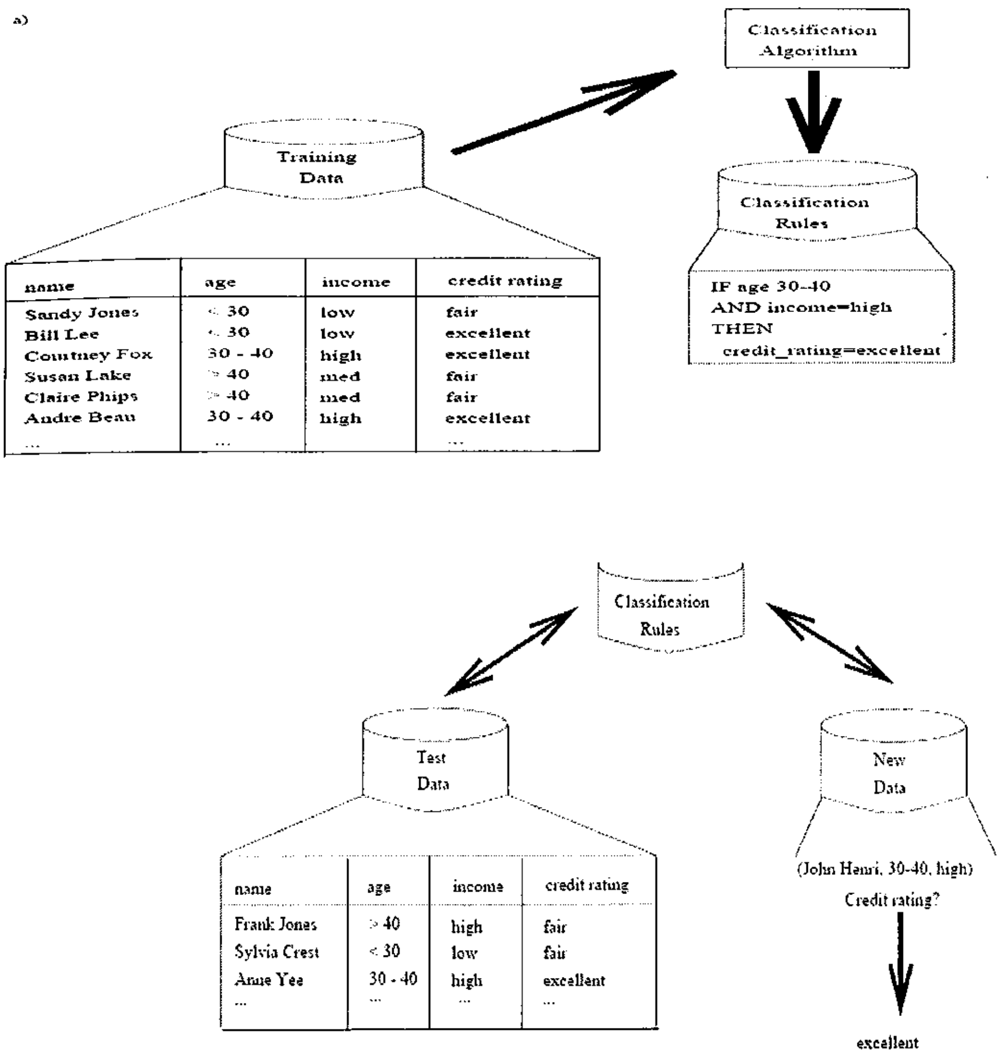
CHAPTER 3: DATA MINING ALGORITHMS

3.1 CLASSIFICATION AND PREDICTION

Data classification is a two step process (Figure). In the first step, a model is built describing a predetermined set of data classes or concepts. The model is constructed by analyzing database tuples described by attributes. Each tuple is assumed to belong to a predefined class, as determined by one of the attributes, called the class label attribute. In the context of classification, data tuples are also referred to as samples, examples, or objects. The data tuples analyzed to build the model collectively form the training data set. The individual tuples making up the training set are referred to as training samples and are randomly selected from the sample population. Since the class label of each training sample is provided, this step is also known as supervised learning (i.e., the learning of the model is 'supervised' in that it is told to which class each training sample belongs). It contrasts with unsupervised learning (or clustering), in which the class labels of the training samples are not known, and the number or set of classes to be learned may not be known in advance.

Typically, the learned model is represented in the form of classification rules, decision trees, or mathematical formulae. For example, given a database of customer credit information, classification rules can be learned to identify customers as having either excellent or fair credit ratings (Figure a). The rules can be used to categorize future data samples, as well as provide a better understanding of the database contents.

In the second step (Figure b), the model is used for classification. First, the predictive accuracy of the model(or classifier) is estimated. The holdout method is a simple technique which uses a test set of class-labeled samples.



name	age	income	credit rating
Sandy Jones	< 30	low	fair
Bill Lee	< 30	low	excellent
Coutney Fox	30 - 40	high	excellent
Susan Lake	> 40	med	fair
Claire Phips	> 40	med	fair
Andre Beau	30 - 40	high	excellent
...	...		...

IF age 30-40
AND income=high
THEN
credit_rating=excellent

name	age	income	credit rating
Frank Jones	> 40	high	fair
Sylvia Crest	< 30	low	fair
Anne Yee	30 - 40	high	excellent
...	...	...	...

Figure 7.1: The data classification process: a) *Learning:* Training data are analyzed by a classification algorithm. Here, the class label attribute is *credit_rating*, and the learned model or classifier is represented in the form of classification rules. b) *Classification:* Test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

The accuracy of a model on a given test set is the percentage of test set samples that are correctly classified by the model. For each test sample, the known class label is compared with the learned model's class prediction for that sample. Note that if the accuracy of the model were estimated based on the training data set, this estimate could be optimistic since the learned model tends to overt the data (that is, it may have incorporated some particular anomalies of the training data which are not present in the overall sample population). Therefore, a test set is used.

If the accuracy of the model is considered acceptable, the model can be used to classify future data tuples or no objects for which the class label is not known. (Such data are also referred to in the machine learning literature as \unknown" or \previously unseen" data). For example, the classification rules learned in Figure 7.1a from the analysis of data from existing customers can be used to predict the credit rating of new or future (i.e., previously unseen) customers.

Prediction can be viewed as the construction and use of a model to assess the class of an unlabeled object, or to assess the value or value ranges of an attribute that a given object is likely to have. In this view, classification and regression are the two major types of prediction problems where classification is used to predict discrete or nominal values, while regression is used to predict continuous or ordered values.

Classification and prediction have numerous applications including credit approval, medical diagnosis, performance prediction, and selective marketing.

3.2 ISSUES REGARDING CLASSIFICATION AND PREDICTION

Preparing the data for classification and prediction. The following preprocessing steps may be applied to the data in order to help improve the accuracy, efficiency, and scalability of the classification or prediction process.

- Data cleaning. This refers to the preprocessing of data in order to remove or reduce noise (by applying smoothing techniques, for example), and the treatment of missing values (e.g., by replacing a missing value with the most commonly occurring value for that attribute, or with the most probable value based on statistics). Although most classification algorithms have some mechanisms for handling noisy or missing data, this step can help reduce confusion during learning.
- Relevance analysis. Many of the attributes in the data may be irrelevant to the classification or prediction task. For example, data recording the day of the week on which a bank loan application was _led is unlikely to be relevant to the success of the application. Furthermore, other attributes may be redundant. Hence, relevance analysis

may be performed on the data with the aim of removing any irrelevant or redundant attributes from the learning process. In machine learning, this step is known as feature selection. Including such attributes may otherwise slow down, and possibly mislead, the learning step. Ideally, the time spent on relevance analysis, when added to the time spent on learning from the resulting \reduced" feature subset, should be less than the time that would have been spent on learning from the original set of features. Hence, such analysis can help improve classification efficiency and scalability.

- Data transformation. The data can be generalized to higher-level concepts. Concept hierarchies may be used for this purpose. This is particularly useful for continuous-valued attributes. For example, numeric values for the attribute income may be generalized to discrete ranges such as low, medium, and high. Similarly, nominal-valued attributes, like street, can be generalized to higher-level concepts, like city. Since generalization compresses the original training data, fewer input/output operations may be involved during learning. The data may also be normalized, particularly when neural networks or methods involving distance measurements are used in the learning step. Normalization involves scaling all values for a given attribute so that they fall within a small specified range, such as -1.0 to 1.0, or 0 to 1.0. In methods which use distance measurements, for example, this would prevent attributes with initially large ranges (like, say income) from outweighing attributes with initially smaller ranges (such as binary attributes).

<u>3.3: CLASSIFICATION BY DECISION TREE ALGORITHM</u>

```
1) create a node N;

2) if samples are all of the same class, C then

3) return N as a leaf node labeled with the class C;

4) if attribute-list is empty then

5) return N as a leaf node labeled with the most common class in samples; // majority voting

6) select test-attribute, the attribute among attribute-list with the highest information gain;

7) label node N with test-attribute;
```

```
8) for each known value ai of test-attribute // partition the samples
9) grow a branch from node N for the condition test-attribute=ai;
10) let si be the set of samples in samples for which test-attribute=ai; // a partition
11) if si is empty then
12) attach a leaf labeled with the most common class in samples;
13) else attach the node returned by Generate decision tree(si, attribute-list - test-attribute);
```

3.3.1 DECISION TREE INDUCTION:

The basic algorithm for decision tree induction is a greedy algorithm which constructs decision trees in a top-down recursive divide-and-conquer manner.

The basic strategy is as follows:

- The tree starts as a single node representing the training samples (step 1).
- If the samples are all of the same class, then the node becomes a leaf and is labeled with that class (steps 2and 3).
- Otherwise, the algorithm uses an entropy-based measure known as information gain as a heuristic for selecting the attribute that will best separate the samples into individual classes (step 6). This attribute becomes the \test" or \decision" attribute at the node (step 7). In this version of the algorithm, all attributes are categorical, i.e., discrete-valued. Continuous-valued attributes must be discrete.
- A branch is created for each known value of the test attribute, and the samples are partitioned accordingly(steps 8-10).
- The algorithm uses the same process recursively to form a decision tree for the samples at each partition. Once an attribute has occurred at a node, it need not be considered in any of the node's descendents (step 13).
- The recursive partitioning stops only when any one of the following conditions is true:

    1. All samples for a given node belong to the same class (step 2 and 3), or

2. There are no remaining attributes on which the samples may be further partitioned (step 4). In this case, majority voting is employed (step 5). This involves converting the given node into a leaf and labeling it with the class in majority among samples. Alternatively, the class distribution of the node samples maybe stored; or

3. There are no samples for the branch test-attribute=ai (step 11). In this case, a leaf is created with the majority class in samples (step 12).

Attribute selection measure. The information gain measure is used to select the test attribute at each node in the tree. Such a measure is referred to as an attribute selection measure or a measure of the goodness of split. The attribute with the highest information gain (or greatest entropy reduction) is chosen as the test attribute for the current node. This attribute minimizes the information needed to classify the samples in the resulting partitions and reflects the least randomness or \impurity" in these partitions. Such an information-theoretic approach minimizes the expected number of tests needed to classify an object and guarantees that a simple (but not necessarily the simplest) tree is found.

3.4 BAYESIAN CLASSIFIERS

Bayesian classifiers are statistical classifiers. They can predict class membership probabilities, such as the probability that a given sample belongs to a particular class. Bayesian classification is based on Bayes theorem, described below. Studies comparing classification algorithms have found a simple Bayesian classifier known as the naive Bayesian classifier to be comparable in performance with decision tree and neural network classifiers. Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases.

Naive Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called class conditional independence. It is made to simplify the computations involved, and in this sense, is considered \naive". Bayesian belief networks are graphical models, which unlike naive Bayesian classifiers, allow the representation of dependencies among subsets of attributes. Bayesian belief networks can also be used for classification.

3.4.1BAYES THEOREM:

Let X be a data sample whose class label is unknown. Let H be some hypothesis, such as that the data sample X belongs to a specified class C. For classification problems, we want to determine P(HjX), the probability that the hypothesis H holds given the observed data sample X.

P(HjX) is the posterior probability, or a posteriori probability, of H conditioned on X. For example, suppose the world of data samples consists of fruits, described by their color and shape. Suppose that X is red and round, and that H is the hypothesis that X is an apple. Then P(HjX) reflects our confidence that X is an apple given that we have seen that X is red and round. In contrast, P(H) is the prior probability, or a priori probability of H. For example, this is the probability that any given data sample is an apple, regardless of how the data sample looks. The posterior probability, P(HjX) is based on more information (such as background knowledge) than the prior probability, P(H), which is independent of X.

Similarly, P(XjH) is the posterior probability of X conditioned on H. That is, it is the probability that X is red and round given that we know that it is true that X is an apple. P(X) is the prior probability of X.

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}$$

3.4.2 NAIVE BAYESIAN CLASSIFIER:

The naive Bayesian classifier, or simple Bayesian classifier, works as follows:

1. Each data sample is represented by an n-dimensional feature vector, X = (x1; x2; : : :; xn), depicting n measurements made on the sample from n attributes, respectively A1;A2; ::;An.

2. Suppose that there are m classes, C1; C2; : : :; Cm. Given an unknown data sample, X (i.e., having no class label), the classifier will predict that X belongs to the class having the highest

posterior probability, conditioned on X. That is, the naive Bayesian classi_er assigns an unknown sample X to the class Ci if and only if :

$$P(C_i|X) > P(C_j|X) \text{ for } 1 \leq j \leq m, j \neq i.$$

Thus we maximize P(CijX). The class Ci for which P(CijX) is maximized is called the maximum posteriori hypothesis. By Bayes theorem,

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}.$$

P(CijX) =P(XjCi)P(Ci)/P(X)

3. As P(X) is constant for all classes, only P(X/Ci)P(Ci) need be maximized. If the class prior probabilities arenot known, then it is commonly assumed that the classes are equally likely, i.e. P(C1) = P(C2) = : : : = P(Cm), and we would therefore maximize P(X/Ci). Otherwise, we maximize P(X/CiP(Ci)). Note that the class prior probabilities may be estimated by P(Ci) = si/s , where si is the number of training samples of class Ci, and s is the total number of training samples.

4. Given data sets with many attributes, it would be extremely computationally expensive to compute P(Xj/Ci). In order to reduce computation in evaluating P(Xj/Ci), the naive assumption of class conditional independence is made. This presumes that the values of the attributes are conditionally independent of one another, given the class label of the sample, i.e., that there are no dependence relationships among the attributes. Thus,

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i).$$

The probabilities P(x1jCi); P(x2jCi); : : :; P(xnjCi) can be estimated from the training samples, where:

(a) If Ak is categorical, then P(xkj/Ci) = sik/si ,where sik is the number of training samples of class Ci having the value xk for Ak, and si is the number of training samples belonging to Ci.

(b) If Ak is continuous-valued, then the attribute is assumed to have a Gaussian distribution. Therefore,

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}) = \frac{1}{\sqrt{2\pi}\sigma_{C_i}} e^{-\frac{(x-\mu_{C_i})^2}{2\sigma^2_{C_i}}},$$

where P(xkjCi) = g(xk; _Ci,where g(xk; _Ci; _Ci) is the Gaussian (normal) density function for attribute Ak, while _Ci and _Ci are the mean and variance respectively given the values for attribute Ak for training samples of class Ci.

5. In order to classify an unknown sample X, P(XjCi)P(Ci) is evaluated for each class Ci. Sample X is then assigned to the class Ci if and only if :

$$P(X|C_i)P(C_i) > P(X|C_j)P(C_j) \text{ for } 1 \leq j \leq m,\ j \neq i.$$

3.4.3 EFFECTIVENESS OF BAYESIAN CLASSIFIER

In theory, Bayesian classifiers have the minimum error rate in comparison to all other classifiers. However, in practice this is not always the case owing to inaccuracies in the assumptions made for its use, such as class conditional independence, and the lack of available probability data. However, various empirical studies of this classifier in comparison to decision tree and neural network classifiers have found it to be comparable in some domains.

Bayesian classifiers are also useful in that they provide a theoretical justification for other classifiers which do not explicitly use Bayes theorem. For example, under certain assumptions, it can be shown that many neural network and curve fitting algorithms output the maximum posteriori hypothesis, as does the naive Bayesian classifier

CHAPTER 4: CLUSTER TECHNIQUES

Different approaches to clustering data can be described with the help of the hierarchy shown in Figure 7 (other taxonometric representations of clustering methodology are possible; ours is based on the discussion in Jain and Dubes [1988]). At the top level, there is a distinction between hierarchical and partitional approaches (hierarchical methods produce a nested series of partitions, while partitional methods produce only one). The taxonomy shown in Figure 7 must be supplemented by a discussion of cross-cutting issues that may (in principle) affect all of the different approaches regardless of their placement in the taxonomy.

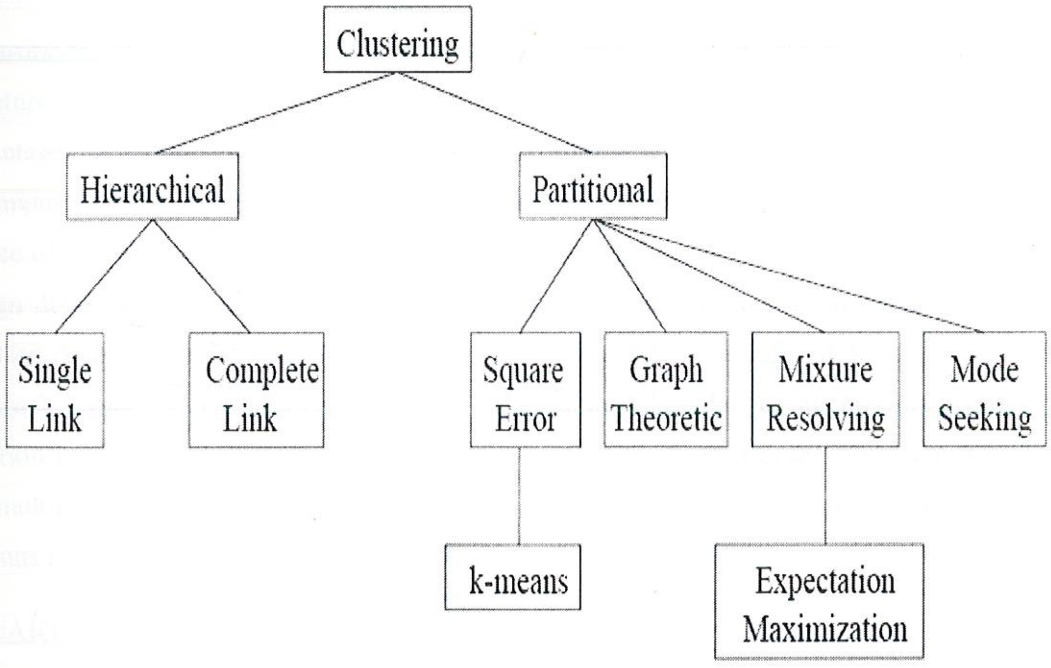


Figure 7. A taxonomy of clustering approaches.

4.1 HIERARCHICAL CLUSTERING ALGORITHM

Most hierarchical clustering algorithms are variants of the single-link, complete-link, and minimum-variance algorithms. Of these, the single-link and complete link algorithms are most popular. These two algorithms differ in the way they characterize the similarity between a pair of clusters.

In the single-link method, the distance between two clusters is the *minimum* of the distances between all pairs of patterns drawn from the two clusters (one pattern from the first cluster, the other from the second). In the complete-link algorithm, the distance between two clusters is the *maximum* of all pair wise distances between patterns in the two clusters. In either case, two clusters are merged to form a larger cluster based on minimum distance criteria. The complete-link algorithm produces tightly bound or compact clusters. The single-link algorithm, by contrast, suffers from a chaining effect .It has a tendency to produce clusters that are straggly or elongated.

4.2 PARTITIONED ALGORITHMS

A partitional clustering algorithm obtains a single partition of the data instead of a clustering structure, such as the dendrogram produced by a hierarchical technique. Partitional methods have advantages in applications involving large data sets for which the construction of a dendrogram is computationally prohibitive. A problem accompanying the use of a partitional algorithm is the choice of the number of desired output clusters. A seminal paper provides guidance on this key design decision. The partitional techniques usually produce clusters by optimizing a criterion function defined either locally (on a subset of the patterns) or globally (defined over all of the patterns). Combinatorial search of the set of possible labelings for an optimum value of a criterion is clearly computationally prohibitive. In practice, therefore, the algorithm is typically run multiple times with different starting states, and the best configuration obtained from all of the runs is used as the output clustering.

SQUARED ERROR CLUSTERING METHOD

(1) Select an initial partition of the patterns with a fixed number of clusters and cluster centers.

(2) Assign each pattern to its closest cluster center and compute the new cluster centers as the centroids of the clusters. Repeat this step until convergence is achieved, i.e., until the cluster membership is stable.

(3) Merge and split clusters based on some heuristic information, optionally

repeating step 2.

(1) Choose *k* cluster centers to coincide with *k* randomly-chosen patterns or *k* randomly defined points inside the hypervolume containing the pattern set.

(2) Assign each pattern to the closest cluster center.

(3) Recompute the cluster centers using the current cluster memberships.

(4) If a convergence criterion is not met, go to step 2. Typical convergence criteria are: no (or minimal) reassignment of patterns to new cluster centers, or minimal decrease in squared error.

Several variants of the *k*-means algorithm have been reported in the literature. Some of them attempt to select a good initial partition so that the algorithm is more likely to find the global minimum value.

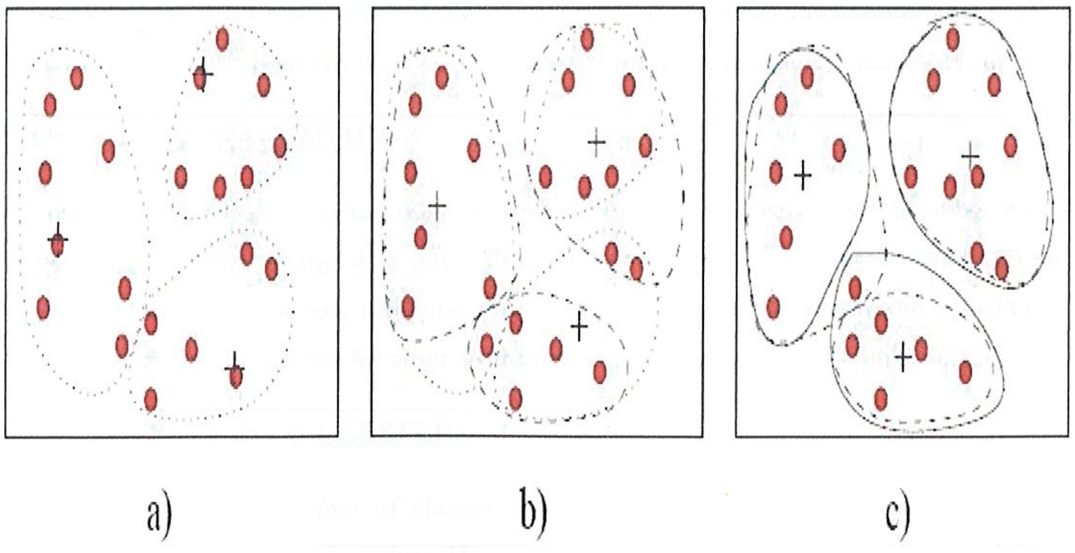


Figure : Clustering of a set of points based on the *k*-means method

The two key features of *k*-means which make it efficient are often regarded as its biggest drawbacks:

- Euclidean distance is used as a metric and variance is used as a measure of cluster scatter.
- The number of clusters *k* is an input parameter: an inappropriate choice of *k* may yield poor results. That is why, when performing k-means, it is important to run diagnostic checks for determining the number of clusters in the data set.

A key limitation of *k*-means is its cluster model. The concept is based on spherical clusters that are separable in a way so that the mean value converges towards the cluster center. The clusters are expected to be of similar size, so that the assignment to the nearest cluster center is the correct assignment. When for example applying *k*-means with a value of $k = 3$ onto the well-known Iris flower data set, the result often fails to separate the three Iris species contained in the data set. With $k = 2$, the two visible clusters (one containing two species) will be discovered, whereas with $k = 3$ one of the two clusters will be split into two even parts. In fact, $k = 2$ is more appropriate for this data set, despite the data set containing 3 *classes*. As with any other clustering algorithm, the *k*-means result relies on the data set to satisfy the assumptions made by the clustering algorithms. It works very well on some data sets, while failing miserably on others.

APPLICATION OF ALGORITHM

k-means clustering in particular when using heuristics such as Lloyd's algorithm is rather easy to implement and apply even on large data sets. As such, it has been successfully used in various topics, ranging from market segmentation, computer vision, geo statistics and astronomy to agriculture. It often is used as a preprocessing step for other algorithms, for example to find a starting configuration.

4.3 REPRESENTATION OF CLUSTERS

In applications where the number of classes or clusters in a data set must be discovered, a partition of the data set is the end product. Here, a partition gives an idea about the reparability of the data points into clusters and whether it is meaningful to employ a supervised classifier that assumes a given number of classes in the data set. However, in many other applications that involve decision making, the resulting clusters have to be represented or described in a compact form to achieve *data abstraction*. Even though the construction of a cluster representation is an important step in decision making, it has not been examined closely by researchers. The notion of cluster representation was introduced in Duran and Odell [1974] and was subsequently studied in Diday and Simon [1976] and Michalski et al. [1981]. They suggested the following representation schemes:

(1) Represent a cluster of points by their centroid or by a set of distant points in the cluster. Figure 17 depicts these two ideas.

(2) Represent clusters using nodes in a classification tree. This is illustrated in Figure 18.

(3) Represent clusters by using conjunctive logical expressions. For example, the expression $X1>3, X2<2$ in Figure 18 stands for the logical statement '$X1$ is greater than 3' *and* '$X2$ is less than 2'.

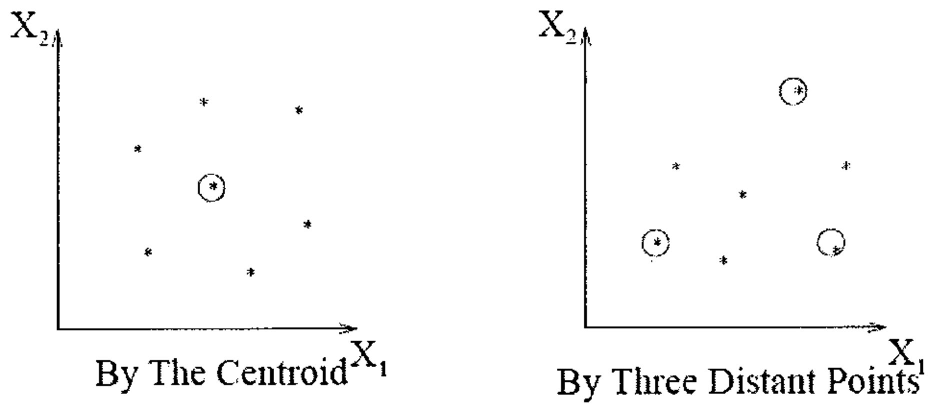
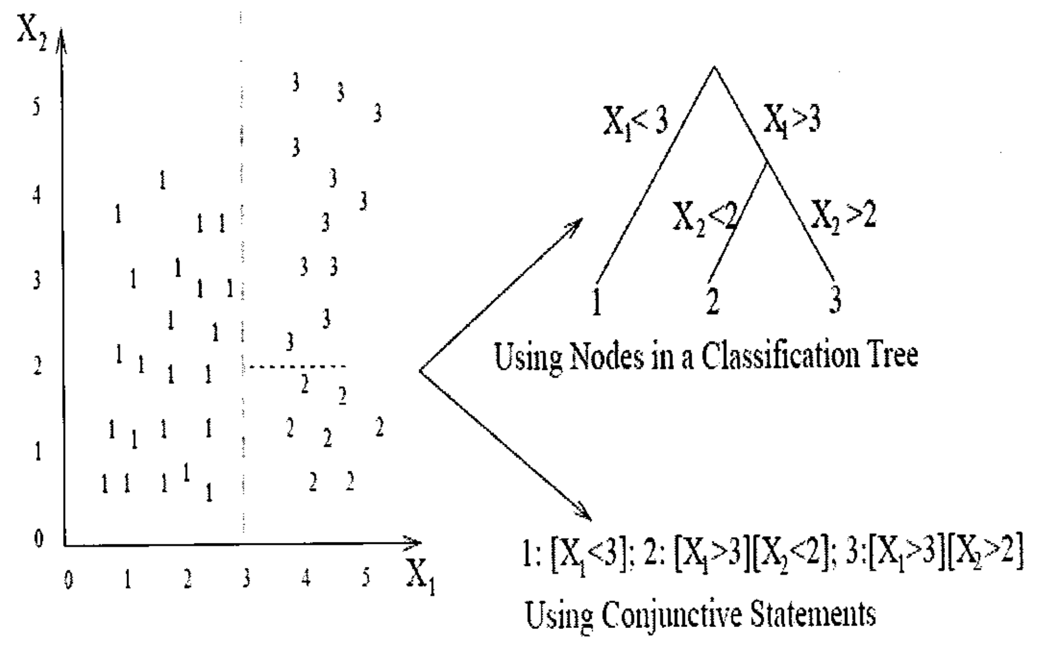


Figure 17. Representation of a cluster by points.



X_2
X_1
$X_1<3$
$X_1>3$
$X_2<2$
$X_2>2$
1 2 3
Using Nodes in a Classification Tree
1: $[X_1<3]$; 2: $[X_1>3][X_2<2]$; 3: $[X_1>3][X_2>2]$
Using Conjunctive Statements

Figure 18. Representation of clusters by a classification tree or by conjunctive statements.

Use of the centroid to represent a cluster is the most popular scheme. It works well when the clusters are compact or isotropic. However, when the clusters are elongated or non-isotropic, then this scheme fails to represent them properly. In such a case, the use of a collection of boundary points in a cluster captures its shape well. The number of points used to represent a cluster should increase as the complexity of its shape increases. The two different representations illustrated in Figure 18 are equivalent. Every path in a classification tree from the root node to a leaf node corresponds to a conjunctive statement. An important limitation of the typical use of the simple conjunctive concept representations is that they can describe only rectangular or isotropic clusters in the feature space.

Data abstraction is useful in decision making because of the following:

(1) It gives a simple and intuitive description of clusters which is easy for human comprehension. In both conceptual and symbolic clustering [Gowda and Diday 1992] this representation is obtained without using an additional step. These algorithms generate the clusters as well as their descriptions. A set of fuzzy rules can be obtained from fuzzy

clusters of a data set. These rules can be used to build fuzzy classifiers and fuzzy controllers.

(2) It helps in achieving data compression that can be exploited further by a computer [Murty and Krishna 1980]. Figure 19(a) shows samples belonging to two chain-like clusters labeled 1 and 2. A partitional clustering like the *k*-means algorithm cannot separate these two structures properly. The single-link algorithm works well on this data, but is computationally expensive. So a hybrid approach may be used to exploit the desirable properties of both these algorithms. We obtain 8 sub clusters of the data using the (computationally efficient) *k*-means algorithm. Each of these sub clusters can be represented by their centroids as shown in Figure 19(a). Now the single- link algorithm can be applied on these centroids alone to cluster them into 2 groups. The resulting groups are shown in Figure 19(b). Here, a data reduction is achieved by representing the sub-clusters by their centroids.

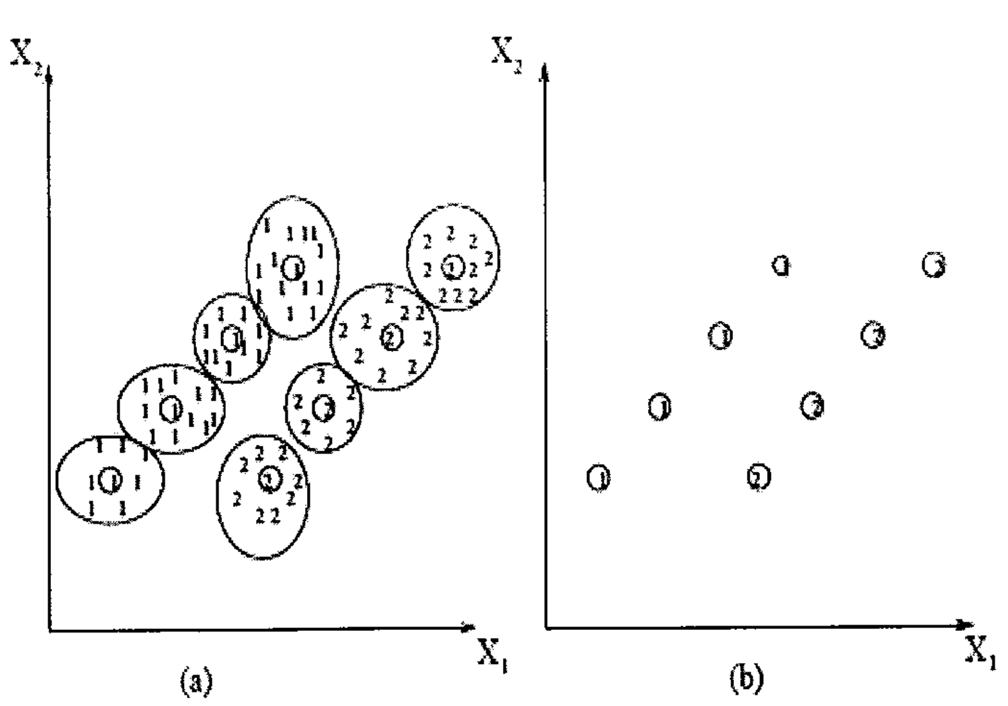


Figure 19. Data compression by clustering.

(3) It increases the efficiency of the decision making task. In a cluster based document retrieval technique [Salton 1991], a large collection of documents is clustered and each of the clusters is represented using its centroid. In order to retrieve documents relevant to a query, the query is matched with the cluster centroids rather than with all the documents. This helps in retrieving relevant documents efficiently. Also in several applications involving large data sets, clustering is used to perform indexing, which helps in efficient decision making.

4.4: K-MEAN CLUSTERING EXAMPLE

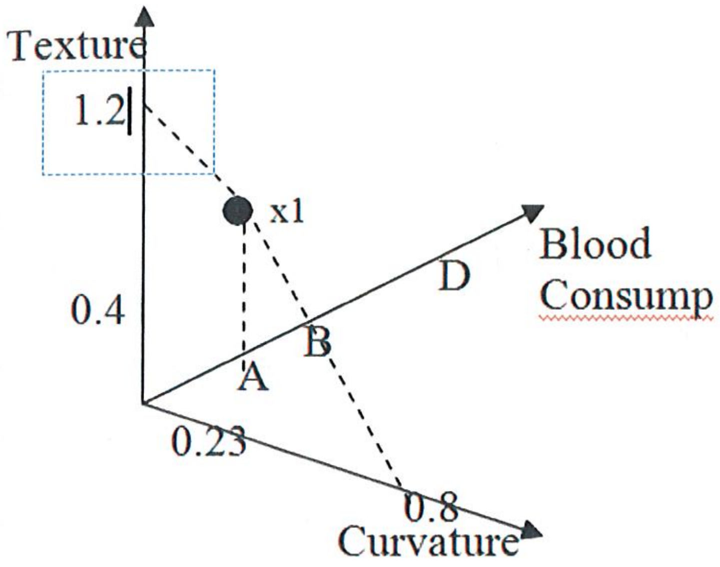
Clustering allows for *unsupervised learning*. That is, the machine / software will learn on its own, using the data (learning set), and will classify the objects into a particular class – for example, if our class (decision) attribute is *Tumor Type* and its values are: malignant, benign, etc. - these will be the classes. They will be represented by cluster1, cluster2, etc. However, the class information is never provided to the algorithm. The class information can be used later on, to evaluate how accurately the algorithm classified the objects.

	Curvature	Texture	Blood Consump	Tumor Type
x1	0.8	1.2	A	Benign
x2	0.75	1.4	B	Benign
x3	0.23	0.4	D	Malignant
x4	0.23	0.5	D	Malignant
.				
.				

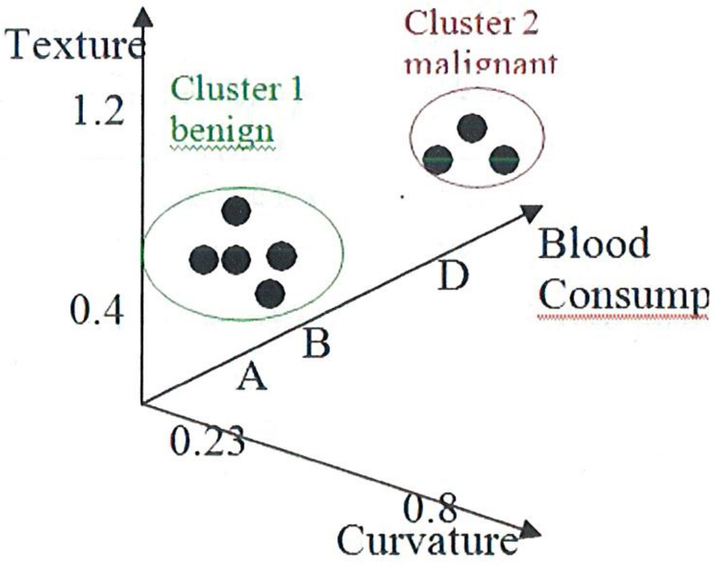
$\rightarrow$

	Curvature	Texture	Blood Consump	Tumor Type
x1	0.8	1.2	A	Benign
x2	0.75	1.4	B	Benign
x3	0.23	0.4	D	t
x4	0.23	0.5	D	t
.				
.				

(learning set)



The way we do that, is by plotting the objects from the database into space. Each attribute is one dimension:



After all the objects are plotted, we will calculate the distance between them, and the ones that are close to each other – we will group them together, i.e. place them in the same cluster.

With the K-Means algorithm, we recall it works as follows:

K MEANS CLUSTERING

- Partition clustering approach.
- Each cluster is associated with the centroid (central point).
- Each point is assigned to the cluster with nearest centroid.
- Number of clusters k must be specified.
- The basics of algorithm are very simple.

DETAIL:

- Initially centroids are often chosen randomly.
    - Clusters produced vary from one run to another.
- The centroid is the mean point in clusters.
- Closeness is measured by Euclidean distance formula
- K-MEANS will converge , most of the convergences occur in the first few steps.

EXAMPLE:

Problem: Cluster the following eight points (with (x, y) representing locations) into three clusters A1(2, 10) A2(2, 5) A3(8, 4) A4(5, 8) A5(7, 5) A6(6, 4) A7(1, 2) A8(4, 9). Initial cluster centers are: A1(2, 10), A4(5, 8) and A7(1, 2). The distance function between two points *a=(x1, y1)* and *b=(x2, y2)* is defined as: $\rho(a, b) = |x2 - x1| + |y2 - y1|$.

Use k-means algorithm to find the three cluster centers after the second iteration.

Solution:

		(2, 10)	(5, 8)	(1, 2)	
	Point	**Dist Mean 1**	**Dist Mean 2**	**Dist Mean 3**	**Cluster**
A1	(2, 10)				
A2	(2, 5)				
A3	(8, 4)				

A4	(5, 8)				
A5	(7, 5)				
A6	(6, 4)				
A7	(1, 2)				
A8	(4, 9)				

First we list all points in the first column of the table above. The initial cluster centers – means, are (2, 10), (5, 8) and (1, 2) - chosen randomly. Next, we will calculate the distance from the first point (2, 10) to each of the three means, by using the distance function:

point	mean1
$x1, y1$	$x2, y2$
(2, 10)	(2, 10)

$\rho(a, b) = |x2 - x1| + |y2 - y1|$

$\rho(point, mean1) = |x2 - x1| + |y2 - y1|$

$= |2 - 2| + |10 - 10|$

$= 0 + 0$

$= 0$

point	mean2
$x1, y1$	$x2, y2$
(2, 10)	(5, 8)

$\rho(a, b) = |x2 - x1| + |y2 - y1|$

$\rho(point, mean2) = |x2 - x1| + |y2 - y1|$

$= |5 - 2| + |8 - 10|$

$= 3 + 2$

$= 5$

point	mean3
x1, y1	*x2, y2*
(2, 10)	(1, 2)

$\rho(a, b) = |x2 - x1| + |y2 - y1|$

$\rho(point, mean2) = |x2 - x1| + |y2 - y1|$

$= |1 - 2| + |2 - 10|$

$= 1 + 8$

$= 9$

So, we fill in these values in the table:

		(2, 10)	(5, 8)	(1, 2)	
	Point	**Dist Mean 1**	**Dist Mean 2**	**Dist Mean 3**	**Cluster**
A1	(2, 10)	0	5	9	1
A2	(2, 5)				
A3	(8, 4)				
A4	(5, 8)				
A5	(7, 5)				

A6	(6, 4)				
A7	(1, 2)				
A8	(4, 9)				

So, which cluster should the point (2, 10) be placed in? The one, where the point has the shortest distance to the mean – that is mean 1 (cluster 1), since the distance is 0.

Cluster 1	Cluster 2	Cluster 3
(2, 10)		

So, we go to the second point (2, 5) and we will calculate the distance to each of the three means, by using the distance function:

point	mean1
$x1, y1$	$x2, y2$
(2, 5)	(2, 10)

$\rho(a, b) = |x2 - x1| + |y2 - y1|$

$\rho(point, mean1) = |x2 - x1| + |y2 - y1|$

$= |2 - 2| + |10 - 5|$

$= 0 + 5$

$= 5$

point	mean2
$x1, y1$	$x2, y2$
(2, 5)	(5, 8)

$\rho(a, b) = |x2 - x1| + |y2 - y1|$

ρ(point, mean2) = |*x2* – *x1*| + |*y2* – *y1*|

= |5 – 2| + |8 – 5|

= 3 + 3

= 6

point	mean3
x1, *y1*	*x2*, *y2*
(2, 5)	(1, 2)

ρ(a, b) = |*x2* – *x1*| + |*y2* – *y1*|

ρ(point, mean2) = |*x2* – *x1*| + |*y2* – *y1*|

= |1 – 2| + |2 – 5|

= 1 + 3

= 4

So, we fill in these values in the table:

		(2, 10)	(5, 8)	(1, 2)	
	Point	**Dist Mean 1**	**Dist Mean 2**	**Dist Mean 3**	**Cluster**
A1	(2, 10)	0	5	9	1
A2	(2, 5)	5	6	4	3
A3	(8, 4)				
A4	(5, 8)				
A5	(7, 5)				

A6	(6, 4)				
A7	(1, 2)				
A8	(4, 9)				

So, which cluster should the point (2, 5) be placed in? The one, where the point has the shortest distance to the mean – that is mean 3 (cluster 3), since the distance is 0.

Cluster 1	Cluster 2	Cluster 3
(2, 10)		(2, 5)

Analogically, we fill in the rest of the table, and place each point in one of the clusters:

		(2, 10)	(5, 8)	(1, 2)	
	Point	**Dist Mean 1**	**Dist Mean 2**	**Dist Mean 3**	**Cluster**
A1	(2, 10)	0	5	9	1
A2	(2, 5)	5	6	4	3
A3	(8, 4)	12	7	9	2
A4	(5, 8)	5	0	10	2
A5	(7, 5)	10	5	9	2
A6	(6, 4)	10	5	7	2
A7	(1, 2)	9	10	0	3
A8	(4, 9)	3	2	10	2

Cluster 1	Cluster 2	Cluster 3
(2, 10)	(8, 4)	(2, 5)
	(5, 8)	(1, 2)
	(7, 5)	
	(6, 4)	
	(4, 9)	

Next, we need to re-compute the new cluster centers (means). We do so, by taking the mean of all points in each cluster.

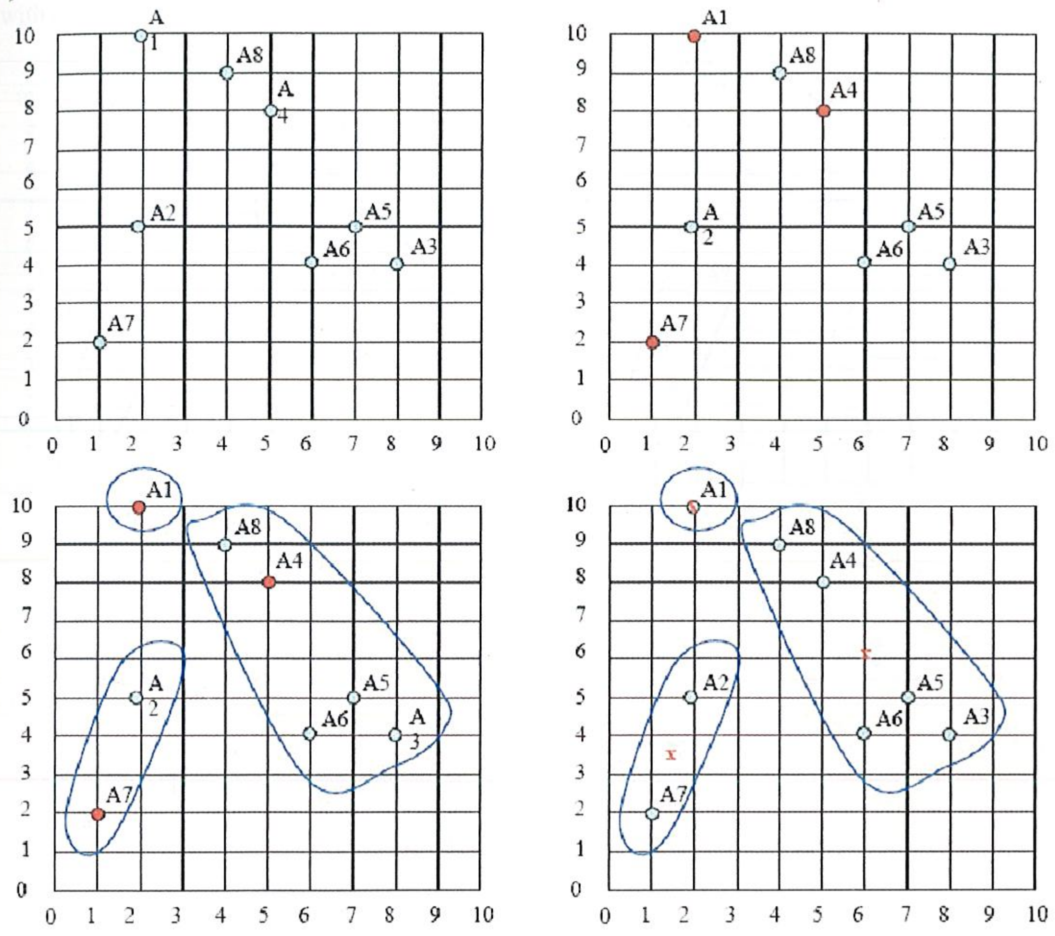
For Cluster 1, we only have one point A1(2, 10), which was the old mean, so the cluster center remains the same.

For Cluster 2, we have ((8+5+7+6+4)/5, (4+8+5+4+9)/5) = (6, 6)

For Cluster 3, we have ((2+1)/2, (5+2)/2) = (1.5, 3.5) new clusters: 1: {A1}, 2: {A3, A4, A5, A6, A8}, 3: {A2, A7}

b) centers of the new clusters:
C1= (2, 10), C2= ((8+5+7+6+4)/5, (4+8+5+4+9)/5) = (6, 6), C3= ((2+1)/2, (5+2)/2) = (1.5, 3.5)

c)

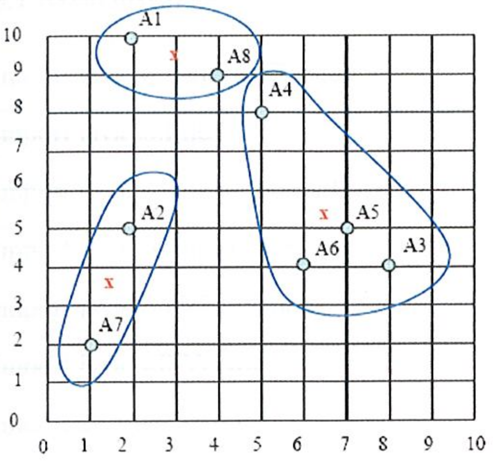


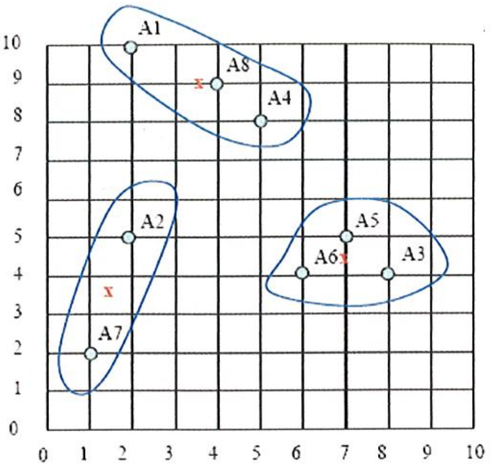
The initial cluster centers are shown in red dot. The new cluster centers are shown in red x.

That was Iteration1 (epoch1). Next, we go to Iteration2 (epoch2), Iteration3, and so on until the means do not change anymore.

In Iteration2, we basically repeat the process from Iteration1 this time using the new means we computed.

d)
We would need two more epochs. After the 2nd epoch the results would be:
1: {A1, A8}, 2: {A3, A4, A5, A6}, 3: {A2, A7}
with centers C1=(3, 9.5), C2=(6.5, 5.25) and C3=(1.5, 3.5).
After the 3rd epoch, the results would be:
1: {A1, A4, A8}, 2: {A3, A5, A6}, 3: {A2, A7}
with centers C1=(3.66, 9), C2=(7, 4.33) and C3=(1.5, 3.5).





CHAPTER 5 CODING OF THE PROJECT

5.1 Areaclust.java:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Vector;
import java.util.Iterator;
import java.util.Scanner;
import java.util.StringTokenizer;


public class Areaclust {
  private static int l,t;
  private static int rowtota=0;
  private static int rowtotal=0;
  private static int ClustNumber;
  private String ad;
  public Areaclust (String as) throws FileNotFoundException, IOException{
    ad=as;
    Vector dataPoints = new Vector();

File file1 = new File("hospitals.csv");
BufferedReader bufRdrr = new BufferedReader(new FileReader(file1));
BufferedReader bufRdrr1 = new BufferedReader(new FileReader(file1));
String lines = null;
int row1 = 0;
int col1 = 0;


while((lines = bufRdrr.readLine()) != null )
{
rowtotal++;
}

File file = new File("area.csv");
BufferedReader bufRdr = new BufferedReader(new FileReader(file));
BufferedReader bufRdr2 = new BufferedReader(new FileReader(file));
String line = null;
int row = 0;
int col = 0;


while((line = bufRdr.readLine()) != null )
{
rowtota++;
}
String [][] numbers = new String [rowtota][3];
double Cordx[] =new double[rowtota];
double Cordy[] =new double[rowtota];
String Cordz[] =new String[rowtota];
```

```
//read each line of text file
while((line = bufRdr2.readLine()) != null && row< rowtota )
{
StringTokenizer st = new StringTokenizer(line,",");
while (st.hasMoreTokens())
{
//get next token and store it in the array
numbers[row][col] = st.nextToken();
col++;
}
col = 0;
row++;
}


for(row=0;row < rowtota;row++)
{
for(col=0; col<3;col++)
{
System.out.print(" " + numbers[row][col]);
Cordz[row]=numbers[row][2];
}
System.out.println(" ");
}
for(row=0;row<rowtota;row++)
{
```

```
Cordx[row]=Double.parseDouble(numbers[row][0]);
Cordy[row]=Double.parseDouble(numbers[row][1]);
}



for(l=0;l<rowtota;l++)
{
    dataPoints.add(new DataPoint(Cordx[l],Cordy[l],+l+"th location"));
}


    System.out.println("Enter no. of clusters : ");
    Scanner input = new Scanner(System.in);
    ClustNumber=input.nextInt();


    Jca jca = new Jca(ClustNumber,1000,dataPoints);
    jca.startAnalysis();


    Vector[] v = jca.getClusterOutput();
    for (int i=0; i<v.length; i++){
      Vector tempV = v[i];
      System.out.println("-----------Cluster"+(i+1)+"---------");
      Iterator iter = tempV.iterator();
     while(iter.hasNext()){
        DataPoint dpTemp = (DataPoint)iter.next();
System.out.println(dpTemp.getObjName()+"["+dpTemp.getX()+","+dpTemp.getY()+"]");


        for(row=0;row < rowtota;row++)
```

```
          {
            if(Cordx[row]==dpTemp.getX()&& Cordy[row]==dpTemp.getY())
          {
           if(ad.equals(Cordz[row]))
               {
                 t=(i+1);



               }
            }
          }
        }


      }
     System.out.println(" ]"+t);
    }


    Areaclust(String[] Cordxx, double[] Cordyx, double[] Cordzx, int rox) {
       String[] corx=Cordxx;
       double[] cory=Cordyx;
      double[] corz=Cordzx;


    }

  }
```

5.2 DataPoint.java:

```
public class DataPoint {
   private double mX,mY;
   private String mObjName;
   private Cluster mCluster;
   private double mEuDt;

   public DataPoint(double x, double y, String name) {
      this.mX = x;
      this.mY = y;
      this.mObjName = name;
      this.mCluster = null;
   }

   public void setCluster(Cluster cluster) {
      this.mCluster = cluster;
      calcEuclideanDistance();
   }

   public void calcEuclideanDistance() {

   //called when DP is added to a cluster or when a Centroid is recalculated.
      mEuDt = Math.sqrt(Math.pow((mX - mCluster.getCentroid().getCx()),
2) + Math.pow((mY - mCluster.getCentroid().getCy()), 2));
   }
```

```
    public double testEuclideanDistance(Centroid c) {
        return Math.sqrt(Math.pow((mX - c.getCx()), 2) + Math.pow((mY - c.getCy()), 2));
    }

    public double getX() {
        return mX;
    }

    public double getY() {
        return mY;
    }

    public Cluster getCluster() {
        return mCluster;
    }

    public double getCurrentEuDt() {
        return mEuDt;
    }

    public String getObjName() {
        return mObjName;
    }

}
```

5.3 Jca.java:

```
public class Jca {
    private Cluster[] clusters;
    private int miter;
    private Vector mDataPoints = new Vector();
    private double mSWCSS;

    public Jca(int k, int iter, Vector dataPoints) {
        clusters = new Cluster[k];
        for (int i = 0; i < k; i++) {
            clusters[i] = new Cluster("Cluster" + i);
        }
        this.miter = iter;
        this.mDataPoints = dataPoints;
    }

    private void calcSWCSS() {
        double temp = 0;
        for (int i = 0; i < clusters.length; i++) {
            temp = temp + clusters[i].getSumSqr();
        }
        mSWCSS = temp;
    }

    public void startAnalysis() {
        //set Starting centroid positions - Start of Step 1
```

```
setInitialCentroids();
int n = 0;
//assign DataPoint to clusters
loop1: while (true) {
    for (int l = 0; l < clusters.length; l++)
    {
        clusters[l].addDataPoint((DataPoint)mDataPoints.elementAt(n));
        n++;
        if (n >= mDataPoints.size())
            break loop1;
    }
}

//calculate E for all the clusters
calcSWCSS();

//recalculate Cluster centroids - Start of Step 2
for (int i = 0; i < clusters.length; i++) {
    clusters[i].getCentroid().calcCentroid();
}

//recalculate E for all the clusters
calcSWCSS();

for (int i = 0; i < miter; i++) {
    //enter the loop for cluster 1
    for (int j = 0; j < clusters.length; j++) {
        for (int k = 0; k < clusters[j].getNumDataPoints(); k++) {

          //pick the first element of the first cluster
          //get the current Euclidean distance
          double tempEuDt = clusters[j].getDataPoint(k).getCurrentEuDt();
          Cluster tempCluster = null;
          boolean matchFoundFlag = false;

          //call testEuclidean distance for all clusters
          for (int l = 0; l < clusters.length; l++) {

          //if testEuclidean < currentEuclidean then
            if                         (tempEuDt                         >
clusters[j].getDataPoint(k).testEuclideanDistance(clusters[l].getCentroid())) {
              tempEuDt                                                   =
clusters[j].getDataPoint(k).testEuclideanDistance(clusters[l].getCentroid());
              tempCluster = clusters[l];
              matchFoundFlag = true;
            }
            //if statement - Check whether the Last EuDt is > Present EuDt

            }
//for variable 'l' - Looping between different Clusters for matching a Data Point.
//add DataPoint to the cluster and calcSWCSS

    if (matchFoundFlag) {
            tempCluster.addDataPoint(clusters[j].getDataPoint(k));
            clusters[j].removeDataPoint(clusters[j].getDataPoint(k));
```

```
                    for (int m = 0; m < clusters.length; m++) {
                        clusters[m].getCentroid().calcCentroid();
                    }

//for variable 'm' - Recalculating centroids for all Clusters

                    calcSWCSS();
                }

//if statement - A Data Point is eligible for transfer between Clusters.
                }
                //for variable 'k' - Looping through all Data Points of the current Cluster.
            }//for variable 'j' - Looping through all the Clusters.
        }//for variable 'i' - Number of iterations.
    }

    public Vector[] getClusterOutput() {
        Vector v[] = new Vector[clusters.length];
        for (int i = 0; i < clusters.length; i++) {
            v[i] = clusters[i].getDataPoints();
        }
        return v;
    }


    private void setInitialCentroids() {
        //kn = (round((max-min)/k)*n)+min where n is from 0 to (k-1).
        double cx = 0, cy = 0;
        for (int n = 1; n <= clusters.length; n++) {
            cx = (((getMaxXValue() - getMinXValue()) / (clusters.length + 1)) * n) +
getMinXValue();
            cy = (((getMaxYValue() - getMinYValue()) / (clusters.length + 1)) * n) +
getMinYValue();
            Centroid c1 = new Centroid(cx, cy);
            clusters[n - 1].setCentroid(c1);
            c1.setCluster(clusters[n - 1]);
        }
    }

    private double getMaxXValue() {
        double temp;
        temp = ((DataPoint) mDataPoints.elementAt(0)).getX();
        for (int i = 0; i < mDataPoints.size(); i++) {
            DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
            temp = (dp.getX() > temp) ? dp.getX() : temp;
        }
        return temp;
    }

    private double getMinXValue() {
        double temp = 0;
        temp = ((DataPoint) mDataPoints.elementAt(0)).getX();
        for (int i = 0; i < mDataPoints.size(); i++) {
            DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
            temp = (dp.getX() < temp) ? dp.getX() : temp;
    }
    return temp;
}

private double getMaxYValue() {
    double temp = 0;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getY();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getY() > temp) ? dp.getY() : temp;
    }
    return temp;
}

private double getMinYValue() {
    double temp = 0;
    temp = ((DataPoint) mDataPoints.elementAt(0)).getY();
    for (int i = 0; i < mDataPoints.size(); i++) {
        DataPoint dp = (DataPoint) mDataPoints.elementAt(i);
        temp = (dp.getY() < temp) ? dp.getY() : temp;
    }
    return temp;
}

public int getKValue() {
    return clusters.length;
}
```

```
    public int getIterations() {
        return miter;
    }

    public int getTotalDataPoints() {
        return mDataPoints.size();
    }

    public double getSWCSS() {
        return mSWCSS;
    }

    public Cluster getCluster(int pos) {
        return clusters[pos];
    }
```

5.4 Centroid.java:

```
class Centroid {
    private double mCx, mCy;
    private Cluster mCluster;
```

```
public Centroid(double cx, double cy) {
  this.mCx = cx;
  this.mCy = cy;
}

public void calcCentroid() { //only called by CAInstance
  int numDP = mCluster.getNumDataPoints();
  double tempX = 0, tempY = 0;
  int i;
  //caluclating the new Centroid
  for (i = 0; i < numDP; i++) {
    tempX = tempX + mCluster.getDataPoint(i).getX();
    //total for x
    tempY = tempY + mCluster.getDataPoint(i).getY();
    //total for y
  }
  this.mCx = tempX / numDP;
  this.mCy = tempY / numDP;
  //calculating the new Euclidean Distance for each Data Point
  tempX = 0;
  tempY = 0;
  for (i = 0; i < numDP; i++) {
    mCluster.getDataPoint(i).calcEuclideanDistance();
  }
  //calculate the new Sum of Squares for the Cluster
  mCluster.calcSumOfSquares();
```

```
    }

    public void setCluster(Cluster c) {
        this.mCluster = c;
    }

    public double getCx() {
        return mCx;
    }

    public double getCy() {
        return mCy;
    }

    public Cluster getCluster() {
        return mCluster;
    }

}
```

5.5 Cluster.java:

```
import java.util.Vector;
class Cluster {    private String mName;
```

```
private Centroid mCentroid;
private double mSumSqr;
private Vector mDataPoints;

public Cluster(String name) {
    this.mName = name;
    this.mCentroid = null;
    mDataPoints = new Vector();
}

public void setCentroid(Centroid c) {
    mCentroid = c;
}

public Centroid getCentroid() {
    return mCentroid;
}

public void addDataPoint(DataPoint dp) {
    dp.setCluster(this); //initiates a inner call tocalcEuclideanDistance() in DP.
    this.mDataPoints.addElement(dp);
    calcSumOfSquares();
}

public void removeDataPoint(DataPoint dp) {
    this.mDataPoints.removeElement(dp);
    calcSumOfSquares();
    }

    public int getNumDataPoints() {
        return this.mDataPoints.size();
    }

    public DataPoint getDataPoint(int pos) {
        return (DataPoint) this.mDataPoints.elementAt(pos);
    }

    public void calcSumOfSquares() { //called from Centroid
        int size = this.mDataPoints.size();
        double temp = 0;
        for (int i = 0; i < size; i++) {
            temp = temp + ((DataPoint)
this.mDataPoints.elementAt(i)).getCurrentEuDt();
        }
        this.mSumSqr = temp;
    }

    public double getSumSqr() {
        return this.mSumSqr;
    }

    public String getName() {
        return this.mName;
    }
    public Vector getDataPoints() {
        return this.mDataPoints;
    }

}
```

5.6 Location.java:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * customer.java
 *
 * Created on 8 May, 2011, 8:37:28 PM
 */

/**
 *
 * @author K
 */
import java.io.FileNotFoundException;
import java.io.IOException;
import java.lang.System;
import java.util.logging.Level;
import java.util.logging.Logger;
public class customer extends javax.swing.JFrame {

  /** Creates new form customer */
  public customer() {
    initComponents();
  }

  /** This method is called from within the constructor to
   * initialize the form.
   * WARNING: Do NOT modify this code. The content of this method is
   * always regenerated by the Form Editor.
   */
  @SuppressWarnings("unchecked")
  //    <editor-fold    defaultstate="collapsed"    desc="Generated    Code">//GEN-
BEGIN:initComponents
  private void initComponents() {

    jPanel1 = new javax.swing.JPanel();
    c1 = new javax.swing.JCheckBox();
    c2 = new javax.swing.JCheckBox();
    c3 = new javax.swing.JCheckBox();
    c4 = new javax.swing.JCheckBox();
    c5 = new javax.swing.JCheckBox();
    c6 = new javax.swing.JCheckBox();
```

```
c7 = new javax.swing.JCheckBox();
c8 = new javax.swing.JCheckBox();
c9 = new javax.swing.JCheckBox();
c10 = new javax.swing.JCheckBox();
c11 = new javax.swing.JCheckBox();
c12 = new javax.swing.JCheckBox();
c13 = new javax.swing.JCheckBox();
c14 = new javax.swing.JCheckBox();
c15 = new javax.swing.JCheckBox();
c16 = new javax.swing.JCheckBox();
c17 = new javax.swing.JCheckBox();
c18 = new javax.swing.JCheckBox();
c19 = new javax.swing.JCheckBox();
c20 = new javax.swing.JCheckBox();
c21 = new javax.swing.JCheckBox();
c22 = new javax.swing.JCheckBox();
c23 = new javax.swing.JCheckBox();
c24 = new javax.swing.JCheckBox();
submit = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

c1.setText("head ache");
c1.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    c1ActionPerformed(evt);
  }
```

```
});

c2.setText("body ache");
c2.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      c2ActionPerformed(evt);
   }
});

c3.setText("stomach pain");
c3.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      c3ActionPerformed(evt);
   }
});

c4.setText("sneezes");
c4.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      c4ActionPerformed(evt);
   }
});

c5.setText("shivering");
c5.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      c5ActionPerformed(evt);
```

```
    }
});

c6.setText("eye itching");

c7.setText("joint pain");
c7.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        c7ActionPerformed(evt);
    }
});

c8.setText("yellowish nail and eye");
c8.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        c8ActionPerformed(evt);
    }
});

c9.setText("fever");
c9.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        c9ActionPerformed(evt);
    }
});

c10.setText("urinal pain");
```

```
c11.setText("muscle pain");
c11.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    c11ActionPerformed(evt);
  }
});

c12.setText("chest pain");
c12.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    c12ActionPerformed(evt);
  }
});

c13.setText("jCheckBox13");

c14.setText("jCheckBox14");

c15.setText("jCheckBox15");

c16.setText("jCheckBox16");

c17.setText("jCheckBox17");

c18.setText("jCheckBox18");
```

```
c19.setText("jCheckBox19");
c19.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      c19ActionPerformed(evt);
   }
});

c20.setText("jCheckBox20");

c21.setText("jCheckBox21");

c22.setText("jCheckBox22");

c23.setText("jCheckBox23");

c24.setText("jCheckBox24");

submit.setText("submit");
submit.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      submitActionPerformed(evt);
   }
});

javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(
        jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel1Layout.createSequentialGroup()
            .addContainerGap()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(c1)
                        .addComponent(c2)
                        .addComponent(c3)
                        .addComponent(c4)
                        .addComponent(c5)
                        .addComponent(c6)
                        .addComponent(c7)
                        .addComponent(c9)
                        .addComponent(c10)
                        .addComponent(c11)
                        .addComponent(c12))
                    .addGap(27, 27, 27)
                    .addComponent(submit))
                .addComponent(c8))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,        32,
Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(c13)
                .addComponent(c14)
                .addComponent(c15)
```

```
                    .addComponent(c16)
                    .addComponent(c17)
                    .addComponent(c18)
                    .addComponent(c19)
                    .addComponent(c20)
                    .addComponent(c21)
                    .addComponent(c22)
                    .addComponent(c23)
                    .addComponent(c24))
                .addContainerGap(260, Short.MAX_VALUE))
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                    .addComponent(c1)
                    .addComponent(c13))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                    .addComponent(c2)
                    .addComponent(c14))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
```

```
            .addComponent(c3)
            .addComponent(c15))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addComponent(c4)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c5)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c6)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c7)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c8,        javax.swing.GroupLayout.PREFERRED_SIZE,        23,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c9)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c10)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c11)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c12))
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addComponent(c16)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c17)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c18)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c19)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c20)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c21)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c22)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c23)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c24)))
            .addContainerGap(17, Short.MAX_VALUE))
          .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
            .addContainerGap(277, Short.MAX_VALUE)
            .addComponent(submit))
    );


    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addComponent(jPanel1,                    javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    );
```

```
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1,                          javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents
int a[]=new int[8],i=0;
    private        void        c1ActionPerformed(java.awt.event.ActionEvent        evt)        {//GEN-
FIRST:event_c1ActionPerformed
        // TODO add your handling code here:
}//GEN-LAST:event_c1ActionPerformed

    private        void        c2ActionPerformed(java.awt.event.ActionEvent        evt)        {//GEN-
FIRST:event_c2ActionPerformed
        // TODO add your handling code here:
}//GEN-LAST:event_c2ActionPerformed

    private        void        c4ActionPerformed(java.awt.event.ActionEvent        evt)        {//GEN-
FIRST:event_c4ActionPerformed
        // TODO add your handling code here:
}//GEN-LAST:event_c4ActionPerformed

    private        void        c5ActionPerformed(java.awt.event.ActionEvent        evt)        {//GEN-
FIRST:event_c5ActionPerformed
        // TODO add your handling code here:
}//GEN-LAST:event_c5ActionPerformed
```

```
    private        void        c7ActionPerformed(java.awt.event.ActionEvent        evt)        {//GEN-
FIRST:event_c7ActionPerformed

        // TODO add your handling code here:

}//GEN-LAST:event_c7ActionPerformed


    private        void        c8ActionPerformed(java.awt.event.ActionEvent        evt)        {//GEN-
FIRST:event_c8ActionPerformed

        // TODO add your handling code here:

}//GEN-LAST:event_c8ActionPerformed


    private        void        c9ActionPerformed(java.awt.event.ActionEvent        evt)        {//GEN-
FIRST:event_c9ActionPerformed

        // TODO add your handling code here:

}//GEN-LAST:event_c9ActionPerformed


    private        void        c11ActionPerformed(java.awt.event.ActionEvent        evt)        {//GEN-
FIRST:event_c11ActionPerformed

        // TODO add your handling code here:

}//GEN-LAST:event_c11ActionPerformed


    private        void        c19ActionPerformed(java.awt.event.ActionEvent        evt)        {//GEN-
FIRST:event_c19ActionPerformed

        // TODO add your handling code here:

}//GEN-LAST:event_c19ActionPerformed


    private        void        c12ActionPerformed(java.awt.event.ActionEvent        evt)        {//GEN-
FIRST:event_c12ActionPerformed

        // TODO add your handling code here:

}//GEN-LAST:event_c12ActionPerformed
```

```
  private void c3ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_c3ActionPerformed

      // TODO add your handling code here:

}//GEN-LAST:event_c3ActionPerformed


  private void submitActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_submitActionPerformed

new customer().setVisible(false);


      if(c1.isSelected())

{


   {

   a[i]=1;

   }

   i++;

}//


if(c2.isSelected())

{


   {

   a[i++]=2;

   }

}

if(c3.isSelected())

{
    {
    a[i++]=3;
    }
 }// T
 if(c4.isSelected())
 {

    {
    a[i++]=4;
    }
 }// T
 if(c5.isSelected())
 {

    {
    a[i++]=5;
    }
 }// T
 if(c6.isSelected())
 {

    {
    a[i++]=6;
    }
 }// T
 if(c7.isSelected())
 {
    {
    a[i++]=7;
    }
}// T
if(c8.isSelected())
{

    {
    a[i++]=8;
    }
}// T
if(c9.isSelected())
{

    {
    a[i++]=9;
    }
}// T
if(c10.isSelected())
{

    {
    a[i++]=10;
    }
}// T
if(c11.isSelected())
```

```
{

  {
  a[i++]=11;
  }
}// T
if(c12.isSelected())
{

  {
  a[i++]=12;
  }
}// T
if(c13.isSelected())
{

  {
  a[i++]=13;
  }
}// T
if(c14.isSelected())
{

  {
  a[i++]=14;
  }
}// T
```

```
if(c15.isSelected())
{

   {
   a[i++]=15;
   }
}
if(c16.isSelected())
{

   {
   a[i++]=16;
   }
}//
if(c17.isSelected())
{

   {
   a[i++]=17;
   }
}//
if(c18.isSelected())
{

   {
   a[i++]=18;
   }
```

```
}//
if(c19.isSelected())
{

  {
  a[i++]=19;
  }
}//
if(c20.isSelected())
{
   i++;
  {
  a[i]=20;
  }
}//
if(c21.isSelected())
{

  {
  a[i++]=21;
  }
}//
if(c22.isSelected())
{

  {
  a[i++]=22;
```

```
   }
}//
if(c23.isSelected())
{

   {
   a[i++]=23;
   }
}//
if(c24.isSelected())
{

   {
   a[i++]=24;
   }
}//

      try {
         //
         choice ch = new choice(a, i);
      } catch (FileNotFoundException ex) {
         Logger.getLogger(customer.class.getName()).log(Level.SEVERE, null, ex);
      } catch (IOException ex) {
         Logger.getLogger(customer.class.getName()).log(Level.SEVERE, null, ex);
      }
}//GEN-LAST:event_submitActionPerformed
```

```
/**
  *  * @param args the command line arguments
  */



  // Variables declaration - do not modify//GEN-BEGIN:variables
  private javax.swing.JCheckBox c1;
  private javax.swing.JCheckBox c10;
  private javax.swing.JCheckBox c11;
  private javax.swing.JCheckBox c12;
  private javax.swing.JCheckBox c13;
  private javax.swing.JCheckBox c14;
  private javax.swing.JCheckBox c15;
  private javax.swing.JCheckBox c16;
  private javax.swing.JCheckBox c17;
  private javax.swing.JCheckBox c18;
  private javax.swing.JCheckBox c19;
  private javax.swing.JCheckBox c2;
  private javax.swing.JCheckBox c20;
  private javax.swing.JCheckBox c21;
  private javax.swing.JCheckBox c22;
  private javax.swing.JCheckBox c23;
  private javax.swing.JCheckBox c24;
  private javax.swing.JCheckBox c3;
  private javax.swing.JCheckBox c4;
  private javax.swing.JCheckBox c5;
```

```
    private javax.swing.JCheckBox c6;
    private javax.swing.JCheckBox c7;
    private javax.swing.JCheckBox c8;
    private javax.swing.JCheckBox c9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JButton submit;
    // End of variables declaration//GEN-END:variables
}
```

Chapter 6: TESTING

6.1 Testing

Testing is the process of executing the program(s) with the intention of finding out errors. During testing, the program to be tested is executed with a set of test cases and the output of the programs for the test case is evaluated to determine if the program is performing as it is expected to be. The success of testing in revealing errors in programs depends critically on the test cases

6.1.1 LEVELS OF TESTING

UNIT TESTING: The first level of testing is called unit testing. In this different modules are tested against the specifications produced during design of the modules. Unit testing is essentially for verification of the code produced during coding phase, and hence the goal is to test the internal logic of the modules. The programmer of the module typically does it.

INTEGRATION TESTING: The next level of testing is often called integration testing. In this, many unit-tested modules are combined into subsystems, which are then tested. The goal is here to see if the modules can be integrated properly. Hence, the emphasis is on testing interfaces between modules. The testing activity can be considered testing the design. The integration plan specifies the steps and order in which modules are combined to realize the full system. After each integration step, the partially integrated system is tested. An important factor that guides the integration is the module dependency graph.

SYSTEM TESTING: System tests are designed to validate a fully developed system to assure that it meets its requirements. There are essentially three main kinds of system testing:

ALPHA TESTING: Alpha refers to the system testing carried out by the test team within the developing organization.

BETA TESTING: Beta testing is the system testing performed by a select group of friendly customers.

ACCEPTANCE TESTING: Acceptance testing is the system testing performed by the customer to determine whether to accept or reject the delivery of the system.

6.1.2 TYPES OF TESTING:

BLACK BOX TESTING: This testing is also known as functional testing. The basis for deciding test cases in functional testing is the requirements or specifications of the system or modules. For the entire system test cases are designed form the requirement specification document from the system. There are number of techniques that can be used to select test cases that have been found to be very successful in detecting errors.Some of them are:

<u>Equivalence class partitioning:</u> In this we divide the domain of all the inputs into a set of equivalence classes. That is we want to identify classes of test cases such that the success of one test case in a class implies the success of others. It is often useful to consider equivalence classes in the output. For an output equivalence class, the goal is to generate test cases such that the output of that test case lies in the output equivalence class.

<u>Boundary value analysis:</u> In boundary value analysis, we choose an input for a test case from an equivalence class, such that the input lies at the edge of the equivalence class. Boundary value rest cases are also called "extreme cases".

<u>Cause-effect graphing:</u> It is a technique that aids in selecting combinations of input conditions in a systematic way. A cause is a distinct input condition, and an effect is n distinct output condition. Each condition forms a node in the cause-effect graph. Beyond generating high-yield test cases, it also aids the understanding of the functionality of the system, because the tester must identify the distinct causes and effects.

<u>Special cases:</u> It depends on the data structures and the function of the module. There are no rules to determine special cases, and the tester has to use his intuition and experience to identify such test cases. Consequently, determining special cases is also called "error guessing.

WHITE BOX TESTING: There are several white box-testing strategies. Each testing strategy is based on some heuristic. One white box testing strategy is said to be stronger than another strategy, if all types of errors detected by the first testing strategy (say B) are also detected by the second testing strategy (say A), and the second strategy additionally detects some more types of errors. When two testing strategies detect errors that are different at least with respect to some types of errors, they are then called complementary.

<u>Statement coverage:</u> It aims to design test cases so that every statement in the program is executed at least once. The principal idea is that unless we execute a statement, we have no way of determining if error exists in that statement.

Branch coverage: In this strategy, test cases are designed to make each branch condition assume true and false value in turn. It is also known as "edge-testing" as in this testing scheme, each edge of a program's control flow graph is traversed at least once.

Condition coverage: In this test cases are designed to make each component of a composite conditional expression assume both true and false values. Thus, condition testing is a stronger testing strategy than branch testing. For conditional coverage, the number of test cases increases exponentially with the number of component conditions.

Path coverage: It requires us to design test cases such that all linearly independent paths in the program are executed at least once. These paths are defined in terms of control-flow graph of a program.

As testing forms the first step towards determining the errors in a program it should be properly carried out.

UNIT TESTING:

Unit testing was carried out for each module against the specifications produced during the design of the module.

Unit testing of each program and module was done with the following perception.

USER INTERFACE:

User interface was tested which gave rise to more user understandable errors and help messages.

INTERNAL LOGIC:

While testing a module, the internal logic was tested.

INTEGRATED TESTING:

Tint the integrated testing, the unit-tested modules were combined into subsystems and then tested.

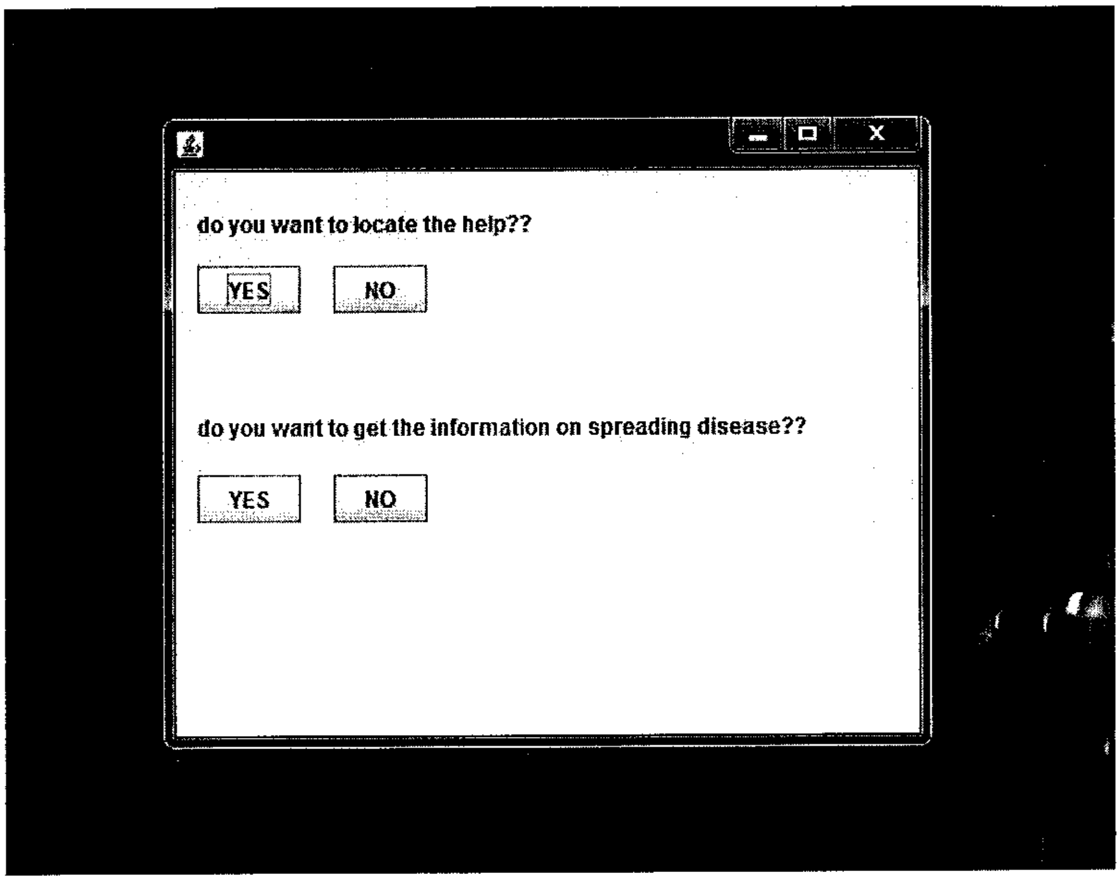
The strategies for integrated tested comprised of :

- Performance time testing.
- Logical cycle of data.
- Test data.
- Live data obtained from users.

6.1.3.3 Environments

The environment on which we have performed the firewall configuration and testing was a linux workstation that has virtual network of virtual computers that are managed by the MLN tool. The network schema is described in the lab document. The tests are done using external and internal UML instances.

Chapter 7: Results and Analysis


do you want to locate the help??

do you want to get the information on spreading disease??

Fig. Initial.gui

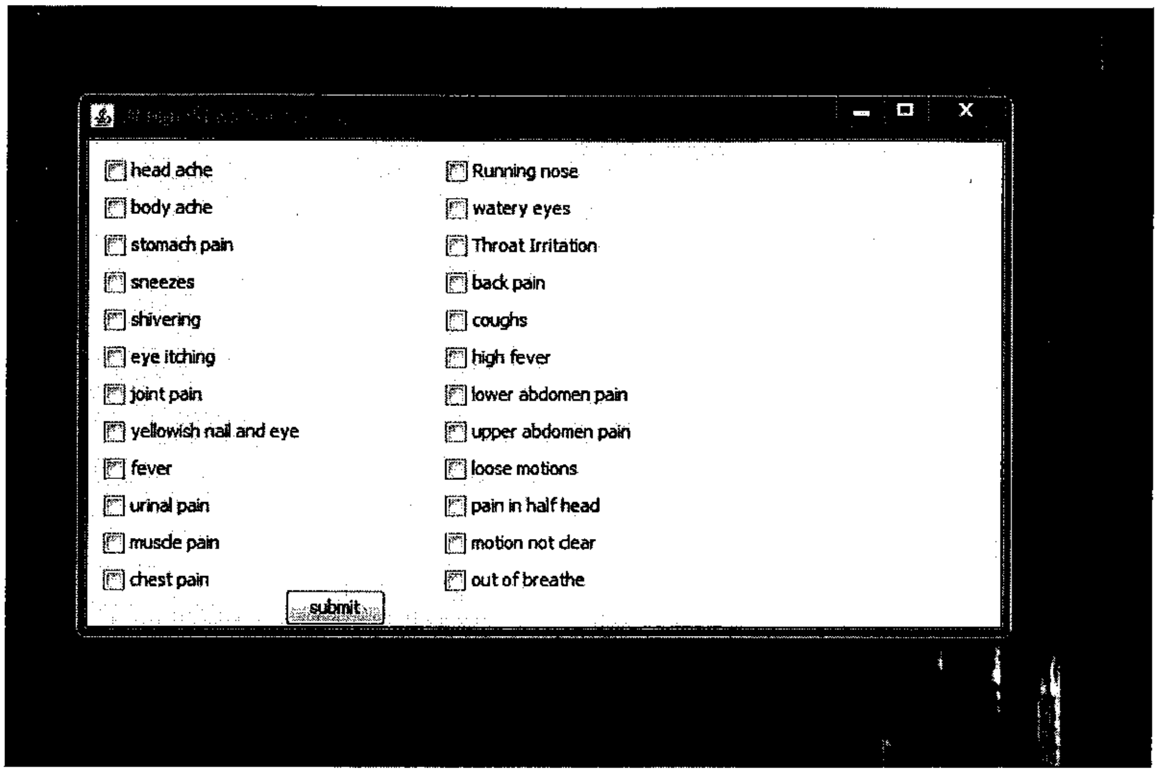


Fig: Customer.gui

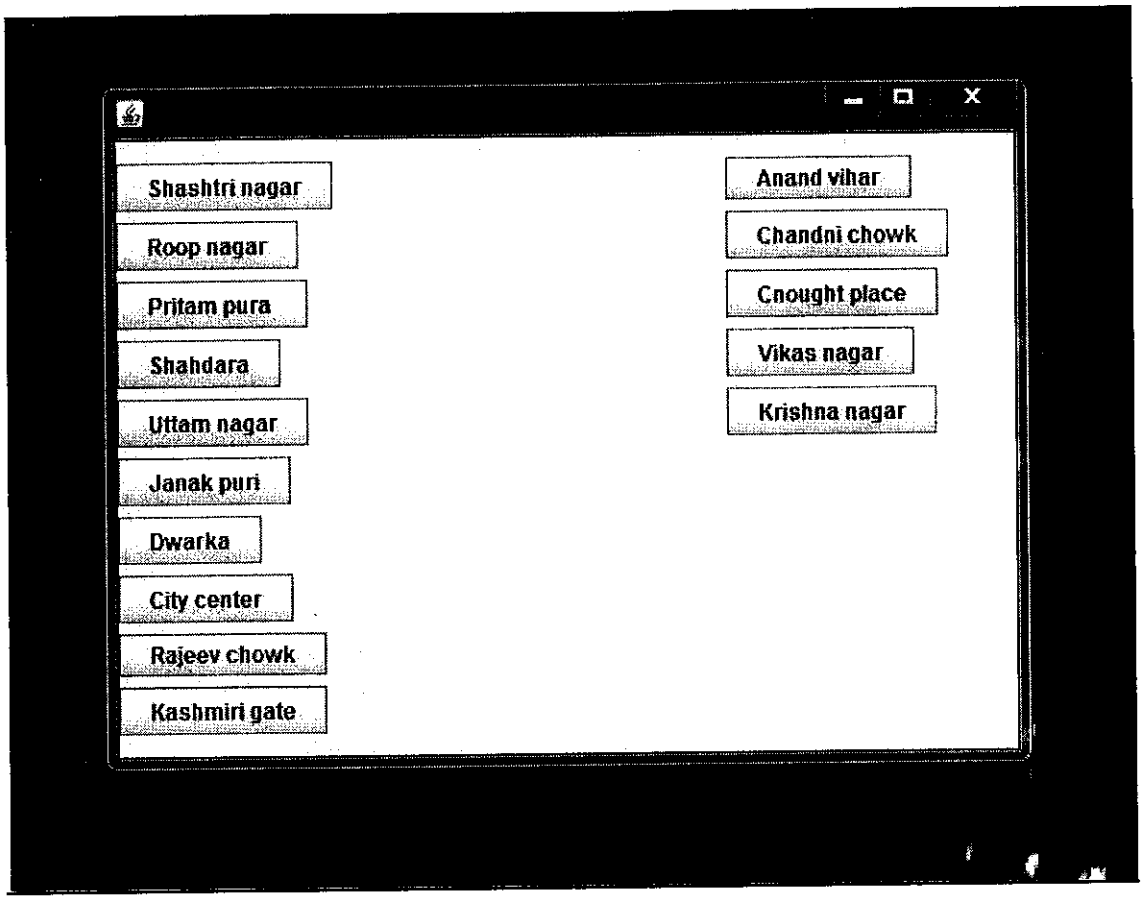


Fig : Location.gui

From the project we came to a conclusion that the person who suffer from a particular disease they enter the symptoms they are facing after getting the symptoms from the patient we tell them the probable disease he may be facing from and on the basis of their location id . We divide them into clusters and tell them the nearest possible hospital he can visit. Then on the basis of location id to we can find out the area which is prone to a particular disease.

Chapter 8 : OBJECTIVE AND SCOPE

This project aims to automate the process of finding solutions to some health related problems spreading all over the world.

Objectives:

- Provide the system that automates the process of health surveys in different part of the world
- Optimal solutions for the various heath related problems.
- Prevent the myths to spread in the field of medicine.

The Scope of the project is to overcome the exhaustive process of surveying for different health related problems to find out the spreading areas ,symptoms of various diseases people can get the specialise health centre location near their home for their health problem.

CHAPTER 9: REFERENCES

- Distributed Data Mining by Haimanty Dutta , 2007.
- Data Clustering by A.K Jain, M.N Murthy and P.J Flynn, 2000.
- Data Mining Concepts and Techniques By Jiawei Han and Micheline Kamber
- Data Mining and Data Analysis For Counter Terrorism By MARY DEROSA.
- *Data Mining* Methods and Models Information Retrieval by Larose, Daniel T.
- Research paper by Olivier Chapell ,Max Planck Institute for Biological Cybernetics ,september 2008.
- Secure distributed data-mining and its application to large-scale network measurements, Matthew Roughan and yin zhang ,January 2006.