MINOR PROJECT

Effective analysis and diagnosis of Liver disorder using clustering and association rules in data mining

Submitted in partial fulfilment of the
Requirement for the award of
Degree of Bachelor of Technology
In
Computer Engineering
(Batch : 2014-2018)

Under the Supervision of

Ms. Indu Singh
(Assistant Professor,CSE)

By:

Ajit Singh Kushwaha(2K14/CO/006)

Bablu Singh(2K14/CO/020)

Tabrez Alam(2K14/CO/131)

To:

Department of Computer Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road, Delhi-110042

DECLARATION

I hearby certify that the work which is presented in the Minor Project entitled " **Effective analysis and diagnosis of Liver disorder using clustering and association rules in data mining** " in fulfillment of the requirement for the award of Degree of Bachelor of Technology and submitted to the Department of Computer Engineering, Delhi Technological University (Formerly Delhi College of Engineering), New Delhi is an authentic record of my own, carried out during a period from August 2016 to November 2016, under the supervision of **Ms. Indu Singh (Assistant Professor, CSE Department)**.

The matter presented in this report has not been submitted by me for the award of any other degree of this or any other Institute/University.

Signature

Ajit Singh Kushwaha (2K14/CO/006)

Bablu Singh (2K14/CO/020)

Tabrez Alam (2K14/CO/131)

SUPERVISOR CERTIFICATE

This is to certify that **AJIT SINGH KUSHWAHA(2K14/CO/006)**, **BABLU SINGH(2K14/CO/020)** and **TABREZ ALAM(2K14/CO/131)** the bonafide students of **Bachelor of Technology in Computer Engineering** of **Delhi Technological Univerity** (Formerly Delhi College of Engineering), New Delhi of **2014-2018** batch has completed their minor project entitled "**Effective analysis and diagnosis of Liver disorder using clustering and association rules in data mining** " under the supervision of **Ms. Indu Singh** (Assistant Professor, CSE Department).

It is further certified that the work done in this dissertation is a result of candidate's own efforts.

I wish for all success in their life.

Date :

Ms. Indu Singh

Assistant Professor

CSE Department

Delhi Technological University

(Feormerly Delhi College of Engineering)

Shahbad Daulatpur, Bawana Road, Delhi - 110042

ABSTRACT

There are many disorders of the liver that require clinical care by a physician or other healthcare professional. In this project we will present classification mode of liver patient dataset in order to predict liver diagnosis. Medical data mining is one of the critical aspect of automated disease diagnosis and disease prediction. Medical Data Diagnosis involves developing data mining algorithms and techniques to analyse medical data. In recent years, liver disorders have excessively increased and liver disease are becoming one of the most fatal diseases in several countries.

In this study, one real patient dataset was investigated for building classification models in order to predict liver diagnosis. Five data mining classification algorithms were applied to the dataset and the performance of all classifiers are compared against each other in terms of accuracy, precision, and recall. Several investigations have also been carried out to improve performance of the classification models. Finally, the results shown promising methodology in diagnosing liver disease during the earlier stages.

This data set contains 416 liver patient records and 167 non liver patient records. The data set was collected from north east of Andhra Pradesh, India. **Selector.field** is a class label used to divide into groups(liver patient or not).This data set contains 441 male patient records and 142 female patient records. Any patient whose age exceeded 89 is listed as being of age "90".

In this Dataset Selector field is made as a factor field on which all the algorithms make their decisions. Every dataset contains atleast one factor field which have some levels in this dataset levels are two in numbers. 1stands for the liver patient and 2 for the non-liver patient.

INDEX

Introduction

1.1 Objective

Our objective is to "**Analyse the liver patient dataset and predict accuracy on applying data mining rule.**"

There are many disorders of the liver that require clinical care by a physician or other healthcare professional. In this project we will present classification mode of liver patient dataset in order to predict liver diagnosis. Medical data mining is one of the critical aspect of automated disease diagnosis and disease prediction. Medical Data Diagnosis involves developing data mining algorithms and techniques to analyze medical data. In recent years, liver disorders have excessively increased and liver disease are becoming one of the most fatal diseases in several countries.

In this study, one real patient dataset was investigated for building classification models in order to predict liver diagnosis. Five data mining classification algorithms were applied to the dataset and the performance of all classifiers are compared against each other in terms of accuracy, precision, and recall. Several investigations have also been carried out to improve performance of the classification models. Finally, the results shown promising methodology in diagnosing liver disease during the earlier stages.

This data set contains 416 liver patient records and 167 non liver patient records. The data set was collected from north east of Andhra Pradesh, India. Selector.field is a class label used to divide into groups(liver patient or not).This data set contains 441 male patient records and 142 female patient records. Any patient whose age exceeded 89 is listed as being of age "90". It contains 11 attributes which are :

1. Age — Age of the patient
2. Gender — Gender of the patient
3. TB — Total Bilirubin
4. DB — Direct Bilirubin
5. Alkphos — Alkaline Phosphotase
6. Sgpt Alamine — Aminotransferase
7. Sgot — Aspartate Aminotransferase

8.	TP	Total Protiens
9.	ALB	Albumin
10.	A/G Ratio	Albumin and Globulin Ratio
11.	Selector field	used to split the data into two sets (labeled by the experts).

To create the predictions, this liver patient's dataset is further devided into two parts i.e. training data and the testing data. According to each Algorithm. This division is done for each algorithm to remove the redundancy of overlapping of results, which causes wrong results.

Chapter 2

Data Mining

Database technology has evolved from primitive file processing to the development of database management systems with query and transaction processing. Further progress has led to the increasing demand for efficient and effective advanced data analysis tools. This need is a result of the explosive growth in data collected from applications, including business and management, government administration, science and engineering, and environmental control. Data mining has attracted a great deal of attention in the information industry and in society as a whole in recent years, due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information and knowledge. The information and knowledge gained can be used for applications ranging from market analysis, fraud detection, and customer retention, to production control and science exploration.[1]

2.1 What Is Data Mining?

The term *data mining* refers to extracting or "mining" knowledge from large amounts of data. Data mining is an essential step in the process of knowledge discovery. [1]]

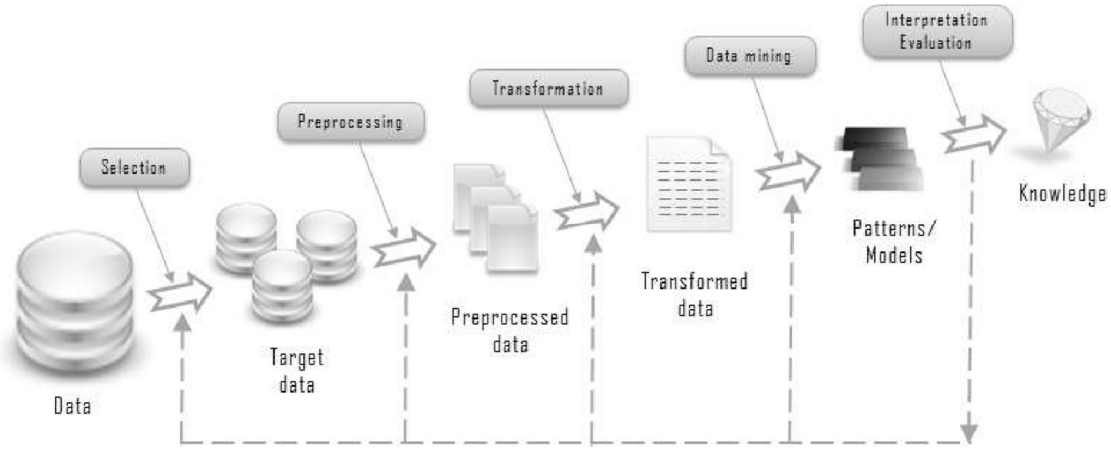


Figure 2.1: Data mining as a step in the process of knowledge discovery.[1]

Knowledge discovery as a process is depicted in Figure 3.1 and consists of an iterative sequence of the following steps:

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

Steps 1 to 4 are different forms of data preprocessing, where the data are prepared for mining.[1]

2.2 Data Mining Functionalities—What Kinds of Patterns Can Be Mined?

Data mining functionalities include the discovery of concept/class descriptions, associations and correlations, classification, prediction, clustering, trend analysis, outlier and deviation analysis, and similarity analysis. Data mining is the task of discovering interesting patterns from large amounts of data, where the data can be stored in databases, data warehouses, or other information repositories. A *pattern* represents knowledge if it is easily understood by humans; valid on test data with some degree of certainty; and potentially useful, novel, or validates a hunch about which the user was curious. Measures of pattern interestingness, either objective or subjective, can be used to guide the discovery process. In general, data mining tasks can be classified into two categories: descriptive and predictive. *Descriptive mining* tasks characterize the general properties of the data in the database. *Predictive mining* tasks perform inference on the current data in order to make predictions.[1]

2.2.1 Classification and Prediction

Classification is the process of finding a model (or function) that describes and distinguishes data classes or concepts, for the purpose 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).

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 easily be converted to classification rules. A neural network, when used for classification, is typically a collection of neuron-like processing units with weighted connections between the units. There are many other methods for constructing classification models, such as naïve Bayesian classification, support vector machines, and k-nearest neighbor classification.

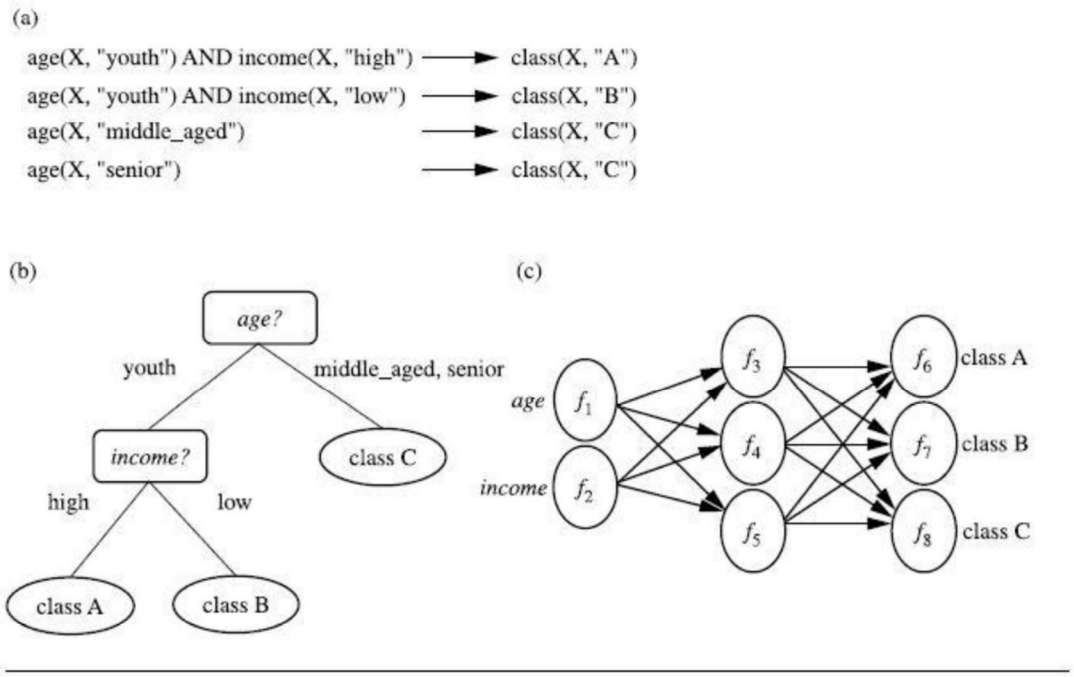


Figure 2.2: A classification model can be represented in various forms, such as (a) IF-THEN rules, (b) a decision tree, or a (c) neural network.[1]

Whereas classification predicts categorical (discrete, unordered) labels, prediction models continuous-valued functions. That is, it is used to predict missing or unavailable numerical data values rather than class labels.

Regression analysis is a statistical methodology that is most often used for numeric prediction, although other methods exist as well. Prediction also encompasses the identification of distribution trends based on the available data. Classification and prediction may need to be preceded by relevance analysis, which attempts to identify attributes that do not contribute to the classification or prediction process.[1]

2.2.2 Cluster Analysis

"What is cluster analysis?" Unlike classification and prediction, which analyze class-labeled data objects, clustering analyzes data objects without consulting a known class label. In general, the class labels are not present in the training data simply because they are not known to begin with. Clustering can be used to generate such labels. The objects are clustered or grouped based on the principle of maximizing the intra class similarity and minimizing the

interclass similarity. That is, clusters of objects are formed so that objects within a cluster have high similarity in comparison to one another, but are very dissimilar to objects in other clusters. Each cluster that is formed can be viewed as a class of objects, from which rules can be derived. Clustering can also facilitate taxonomy formation, that is, the organization of observations into a hierarchy of classes that group similar events together.[1]

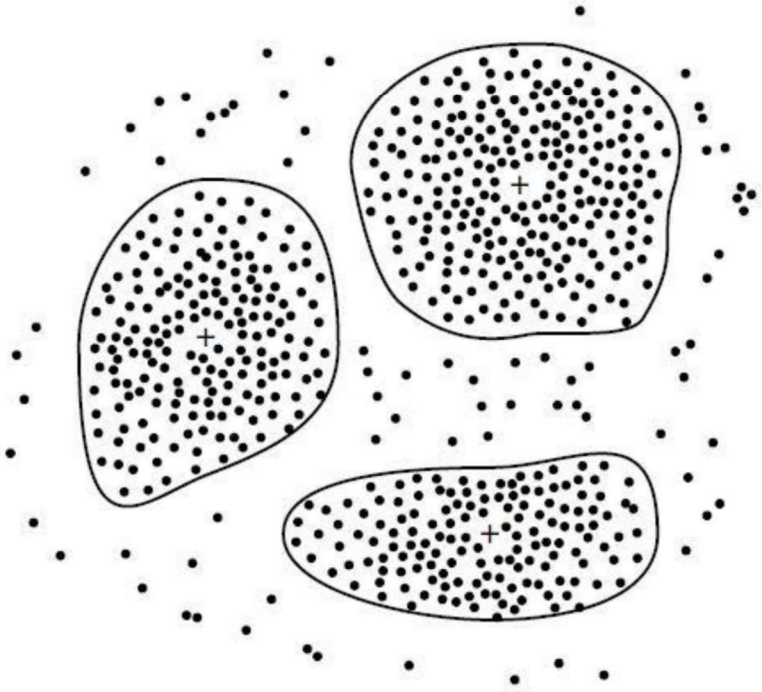


Figure 2.3: A 2-D plot, showing three data clusters. Each cluster "center" is marked with a "+".[1]

2.3 Are All of the Patterns Interesting?

A data mining system has the potential to generate thousands or even millions of patterns, or rules. This raises some serious questions for data mining. First, "are all of the patterns interesting?" Typically not — a pattern is interesting if it is (1) easily understood by humans, (2) valid on new or test data with some degree of certainty, (3) potentially useful, and (4) novel. An interesting pattern represents knowledge. Several objective measures of pattern interestingness exist.

Secondly —"Can a data mining system generate all of the interesting patterns?"— refers to the completeness of a data mining algorithm. It is often unrealistic and inefficient for data mining systems to generate all of the possible patterns. Instead, user-provided constraints and interestingness measures should be used to focus the search. For some mining tasks, such as association, this is often sufficient to ensure the completeness of the algorithm.

Finally, the third question—"Can a data mining system generate only interesting patterns?"—is an optimization problem in data mining. It is highly desirable for data mining systems to generate only interesting patterns. This would be much more efficient for users and data mining systems, because neither would have to search through the patterns generated in order to identify the truly interesting ones.[1]

2.4 Classification of Data Mining Systems

It is a young interdisciplinary field, drawing from areas such as database systems, data warehousing, statistics, machine learning, data visualization, information retrieval, and high-performance computing. Other contributing areas include neural networks, pattern recognition, spatial data analysis, image databases, signal processing, and many application fields, such as business, economics, and bioinformatics.[1]

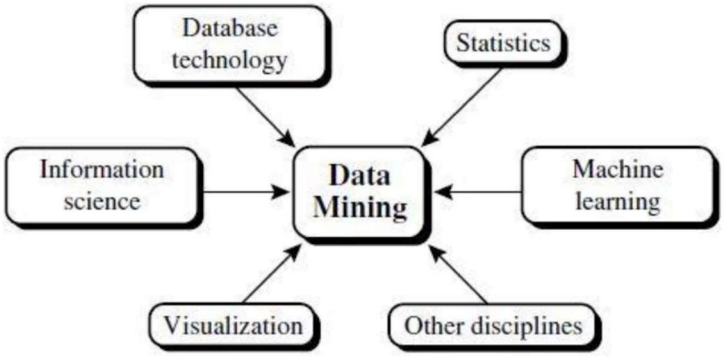


Figure 2.4: Data mining as a confluence of multiple disciplines.[1]

Data mining systems can be categorized according to various criteria, as follows:[1]

- *Classification according to the kinds of databases mined:* If classifying according to data models, we may have a relational, transactional, object-relational, or data warehouse mining system. If classifying according to the special types of data handled, we may have a spatial, time-series, text, stream data, multimedia data mining system, or a World Wide Web mining system.
- *Classification according to the kinds of knowledge mined:* Data mining systems can be categorized according to the kinds of knowledge they mine, that is, based on data mining functionalities, such as characterization, discrimination, association and correlation analysis, classification, prediction, clustering, outlier analysis, and evolution analysis.
- *Classification according to the kinds of techniques utilized:* Data mining systems can be categorized according to the underlying data mining techniques employed. These techniques can be described according to the degree of user interaction involved (e.g., autonomous systems, interactive exploratory systems, query-driven systems) or the methods of data analysis employed (e.g., database-oriented or data warehouse–oriented

techniques, machine learning, statistics, visualization, pattern recognition, neural networks, and so on).

- *Classification according to the applications adapted:* Data mining systems can also be categorized according to the applications they adapt. For example, data mining systems may be tailored specifically for finance, telecommunications, DNA, stock markets, e-mail, and so on.

Chapter 3

Data Preprocessing

Today's real-world databases are highly susceptible to noisy, missing, and inconsistent data due to their typically huge size (often several gigabytes or more) and their likely origin from multiple, heterogeneous sources. Low-quality data will lead to low-quality mining results.

"How can the data be preprocessed in order to help improve the quality of the data and, consequently, of the mining results? How can the data be preprocessed so as to improve the efficiency and ease of the mining process?"

There are a number of data preprocessing techniques. *Data cleaning* can be applied to remove noise and correct inconsistencies in the data. *Data integration* merges data from multiple sources into a coherent data store, such as a data warehouse. *Data transformations*, such as normalization, may be applied. For example, normalization may improve the accuracy and efficiency of mining algorithms involving distance measurements. *Data reduction* can reduce the data size by aggregating, eliminating redundant features, or clustering, for instance. These techniques are not mutually exclusive; they may work together. For example, data cleaning can involve transformations to correct wrong data, such as by transforming all entries for a *date* field to a common format. Data processing techniques, when applied before mining, can substantially improve the overall quality of the patterns mined and/or the time required for the actual mining.[1]

3.1 Why Preprocess the Data?

The data you wish to analyse by data mining techniques are incomplete (lacking attribute values or certain attributes of interest, or containing only aggregate data), noisy (containing errors, or outlier values that deviate from the expected), and inconsistent (e.g., containing discrepancies in the department codes used to categorize items).

Incomplete, noisy, and inconsistent data are commonplace properties of large real world databases and data warehouses. Incomplete data can occur for a number of reasons. Attributes of interest may not always be available, such as customer information for sales transaction data. Other data may not be included simply because it was not considered important at the time of entry. Relevant data may not be recorded due to a misunderstanding,

or because of equipment malfunctions. Data that were inconsistent with other recorded data may have been deleted. Furthermore, the recording of the history or modifications to the data may have been overlooked. Missing data, particularly for tuples with missing values for some attributes, may need to be inferred.

There are many possible reasons for noisy data (having incorrect attribute values). The data collection instruments used may be faulty. There may have been human or computer errors occurring at data entry. Errors in data transmission can also occur. There may be technology limitations, such as limited buffer size for coordinating synchronized data transfer and consumption. Incorrect data may also result from inconsistencies in naming conventions or data codes used, or inconsistent formats for input fields, such as date. Duplicate tuples also require data cleaning.[1]

3.2 Descriptive Data Summarization

For data preprocessing to be successful, it is essential to have an overall picture of your data. Descriptive data summarization techniques can be used to identify the typical properties of your data and highlight which data values should be treated as noise or outliers. Thus, we first introduce the basic concepts of descriptive data summarization before getting into the concrete workings of data preprocessing techniques. For many data preprocessing tasks, users would like to learn about data characteristics regarding both central tendency and dispersion of the data. Measures of central tendency include mean, median, mode, and midrange, while measures of data dispersion include quartiles, interquartile range (IQR), and variance. These descriptive statistics are of great help in understanding the distribution of the data.[1]

3.2.1 Measuring the Central Tendency

The most common and most effective numerical measure of the "center" of a set of data is the *(arithmetic) mean*. Let $x_1, x_2, \ldots, x_N$ be a set of N values or observations. The mean of this set of values is

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} = \frac{x_1 + x_2 + \cdots + x_N}{N}.$$

Although the mean is the single most useful quantity for describing a data set, it is not always the best way of measuring the center of the data. A major problem with the *mean* is its sensitivity to extreme (e.g., outlier) values. Even a small number of extreme values can corrupt the mean. For example, the mean salary at a company may be substantially pushed up by that of a few highly paid managers. Similarly, the average score of a class in an exam could be pulled down quite a bit by a few very low scores. To offset the effect caused by a small number

of extreme values, we can instead use the trimmed mean, which is the mean obtained after chopping off values at the high and low extremes. For example, we can sort the values observed for *salary* and remove the top and bottom 2% before computing the mean. We should avoid trimming too large a portion (such as 20%) at both ends as this can result in the loss of valuable information.

For skewed (asymmetric) data, a better measure of the center of data is the *median*. Suppose that a given data set of N distinct values is sorted in numerical order. If N is odd, then the median is the *middle value* of the ordered set; otherwise (i.e., if N is even), the median is the average of the middle two values.

We can, however, easily *approximate* the median value of a data set. Assume that data are grouped in intervals according to their x_i data values and that the frequency (i.e., number of data values) of each interval is known. Let the interval that contains the median frequency be the *median interval*. We can approximate the median of the entire data set (e.g., the median salary) by interpolation using the formula:

$$median = L_1 + \left(\frac{N/2 - (\sum freq)_t}{freq_{median}}\right) width$$

where L_1 is the lower boundary of the median interval, N is the number of values in the entire data set, $(\sum freq)_l$ is the sum of the frequencies of all of the intervals that are lower than the median interval, $f\ req_{median}$ is the frequency of the median interval, and *width* is the width of the median interval.

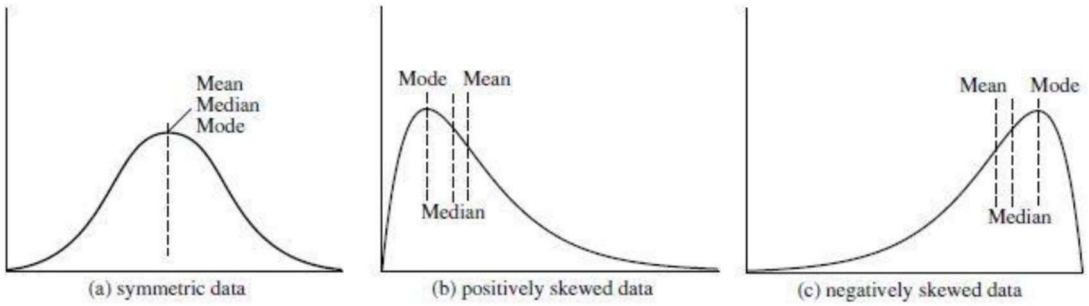


Figure3.1: Mean, median, and mode of symmetric versus positively and negatively skewed data.[1]

Another measure of central tendency is the *mode*. The **mode** for a set of data is the value that occurs most frequently in the set. It is possible for the greatest frequency to correspond to several different values, which results in more than one mode. For unimodal frequency curves that are moderately skewed (asymmetrical), we have the following empirical relation:

$$mean \times mode = 3 \times (mean - median).$$

This implies that the mode for unimodal frequency curves that are moderately skewed can easily be computed if the mean and median values are known. In a unimodal frequency curve with perfect symmetric data distribution, the mean, median, and mode are all at the same

center value, as shown in Figure 3.1(a). However, data in most real applications are not symmetric. They may instead be either positively skewed, where the mode occurs at a value that is smaller than the median (Figure 3.1(b)), or negatively skewed, where the mode occurs at a value greater than the median (Figure 3.1(c)).[1]

3.2.2 Measuring the Dispersion of Data

The degree to which numerical data tend to spread is called the *dispersion*, or *variance* of the data. The most common measures of data dispersion are range, the five-number summary (based on quartiles), the interquartile range, and the standard deviation. Boxplots can be plotted based on the five-number summary and are a useful tool for identifying outliers.[1]

Range, Quartiles, Outliers, and Boxplots

Let $x_1, x_2, \ldots, x_N$ be a set of observations for some attribute. The **range** of the set is the difference between the largest (max()) and smallest (min()) values. For the remainder of this section, let's assume that the data are sorted in increasing numerical order. The ***k*th percentile** of a set of data in numerical order is the value x_i having the property that k percent of the data entries lie at or below x_i. The *median* is the 50th percentile.

The most commonly used percentiles other than the median are **quartiles**. The first quartile, denoted by Q_1, is the 25th percentile; the third quartile, denoted by Q_3, is the 75th percentile. The quartiles, including the median, give some indication of the center, spread, and shape of a distribution. The distance between the first and third quartiles is a simple measure of spread that gives the range covered by the middle half of the data. This distance is called the **interquartile range (*IQR*)** and is defined as

$$IQR = Q_3 - Q_1.$$

No single numerical measure of spread, such as *IQR*, is very useful for describing skewed distributions. The spreads of two sides of a skewed distribution are unequal (Figure 3.1). Therefore, it is more informative to also provide the two quartiles Q_1 and Q_3, along with the median. A common rule of thumb for identifying suspected **outliers** is to single out values falling at least $1.5 \times IQR$ above the third quartile or below the first quartile.

Because Q_1, the median, and Q_3 together contain no information about the endpoints (e.g., tails) of the data, a fuller summary of the shape of a distribution can be obtained by providing the lowest and highest data values as well. This is known as the *five-number summary*. The five-number summary of a distribution consists of the median, the quartiles Q_1 and Q_3, and the smallest and largest individual observations, written in the order *Minimum*, Q_1, *Median*, Q_3, *Maximum*.

Boxplots are a popular way of visualizing a distribution. A boxplot incorporates the five-number summary as follows:

- Typically, the ends of the box are at the quartiles, so that the box length is the interquartile range, *IQR*.
- The median is marked by a line within the box.
- Two lines (called *whiskers*) outside the box extend to the smallest (*Minimum*) and largest (*Maximum*) observations.

When dealing with a moderate number of observations, it is worthwhile to plot potential outliers individually. To do this in a boxplot, the whiskers are extended to the extreme low and high observations *only if* these values are less than 1.5 × *IQR* beyond the quartiles. Otherwise, the whiskers terminate at the most extreme observations occurring within 1.5 × *IQR* of the quartiles. The remaining cases are plotted individually. Boxplots can be used in the comparisons of several sets of compatible data. The efficient computation of boxplots, or even *approximate boxplots* (based on approximates of the five-number summary), remains a challenging issue for the mining of large data sets.[1]

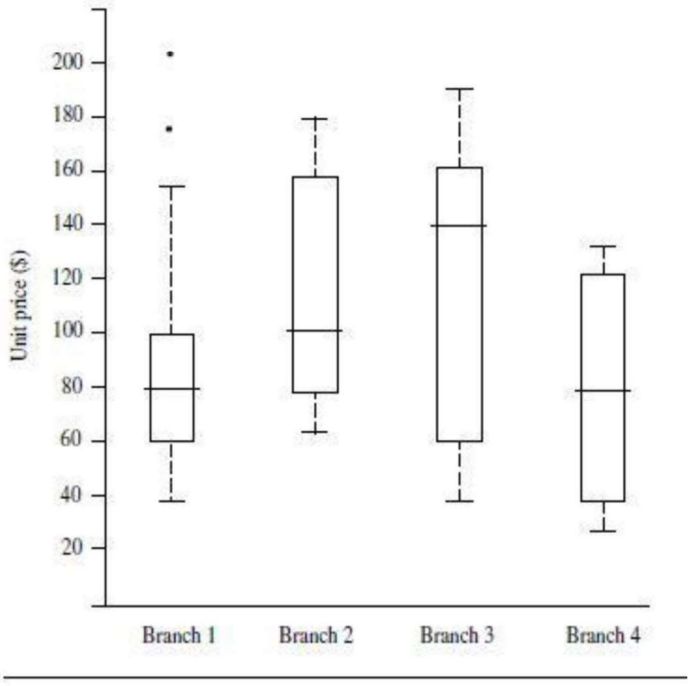


Figure 3.2: Example of boxplot showing the unit price data for items sold at four branches of AllElectronics during a given time period.[1]

Variance and Standard Deviation

The **variance** of N observations, $x_1, x_2, \ldots, x_N$, is

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 = \frac{1}{N}\left[\sum x_i^2 - \frac{1}{N}\left(\sum x_i\right)^2\right]$$

where $\bar{x}$ is the mean value of the observations. The **standard deviation**, σ, of the observations

is the square root of the variance, σ^2. The basic properties of the standard deviation, σ, as a measure of spread are

- measures spread about the mean and should be used only when the mean is chosen as the measure of center.
- $\sigma=0$ only when there is no spread, that is, when all observations have the same value. Otherwise $\sigma > 0$.

The variance and standard deviation are algebraic measures because they can be computed from distributive measures.[1]

3.3 Graphic Displays of Basic Descriptive Data Summaries

Aside from the bar charts, pie charts, and line graphs used in most statistical or graphical data presentation software packages, there are other popular types of graphs for the display of data summaries and distributions, such as *histograms, quantile plots, q-q plots, scatter plots*, and *loess curves*. These graphs are very helpful for the visual inspection of your data.

Plotting **histograms**, or **frequency histograms**, is a graphical method for summarizing the distribution of a given attribute. A histogram for an attribute *A* partitions the data distribution of *A* into disjoint subsets, or *buckets*. Typically, the width of each bucket is uniform. Each bucket is represented by a rectangle whose height is equal to the count or relative frequency of the values at the bucket. If *A* is categoric, such as *automobile model* or *item type*, then one rectangle is drawn for each known value of *A*, and the resulting graph is more commonly referred to as a **bar chart**. If *A* is numeric, the term *histogram* is preferred.[1]

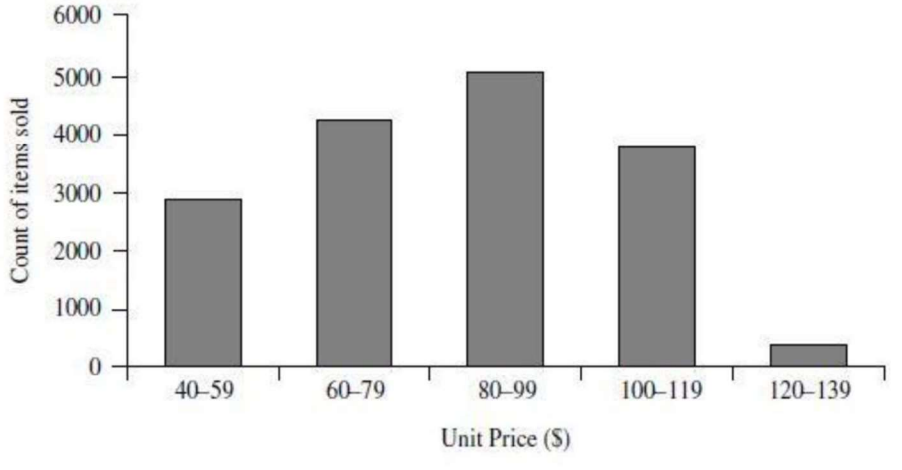


Figure 3.3: Example of histogram.[1]

However, histograms can be a poor method for determining the shape of a distribution because it is so strongly affected by the number of bins used. **Kernal density plots** (or just *density plots*) are usually a much more effective way to view the distribution of a variable.[36]

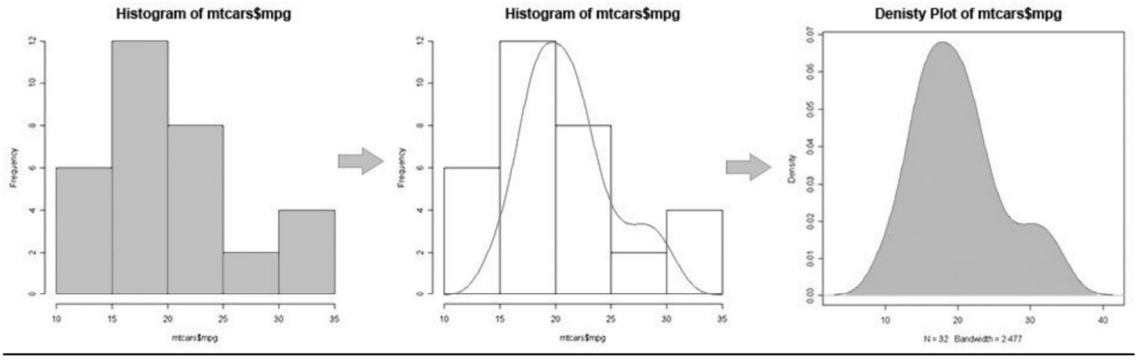


Figure 3.4: Transformation of histogram into Kernal density plot.[36]

A **quantile plot** is a simple and effective way to have a first look at a univariate data distribution. First, it displays all of the data for the given attribute (allowing the user to assess both the overall behaviour and unusual occurrences). Second, it plots quantile information. Let x_i, for $i = 1$ to N, be the data sorted in increasing order so that x_1 is the smallest observation and x_N is the largest. Each observation, x_i, is paired with a percentage, f_i, which indicates that approximately $100f_i$ % of the data are below or equal to the value, x_i. We say "approximately" because there may not be a value with exactly a fraction, f_i, of the data below or equal to x_i. Note that the 0.25 quantile corresponds to quartile Q_1, the 0.50 quantile is the median, and the 0.75 quantile is Q_3.

Let

$$f_i = \frac{i - 0.5}{N}.$$

These numbers increase in equal steps of *1/N*, ranging from *1/2N* (which is slightly above zero) to 1−1/2N (which is slightly below one). On a quantile plot, x_i is graphed against f_i. This allows us to compare different distributions based on their quantiles. For example, given the quantile plots of sales data for two different time periods, we can compare their Q_1, median, Q_3, and other f_i values at a glance.[1]

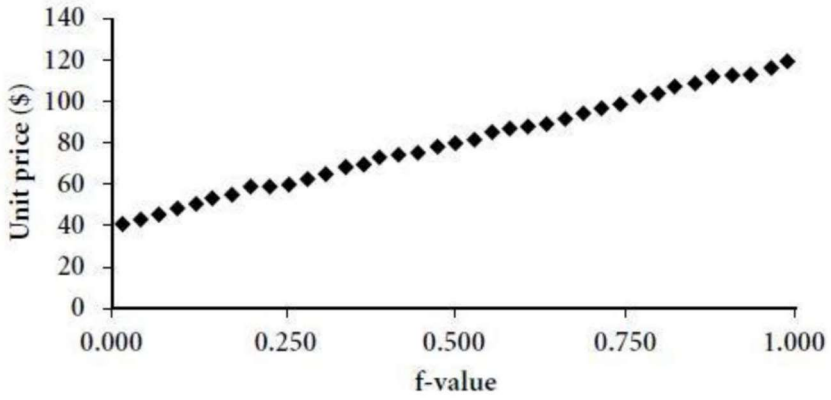


Figure 3.5: Example of quantile plot.[1]

A **quantile-quantile plot**, or q-q plot, graphs the quantiles of one univariate distribution against the corresponding quantiles of another. It is a powerful visualization tool in that it allows the user to view whether there is a shift in going from one distribution to another.

Suppose that we have two sets of observations for the variable *unit price*, taken from two different branch locations. Let $x_1, \ldots, x_N$ be the data from the first branch, and $y_1, \ldots, y_M$ be the data from the second, where each data set is sorted in increasing order. If $M = N$ (i.e., the number of points in each set is the same), then we simply plot y_i against x_i, where y_i and x_i are both $(i–0.5)/N$ quantiles of their respective data sets. If $M < N$ (i.e., the second branch has fewer observations than the first), there can be only M points on the q-q plot. Here, y_i is the $(i–0.5)/M$ quantile of the y data, which is plotted against the $(i–0.5) = M$ quantile of the x data. This computation typically involves interpolation.[1]

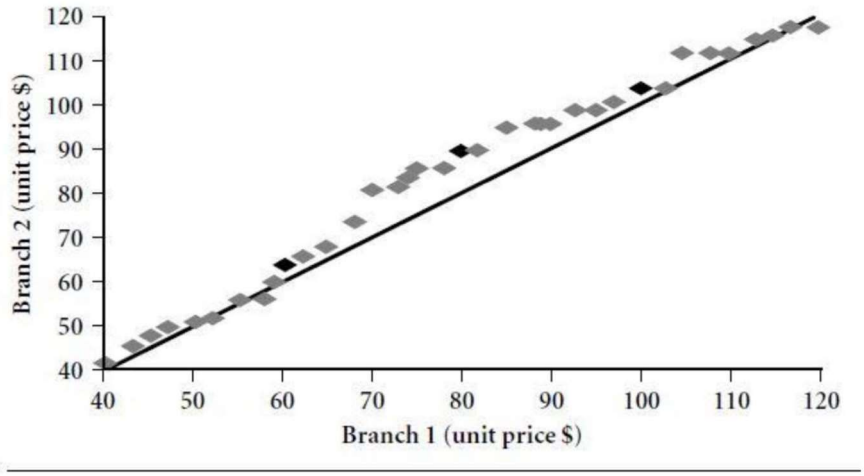


Figure 3.6: Example of quantile-quantile plot.[1]

A **scatter plot** is one of the most effective graphical methods for determining if there appears to be a relationship, pattern, or trend between two numerical attributes. To construct a scatter plot, each pair of values is treated as a pair of coordinates in an algebraic sense and plotted as points in the plane. The scatter plot is a useful method for providing a first look at bivariate data to see clusters of points and outliers, or to explore the possibility of correlation relationships.

Figure 3.7: Example of scatter plot.[1]

In Figure 3.8, we see examples of positive and negative correlations between two attributes in two different data sets. Figure 3.9 shows three cases for which there is no correlation relationship between the two attributes in each of the given data sets.

When dealing with several attributes, the scatter-plot matrix is a useful extension to the scatter plot. Given n attributes, a scatter-plot matrix is an $n \times n$ grid of scatter plots that provides a visualization of each attribute (or dimension) with every other attribute. The scatter-plot matrix becomes less effective as the number of attributes under study grows.[1]

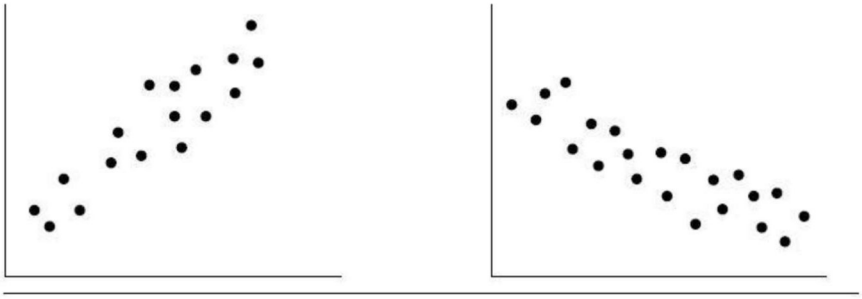


Figure 3.8: Scatter plots can be used to find (a) positive or (b) negative correlations between attributes.[1]

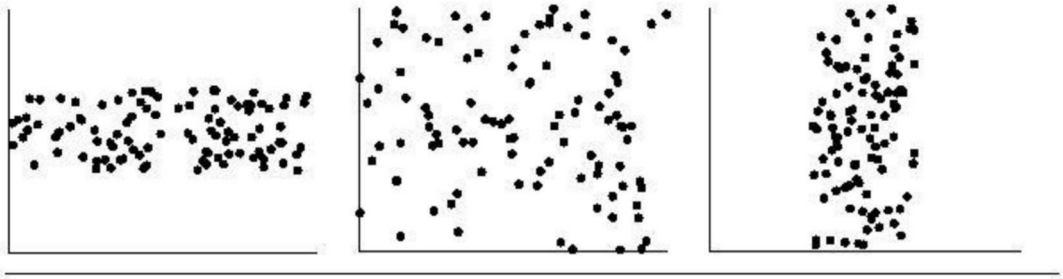


Figure 3.9: Three cases where there is no observed correlation between the two plotted attributes in each of the data sets.[1]

A **loess curve** is another important exploratory graphic aid that adds a smooth curve to a scatter plot in order to provide better perception of the pattern of dependence. The word *loess* is short for "*local regression.*"

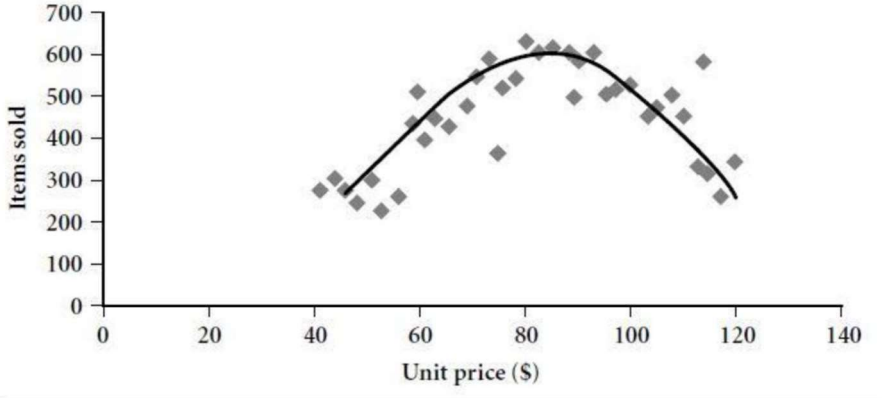


Figure 3.10: Example of loess plot.[1]

To fit a loess curve, values need to be set for two parameters—α, a smoothing parameter, and λ, the degree of the polynomials that are fitted by the regression. While α can be any positive number (typical values are between 1/4 and 1), λ can be 1 or 2. The goal in choosing α is to produce a fit that is as smooth as possible without unduly distorting the underlying pattern in the data. The curve becomes smoother as α increases. There may be some lack of fit, however, indicating possible "missing" data patterns. If α is very small, the underlying pattern is tracked, yet overfitting of the data may occur where local "wiggles" in the curve may not be supported by the data. If the underlying pattern of the data has a "gentle" curvature with no local maxima and minima, then local linear fitting is usually sufficient (λ = 1). However, if there are local maxima or minima, then local quadratic fitting (λ = 2) typically does a better job of following the pattern of the data and maintaining local smoothness. In conclusion, descriptive data summaries provide valuable insight into the overall behaviour of your data. By helping to identify noise and outliers, they are especially useful for data cleaning.[1]

3.4 Forms of Data Preprocessing

Data preprocessing is an important issue for both data warehousing and data mining, as real-world data tend to be incomplete, noisy, and inconsistent. Data preprocessing includes *data cleaning, data integration, data transformation,* and *data reduction.* Figure 3.10 summarizes the data preprocessing steps. These data preprocessing techniques can improve the quality of the data, thereby helping to improve the accuracy and efficiency of the subsequent mining process.[1]

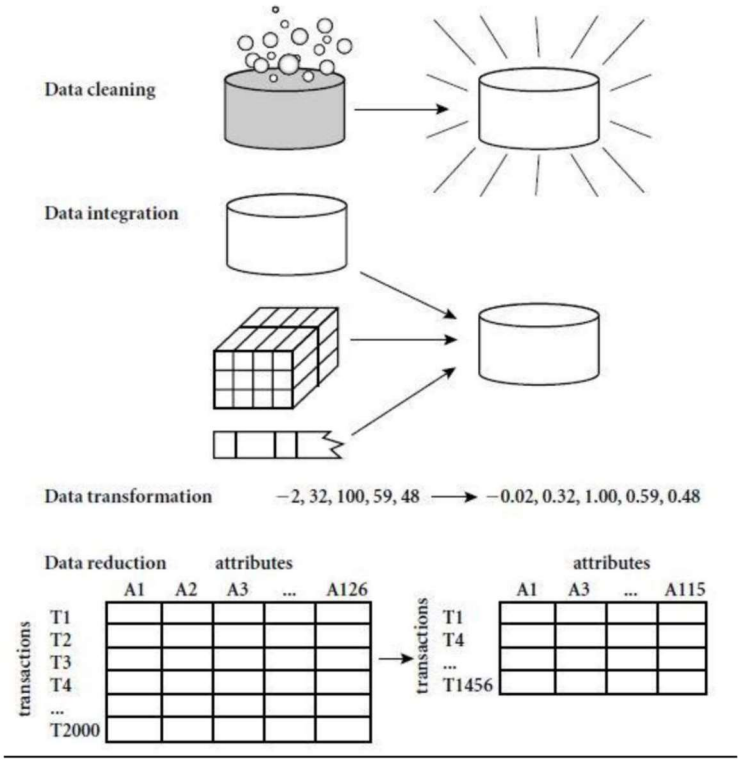


Figure 3.11: Forms of Data Preprocessing.[1]

3.4.1 Data Cleaning

Data cleaning (or *data cleansing*) routines work to "clean" the data by filling in missing values, smoothing noisy data, identifying or removing outliers, and resolving inconsistencies. If users believe the data are dirty, they are unlikely to trust the results of any data mining that has been applied to it. Furthermore, dirty data can cause confusion for the mining procedure, resulting in unreliable output. But how can we go about filling in the missing values for an attribute? We can use the methods like:

(i) ignore the tuple,
(ii) fill in the missing value manually,
(iii) use a global constant to fill in the missing value,
(iv) use the attribute mean to fill in the missing value,
(v) use the attribute mean for all samples belonging to the same class, and
(vi) use the most probable value to fill in the missing value.

Methods 3 to 6 bias the data. Method 6, however, is a popular strategy. In comparison to the other methods, it uses the most information from the present data to predict missing values. By considering the values of the other attributes in its estimation of the missing value there is a greater chance that the relationships between the attributes are preserved. In some cases, a missing value may not imply an error in the data. Each attribute should have one or more rules regarding the *null* condition. The rules may specify whether or not nulls are allowed, and/or how such values should be handled or transformed.[1]

Another major challenge of data cleaning, as discussed, is noise. *"What is noise?"* Noise is a random error or variance in a measured variable. How can we "smooth" out the data to remove the noise? Here are some data smoothing techniques:

- **Binning:** Binning methods smooth a sorted data value by consulting its "neighbourhood," that is, the values around it. The sorted values are distributed into a number of "buckets," or *bins*. Because binning methods consult the neighbourhood of values, they perform *local* smoothing. Some binning strategies are: *smoothing by bin means, smoothing by bin medians* or *smoothing by bin boundaries*.
- **Regression:** Data can be smoothed by fitting the data to a function, such as with regression. *Linear regression* involves finding the "best" line to fit two attributes (or variables), so that one attribute can be used to predict the other. *Multiple linear regression* is an extension of linear regression, where more than two attributes are involved and the data are fit to a multidimensional surface.
- **Clustering:** Outliers may be detected by clustering, where similar values are organized into groups, or "clusters." Intuitively, values that fall outside of the set of clusters may be considered outliers.

So far, we have looked at techniques for handling missing data and for smoothing data. *"But data cleaning is a big job. What about data cleaning as a process?"* The first step in data

cleaning as a process is *discrepancy detection*. Discrepancies can be caused by several factors, including poorly designed data entry forms that have many optional fields, human error in data entry, deliberate errors (e.g., respondents not wanting to divulge information about themselves), and data decay (e.g., outdated addresses). Discrepancies may also arise from inconsistent data representations and the inconsistent use of codes. Errors in instrumentation devices that record data, and system errors, are another source of discrepancies. Errors can also occur when the data are (inadequately) used for purposes other than originally intended. There may also be inconsistencies due to data integration (e.g., where a given attribute can have different names in different databases).

"So, how can we proceed with discrepancy detection?" As a starting point, use any knowledge we may already have regarding properties of the data. Such knowledge or "data about data" is referred to as **metadata**. For example, what are the domain and data type of each attribute? What are the acceptable values for each attribute? What is the range of the length of values? Do all values fall within the expected range? Are there any known dependencies between attributes? The descriptive data summaries are useful here for grasping data trends and identifying anomalies. For example, values that are more than two standard deviations away from the mean for a given attribute may be flagged as potential outliers. In this step, we may write our own scripts and/or use some of the tools. From this, we may find noise, outliers, and unusual values that need investigation. The data should also be examined regarding *unique rules, consecutive rules,* and *null rules*. There are a number of different commercial tools that can aid in the step of discrepancy detection such as data scrubbing tools, data auditing tools, etc.

Most errors, however, will require *data transformations* which is the second step in data cleaning. That is, once we find discrepancies, we typically need to define and apply (a series of) transformations to correct them. Commercial tools that can assist in the data transformation step are data migration tools and ETL (extraction/transformation/loading) tools.[1]

3.4.2 Data Integration

It is likely that your data analysis task will involve data integration, which combines data from multiple sources into a coherent data store. These sources may include multiple databases, data cubes, or flat files. There are a number of issues to consider during data integration like schema integration and object matching. How can equivalent real-world entities from multiple data sources be matched up? This is referred to as the **entity identification problem**.

Redundancy is another important issue. An attribute (such as *annual revenue*, for instance) may be redundant if it can be "derived" from another attribute or set of attributes. Inconsistencies in attribute or dimension naming can also cause redundancies in the resulting data set.

Some redundancies can be detected by correlation analysis. Given two attributes, such analysis can measure how strongly one attribute implies the other, based on the available data. For numerical attributes, we can evaluate the correlation between two attributes, *A* and *B*, by computing the correlation coefficient,

$$r_{A,B} = \frac{\sum_{i=1}^{N}(a_i - \bar{A})(b_i - \bar{B})}{N\sigma_A\sigma_B} = \frac{\sum_{i=1}^{N}(a_i b_i) - N\overline{AB})}{N\sigma_A\sigma_B}$$

where N is the number of tuples, a_i and b_i are the respective values of A and B in tuple i, A and B are the respective mean values of A and B, σ_A and σ_B are the respective standard deviations of A and B and $\sum(a_i b_i)$ is the sum of the AB cross-product (that is, for each tuple, the value for A is multiplied by the value for B in that tuple). Note that $-1 \leq r_{A,B} \leq +1$. If $r_{A,B}$ is greater than 0, then A and B are positively correlated, meaning that the values of A increase as the values of B increase. The higher the value, the stronger the correlation (i.e., the more each attribute implies the other). Hence, a higher value may indicate that A (or B) may be removed as a redundancy. If the resulting value is equal to 0, then A and B are independent and there is no correlation between them. If the resulting value is less than 0, then A and B are negatively correlated, where the values of one attribute increase as the values of the other attribute decrease. This means that each attribute discourages the other. Scatter plots can also be used to view correlations between attributes. Note that correlation does not imply causality. That is, if A and B are correlated, this does not necessarily imply that A causes B or that B causes A. For categorical (discrete) data, a correlation relationship between two attributes, A and B, can be discovered by a χ^2 (chi-square) test.

A third important issue in data integration is the *detection and resolution of data value conflicts*. For example, for the same real-world entity, attribute values from different sources may differ. This may be due to differences in representation, scaling, or encoding. When matching attributes from one database to another during integration, special attention must be paid to the *structure* of the data. This is to ensure that any attribute functional dependencies and referential constraints in the source system match those in the target system. The semantic heterogeneity and structure of data pose great challenges in data integration. Careful integration of the data from multiple sources can help reduce and avoid redundancies and inconsistencies in the resulting data set. This can help improve the accuracy and speed of the subsequent mining process.[1]

3.4.3 Data Transformation

In *data transformation*, the data are transformed or consolidated into forms appropriate for mining. Data transformation can involve the following:[1]

- **Smoothing,** which works to remove noise from the data. Such techniques include binning, regression, and clustering.

- **Aggregation**, where summary or aggregation operations are applied to the data. For example, the daily sales data may be aggregated so as to compute monthly and annual total amounts. This step is typically used in constructing a data cube for analysis of the data at multiple granularities.
- **Generalization of the data**, where low-level or "primitive" (raw) data are replaced by higher-level concepts through the use of concept hierarchies. For example, categorical attributes, like *street*, can be generalized to higher-level concepts, like *city* or *country*. Similarly, values for numerical attributes, like *age*, may be mapped to higher-level concepts, like *youth, middle-aged*, and *senior*.
- **Normalization**, where the attribute data are scaled so as to fall within a small specified range, such as −1.0 to 1.0, or 0.0 to 1.0. Attribute construction (or *feature construction*), where new attributes are constructed and added from the given set of attributes to help the mining process.

An attribute is normalized by scaling its values so that they fall within a small specified range, such as 0.0 to 1.0.Normalization is particularly useful for classification algorithms involving neural networks, or distance measurements such as nearest-neighbour classification and clustering. If using the neural network backpropagation algorithm for classification mining, normalizing the input values for each attribute measured in the training tuples will help speed up the learning phase. For distance-based methods, normalization helps prevent attributes with initially large ranges (e.g., *income*) from outweighing attributes with initially smaller ranges (e.g., binary attributes). There are many methods for data normalization, such as: *min-max normalization, z-score normalization,* and *normalization by decimal scaling.*[1]

3.4.4 Data Reduction

The data set will likely be huge! Complex data analysis and mining on huge amounts of data can take a long time, making such analysis impractical or infeasible. Data reduction techniques can be applied to obtain a reduced representation of the data set that is much smaller in volume, yet closely maintains the integrity of the original data. That is, mining on the reduced data set should be more efficient yet produce the same (or almost the same) analytical results. Some strategies for data reduction include: data cube aggregation, attribute subset selection, dimensionality reduction, numerosity reduction and discretization and concept hierarchy generation. The computational time spent on data reduction should not outweigh or "erase" the time saved by mining on a reduced data set size.[1]

Chapter 4

Model Assessment and Selection

Databases are rich with hidden information that can be used for intelligent decision making. Classification and prediction are two forms of data analysis that can be used to extract models describing important data classes or to predict future data trends. Such analysis can help provide us with a better understanding of the data at large. Whereas *classification* predicts categorical (discrete, unordered) labels, *prediction* models continuous valued functions. Many classification and prediction methods have been proposed by researchers in machine learning, pattern recognition, and statistics. Classification and prediction have numerous applications, including fraud detection, target marketing, performance prediction, manufacturing, and medical diagnosis. The *generalization* performance of a learning method relates to its prediction capability on independent test data. Assessment of this performance is extremely important in practice, since it guides the choice of learning method or model, and gives us a measure of the quality of the ultimately chosen model.[1][2]

4.1 What Is Classification? What Is Prediction?

The data analysis task is **classification**, where a model or **classifier** is constructed to predict *categorical labels.* These categories can be represented by discrete values, where the ordering among values has no meaning. On the other hand, **prediction** is form of analysis, where the model constructed predicts a *continuous-valued function*, or *ordered value*, as opposed to a categorical label. This model is a **predictor**. Regression analysis is a statistical methodology that is most often used for numeric prediction.

"How does classification work? Data classification is a two-step process. In the first step, a classifier is built describing a predetermined set of data classes or concepts. This is the **learning step** (or training phase), where a classification algorithm builds the classifier by analyzing or "learning from" a **training set** made up of database tuples and their associated class labels. A tuple, $\boldsymbol{X}$, is represented by an n-dimensional **attribute vector**, $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$, depicting n measurements made on the tuple from n database attributes, respectively, A_1,

A_2, …, A_n. Each tuple, $\boldsymbol{X}$, is assumed to belong to a predefined class as determined by another database attribute called the **class label attribute**. The class label attribute is discrete-valued and unordered. It is *categorical* in that each value serves as a category or class. The individual tuples making up the training set are referred to as **training tuples** and are selected from the database under analysis.

Because the class label of each training tuple *is provided*, this step is also known as **supervised learning** (i.e., the learning of the classifier is "supervised" in that it is told to which class each training tuple belongs). It contrasts with **unsupervised learning** (or **clustering**), in which the class label of each training tuple is not known, and the number or set of classes to be learned may not be known in advance.

This first step of the classification process can also be viewed as the learning of a mapping or function, $y = f(\boldsymbol{X})$, that can predict the associated class label y of a given tuple $\boldsymbol{X}$. In this view, we wish to learn a mapping or function that separates the data classes. Typically, this mapping is represented in the form of classification rules, decision trees, or mathematical formulae.[1]

"What about classification accuracy?" In the second step (Figure 6.1(b)), the model is used for classification. First, the predictive accuracy of the classifier is estimated. If we were to use the training set to measure the accuracy of the classifier, this estimate would likely be optimistic, because the classifier tends to **overfit** the data (i.e., during learning it may incorporate some particular anomalies of the training data that are not present in the general data set overall). Therefore, a **test set** is used, made up of **test tuples** and their associated class labels. These tuples are randomly selected from the general data set. They are independent of the training tuples, meaning that they are not used to construct the classifier.

The **accuracy** of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier. The associated class label of each test tuple is compared with the learned classifier's class prediction for that tuple. If the accuracy of the classifier is considered acceptable, the classifier can be used to classify future data tuples for which the class label is not known.

"How is (numeric) prediction different from classification?" Data prediction is a two-step process, similar to that of data classification. However, for prediction, we lose the terminology of "class label attribute" because the attribute for which values are being predicted is continuous-valued (ordered) rather than categorical (discrete-valued and unordered). The attribute can be referred to simply as the **predicted attribute**. Prediction can also be viewed as a mapping or function, $y = f(\boldsymbol{X})$, where $\boldsymbol{X}$ is the input (e.g., a tuple describing a loan applicant), and the output y is a continuous or ordered value. That is, we wish to learn a mapping or function that models the relationship between $\boldsymbol{X}$ and y.

Prediction and classification also differ in the methods that are used to build their respective models. As with classification, the training set used to build a predictor should not be used to assess its accuracy. An independent test set should be used instead. The accuracy of a predictor is estimated by computing an error based on the difference between the predicted value and the actual known value of y for each of the test tuples, $\boldsymbol{X}$. There are various predictor error measures.[1]

4.2 Preparing the Data for Classification and Prediction

Before we evaluate any model using data, preprocessing steps may be applied to the data to help improve the accuracy, efficiency, and scalability of the classification or prediction process. Preprocessing of the data in preparation for classification and prediction can involve **data cleaning** to reduce noise or handle missing values, **relevance analysis** to remove irrelevant or redundant attributes, and **data transformation**, such as generalizing the data to higher-level concepts or normalizing the data.[1]

4.2.1 Overfitting and Data Splitting

In statistics and machine learning, **overfitting** occurs when a statistical model describes random error or noise instead of the underlying relationship. Overfitting generally occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model that has been overfit will generally have poor predictive performance, as it can exaggerate minor fluctuations in the data.

The possibility of overfitting exists because the criterion used for training the model is not the same as the criterion used to judge the efficacy of a model. In particular, a model is typically trained by maximizing its performance on some set of training data. However, its efficacy is determined not by its performance on the training data but by its ability to perform well on unseen data. Overfitting occurs when a model begins to "memorize" training data rather than "learning" to generalize from trend. As an extreme example, if the number of parameters is the same as or greater than the number of observations, a simple model or learning process can perfectly predict the training data simply by memorizing the training data in its entirety, but such a model will typically fail drastically when making predictions about new or unseen data, since the simple model has not learned to generalize at all.[2]

In order to avoid overfitting, it is necessary to use the given dataset in splitted form of two sets: *training set* and *test set*. **Training set** is used to train the model and **test set** is the data exclusively used for evaluating and testing the model. Training set is wider than test set. The learned patterns are applied to this test set, and the resulting output is compared to the desired output. A number of statistical methods may be used to evaluate the algorithm, such as ROC curves. If the learned patterns do not meet the desired standards, subsequently it is

necessary to re-evaluate and change the pre-processing and data mining steps. If the learned patterns do meet the desired standards, then the final step is to interpret the learned patterns and turn them into knowledge.

Now, there are two types of error which helps to determine the overfitting of model. **Test error**, also referred to as *generalization error*, is the prediction error over an independent test sample. **Training error** is the average loss over the training sample. Training error consistently decreases with model complexity, typically dropping to zero if we increase the model complexity enough. However, a model with zero training error is overfit to the training data and will typically generalize poorly. Also as the model becomes more and more complex, it uses the training data more and is able to adapt to more complicated underlying structures. Hence there is a decrease in bias but an increase in variance. There is some intermediate model complexity that gives minimum expected test error.

If we are in a data-rich situation, the best approach for both problems is to randomly divide the dataset into three parts: a training set, a validation set, and a test set. The training set is used to fit the models; the validation set is used to estimate prediction error for model selection; the test set is used for assessment of the generalization error of the final chosen model. Ideally, the test set should be kept in a "vault," and be brought out only at the end of the data analysis. Suppose instead that we use the test-set repeatedly, choosing the model with smallest test-set error. Then the test set error of the final chosen model will underestimate the true test error, sometimes substantially. It is difficult to give a general rule on how to choose the number of observations in each of the three parts, as this depends on the signal-to- noise ratio in the data and the training sample size. A typical split might be 50% for training, and 25% each for validation and testing:[2][3]

Figure 4.1: Training, test and validation set.[2]

4.3 Comparing Classification and Prediction Methods

Classification and prediction methods can be compared and evaluated according to the following criteria:

- **Accuracy:** The accuracy of a classifier refers to the ability of a given classifier to correctly predict the class label of new or previously unseen data (i.e., tuples without class label information). Similarly, the accuracy of a predictor refers to how well a given predictor can guess the value of the predicted attribute for new or previously unseen data. Accuracy can be estimated using one or more test sets that are independent of the

training set. Because the accuracy computed is only an estimate of how well the classifier or predictor will do on new data tuples, confidence limits can be computed to help gauge this estimate.

- **Speed:** This refers to the computational costs involved in generating and using the given classifier or predictor.
- **Robustness:** This is the ability of the classifier or predictor to make correct predictions given noisy data or data with missing values.
- **Scalability:** This refers to the ability to construct the classifier or predictor efficiently given large amounts of data.
- **Interpretability:** This refers to the level of understanding and insight that is provided by the classifier or predictor. Interpretability is subjective and therefore more difficult to assess.

Recent data mining research has contributed to the development of scalable algorithms for classification and prediction. Additional contributions include the exploration of mined "associations" between attributes and their use for effective classification.[1]

4.4 Classification by Decision Tree Induction

Decision tree induction is the learning of decision trees from class-labeled training tuples. A **decision tree** is a flowchart-like tree structure, where each **internal node** (nonleaf node) denotes a test on an attribute, each **branch** represents an outcome of the test, and each **leaf node** (or *terminal node*) holds a class label. The topmost node in a tree is the **root** node. Some decision tree algorithms produce only *binary* trees, whereas others can produce non binary trees.

"How are decision trees used for classification?" Given a tuple, ***X***, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple. Decision trees can easily be converted to classification rules.

4.4.1 C5.0 Algorithm

C5.0 algorithm is an extension of C4.5 algorithm. C5.0 is the classification algorithm which applies in big data set. C5.0 is better than C4.5 on the efficiency and the memory. C5.0 model works by splitting the sample based on the field that provides the maximum information gain. The C5.0 model can split samples on basis of the biggest information gain field..The sample subset that is get from the former split will be split afterward. The process will continue until

the sample subset cannot be split and is usually according to another field. Finally, examine the lowest level split, those sample subsets that don't have remarkable contribution to the model will be rejected.

Information Gain

Gain is computed to estimate the gain produced by a split over an attribute
Let S be the sample:

- Ci is Class I; i = 1,2,…,m

$I(s1,s2,..,sm)= - \Sigma\ pi \log2\ (pi)$

- Si is the no. of samples in class i

Pi = Si /S, log2 is the binary logarithm

- Let Attribute A have v distinct values.
- Entropy = E(A) is

  $\Sigma\{(S1j+S2j+..+Smj)/S\}*I(s1j,..smj)\ j=1$
- Where Sij is samples in Class i and subset j of Attribute A.

$I(S1j,S2j,..Smj)= - \Sigma\ pij \log2\ (pij)$

- Gain(A)=I(s1,s2,..,sm) - E(A)

Gain ratio then chooses, from among the tests with at least average gain,
The Gain Ratio= P(A)
Gain Ratio(A)= Gain(A)/P(A)

4.4.2 Boosting in C5.0

When you create a boosted C5.0 model (either a rule set or a decision tree), you actually create a set of related models. The model rule browser for a boosted C5.0 model shows the list of models at the top level of the hierarchy, along with the estimated accuracy of each model and the overall accuracy of the ensemble of boosted models. To examine the rules or splits for a particular model, select that model and expand it as you would a rule or branch in a single model.

You can also extract a particular model from the set of boosted models and create a new Rule Set model nugget containing just that model. To create a new rule set from a boosted

C5.0 model, select the rule set or tree of interest and choose either **Single Decision Tree (GM Palette)** or **Single Decision Tree (Canvas)**from the Generate menu.

4.4.3 Random Forest

Random Forest is a trademark term for an ensemble of decision trees. In Random Forest, we've collection of decision trees (so known as "Forest"). To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is planted & grown as follows:

1. If the number of cases in the training set is N, then sample of N cases is taken at random but *with replacement*. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

Algorithm

Precondition: A training set S := (x1, y1), . . . ,(xn, yn), features F, and number of trees in forest B.

1. function RandomForest(S , F)
2. H ← ∅
3. for i ∈ 1, . . . , B do
4. S (i) ← A bootstrap sample from S
5. hi ← RandomizedTreeLearn(S (i) , F)
6. H ← H ∪ {hi}
7. end for
8. return H
9. end function
10. function RandomizedTreeLearn(S , F)
11. At each node:

12. f ← very small subset of F
13. Split on best feature in f
14. return The learned tree
15. end function

4.5 Bayesian Classification

Naïve 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 "naïve." *Bayesian belief networks* are graphical models, which unlike naïve Bayesian classifiers, allow the representation of dependencies among subsets of attributes. Bayesian belief networks can also be used for classification.[1]

"What are Bayesian classifiers?" Bayesian classifiers are statistical classifiers. They can predict class membership probabilities, such as the probability that a given tuple 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 *naïve Bayesian classifier* to be comparable in performance with decision tree and selected neural network classifiers. Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases.

"How effective are Bayesian classifiers?" Various empirical studies of this classifier in comparison to decision tree and neural network classifiers have found it to be comparable in some domains. 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. Bayesian classifiers are also useful in that they provide a theoretical justification for other classifiers that 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 naïve Bayesian classifier.[1]

"How are these probabilities estimated?" $P(H)$, $P(\mathbf{X}|H)$, and $P(\mathbf{X})$ may be estimated from the given data, as we shall see below. **Bayes' theorem** is useful in that it provides a way of calculating the posterior probability, $P(H|\mathbf{X})$, from $P(H)$, $P(\mathbf{X}|H)$, and $P(\mathbf{X})$. Bayes' theorem is

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

4.5.1 Bayes' Theorem

Let ***X*** be a data tuple. In Bayesian terms, ***X*** is considered "evidence." As usual, it is described by measurements made on a set of *n* attributes. Let *H* be some hypothesis, such as that the data tuple ***X*** belongs to a specified class *C*. For classification problems, we want to determine $P(H_j\boldsymbol{X})$, the probability that the hypothesis *H* holds given the "evidence" or observed data tuple ***X***. In other words, we are looking for the probability that tuple ***X*** belongs to class *C*, given that we know the attribute description of ***X***. $P(H_j\boldsymbol{X})$ is the **posterior probability**, or *a posteriori probability*, of *H* conditioned on ***X***. In contrast, *P*(*H*) is the **prior probability**, or *a priori probability*, of *H*. The posterior probability, $P(H_j\boldsymbol{X})$, is based on more information (e.g., customer information) than the prior probability, *P*(*H*), which is independent of ***X***. Similarly, $P(\boldsymbol{X}_jH)$ is the posterior probability of ***X*** conditioned on *H*.

$$P\left(\frac{X}{C_i}\right) = \prod_{k=1}^{n} P(x_k/C_i)$$

$$P\left(\frac{X}{C_i}\right) = P\left(\frac{x_1}{C_i}\right) * P\left(\frac{x_2}{C_i}\right) * \dots * P\left(\frac{x_n}{C_i}\right)$$

4.5.2 Naïve Bayes Classifier

Derivation:[7]

D: Set of tuples

- Each Tuple is an 'n' dimensional attribute vector
- $X : (x_1,x_2,x_3,\dots, x_n)$

Let there be 'm' classes: $C_1, C_2, C_3, \dots, C_m$.

Naïve Bayes classifier predicts *X* belongs to Class C_i iff

- $P(C_i/X) > P(C_j/X)$ *for* $1 <= j <= m , j <> i$

Maximum Posteriori Hypothesis

- $P(Ci/X) = P(X/C_i)\, P(C_i) / P(X)$
- Maximize $P(X/C_i)\, P(C_i)$ *as* $P(X)$ is constant

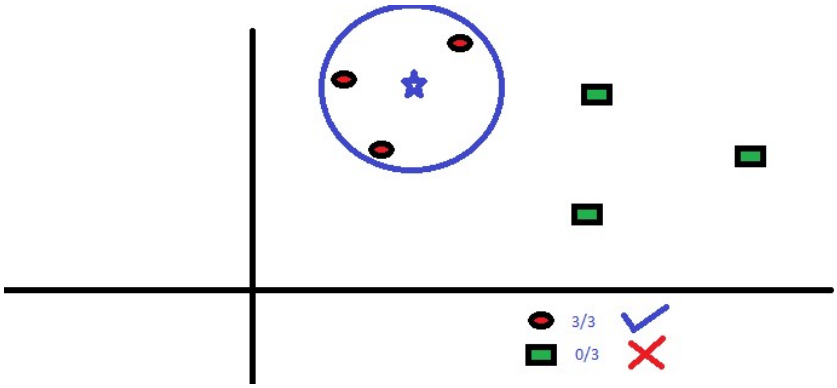
With many attributes, it is computationally expensive to evaluate $P(X/C_i)$.

Naïve Assumption of "class conditional independence"

4.6(A) KNN (K Nearest Neighbors)

It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.

These distance functions can be Euclidean, Manhattan, Minkowski and Hamming distance. First three functions are used for continuous function and fourth one (Hamming) for categorical variables. If K = 1, then the case is simply assigned to the class of its nearest neighbor. At times, choosing K turns out to be a challenge while performing KNN modeling.



KNN can easily be mapped to our real lives. If you want to learn about a person, of whom you have no information, you might like to find out about his close friends and the circles he moves in and gain access to his/her information!

Things to consider before selecting KNN:

- KNN is computationally expensive
- Variables should be normalized else higher range variables can bias it
- Works on pre-processing stage more before going for KNN like outlier, noise removal

These are the steps of the algorithm:

1. Start on an arbitrary vertex as current vertex.
2. Find out the shortest edge connecting current vertex and an unvisited vertex V.
3. Set current vertex to V.

4.Mark V as visited.

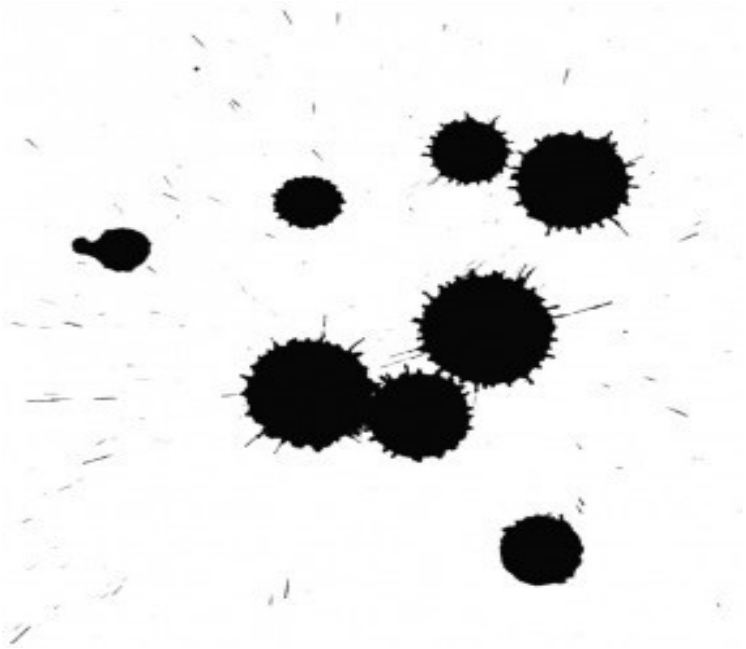
5.If all the vertices in domain are visited, then terminate.

6.Go to step 2.

4.6(B) K MEANS

It is a type of unsupervised algorithm which solves the clustering problem. Its procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters). Data points inside a cluster are homogeneous and heterogeneous to peer groups.

Remember figuring out shapes from ink blots? k means is somewhat similar this activity. You look at the shape and spread to decipher how many different clusters / population are present!



How K-means forms cluster:

1. K-means picks k number of points for each cluster known as centroids.
2. Each data point forms a cluster with the closest centroids i.e. k clusters.

3. Finds the centroid of each cluster based on existing cluster members. Here we have new centroids.
4. As we have new centroids, repeat step 2 and 3. Find the closest distance for each data point from new centroids and get associated with new k-clusters. Repeat this process until convergence occurs i.e. centroids does not change.

How to determine value of K:

In K-means, we have clusters and each cluster has its own centroid. Sum of square of difference between centroid and the data points within a cluster constitutes within sum of square value for that cluster. Also, when the sum of square values for all the clusters are added, it becomes total within sum of square value for the cluster solution.

We know that as the number of cluster increases, this value keeps on decreasing but if you plot the result you may see that the sum of squared distance decreases sharply up to some value of k, and then much more slowly after that. Here, we can find the optimum number of cluster.

4.7 Classifier Accuracy Measures

Now that you may have trained a classifier or predictor, there may be many questions going through your mind. How accurately the classifier can predict? What is accuracy? How can we estimate it? Are there strategies for increasing the accuracy of a learned model?

Using training data to derive a classifier or predictor and then to estimate the accuracy of the resulting learned model can result in misleading overoptimistic estimates due to overspecialization of the learning algorithm to the data. Instead, accuracy is better measured on a test set consisting of class-labeled tuples that were not used to train the model. The **accuracy** of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier. In the pattern recognition literature, this is also referred to as the overall **recognition rate** of the classifier, that is, it reflects how well the classifier recognizes tuples of the various classes.

We can also speak of the **error rate** or **misclassification rate** of a classifier, M, which is simply $1-Acc(M)$, where $Acc(M)$ is the accuracy of M. If we were to use the training set to estimate the error rate of a model, this quantity is known as the **resubstitution** error. This error estimate is optimistic of the true error rate (and similarly, the corresponding accuracy estimate is optimistic) because the model is not tested on any samples that it has not already seen.[1]

The *confusion matrix* is a useful tool for analyzing how well your classifier can recognize tuples of different classes. Given m classes, a **confusion matrix** is a table of at least size m by m. An entry, $CM_{i,j}$ in the first m rows and m columns indicates the number of tuples of class i that were labeled by the classifier as class j. For a classifier to have good accuracy, ideally most of the tuples would be represented along the diagonal of the confusion matrix, from entry $CM_{1,1}$ to entry $CM_{m,m}$, with the rest of the entries being close to zero. The table may have additional rows or columns to provide totals or recognition rates per class.

Given two classes, we can talk in terms of positive tuples versus negative tuples. **True positives** refer to the positive tuples that were correctly labeled by the classifier, while **true negatives** are the negative tuples that were correctly labeled by the classifier. **False positives**

are the negative tuples that were incorrectly labeled. Similarly, **false negatives** are the positive tuples that were incorrectly labeled.

		Predicted class	
		C_1	C_2
Actual class	C_1	true positives	false negatives
	C_2	false positives	true negatives

Figure 4.2: A confusion matrix for positive and negative tuples.[1]

"Are there alternatives to the accuracy measure?" The sensitivity and specificity measures can be used. **Sensitivity** is also referred to as the *true positive (recognition) rate* (that is, the proportion of positive tuples that are correctly identified), while **specificity** is the *true negative rate* (that is, the proportion of negative tuples that are correctly identified). In addition, we may use **precision** to access the percentage of tuples. These measures are defined as

$$sensitivity = \frac{t\_pos}{pos}$$

$$specificity = \frac{t\_neg}{neg}$$

$$precision = \frac{t\_pos}{t\_pos + f\_pos}$$

where *t_pos* is the number of true positives, *pos* is the number of positive tuples, *t_neg* is the number of true negatives, *neg* is the number of negative tuples, and *f_pos* is the number of false positives. It can be shown that accuracy is a function of sensitivity and specificity:

$$accuracy = sensitivity\frac{pos}{pos + neg} + specificity\frac{neg}{pos + neg}$$

The true positives, true negatives, false positives, and false negatives are also useful in assessing the **costs** and **benefits** (or risks and gains) associated with a classification model. The cost associated with a false negative is far greater than that of a false positive. In such cases, we can outweigh one type of error over another by assigning a different cost to each. Similarly, the benefits associated with a true positive decision may be different than that of a true negative.[1]

4.8 Model Selection

Suppose that we have generated two models, M_1 and M_2 (for either classification or prediction), from our data. How can we determine which model is best? It may seem intuitive to select the model with the lowest error rate, however, the mean error rates are just *estimates*

of error on the true population of future data cases. Although the mean error rates obtained for M_1 and M_2 may appear different, that difference may not be statistically significant. What if any difference between the two may just be attributed to chance?[1]

4.8.1 ROC Curves

ROC curves are a useful visual tool for comparing two classification models. The name ROC stands for *Receiver Operating Characteristic*. An ROC curve shows the trade-off between the true positive rate or sensitivity (proportion of positive tuples that are correctly identified) and the false-positive rate (proportion of negative tuples that are incorrectly identified as positive) for a given model. That is, given a two-class problem, it allows us to visualize the trade-off between the rate at which the model can accurately recognize 'yes' cases versus the rate at which it mistakenly identifies 'no' cases as 'yes' for different "portions" of the test set. Any increase in the true positive rate occurs at the cost of an increase in the false-positive rate. The area under the ROC curve is a measure of the accuracy of the model.

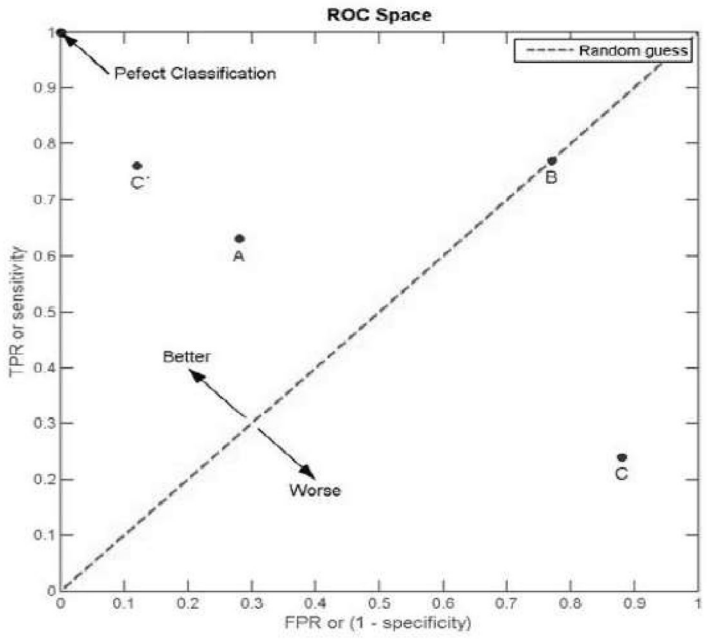


Figure 4.3: ROC Space.[2]

In order to plot an ROC curve for a given classification model, *M*, the model must be able to return a probability or ranking for the predicted class of each test tuple. That is, we need to rank the test tuples in decreasing order, where the one the classifier thinks is most likely to belong to the positive or 'yes' class appears at the top of the list. Naive Bayesian and backpropagation classifiers are appropriate, whereas others, such as decision tree classifiers, can easily be modified so as to return a class probability distribution for each prediction. The vertical axis of an ROC curve represents the true positive rate. The horizontal axis represents the false-positive rate. An ROC curve for *M* is plotted as follows. Starting at the bottom left-hand corner (where the true positive rate and false-positive rate are both 0), we check the

actual class label of the tuple at the top of the list. If we have a true positive (that is, a positive tuple that was correctly classified), then on the ROC curve, we move up and plot a point. If, instead, the tuple really belongs to the 'no' class, we have a false positive. On the ROC curve, we move right and plot a point. This process is repeated for each of the test tuples, each time moving up on the curve for a true positive or toward the right for a false positive.

Figure 4.4 shows the ROC curves of two classification models. The plot also shows a diagonal line where for every true positive of such a model, we are just as likely to encounter a false positive. Thus, the closer the ROC curve of a model is to the diagonal line, the less accurate the model. If the model is really good, initially we are more likely to encounter true positives as we move down the ranked list. Thus, the curve would move steeply up from zero. Later, as we start to encounter fewer and fewer true positives, and more and more false positives, the curve cases off and becomes more horizontal. To assess the accuracy of a model, we can measure the area under the curve. Several software packages are able to perform such calculation. The closer the area is to 0.5, the less accurate the corresponding model is. A model with perfect accuracy will have an area of 1.0.[1]

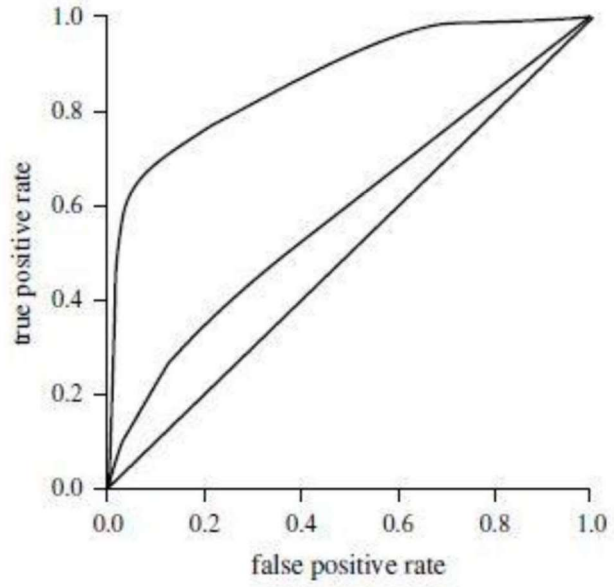


Figure 4.5: The ROC curves of two classification models.[1]

Chapter 5

R Programming and its IDE

5.1 R Programming

R is a programming language and software environment for statistical computing and graphics. The R language is widely used among statisticians and data miners for developing statistical software and data analysis.[8][9] Polls, surveys of data miners, and studies of scholarly literature databases show that R's popularity has increased substantially in recent years.[10][11][12][13]

R is an implementation of the S programming language combined with lexical scoping semantics inspired by Scheme.[14] S was created by John Chambers while at Bell Labs and R was created by Ross Ihaka and Robert Gentleman[15] at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team, of which Chambers is a member. R is named partly after the first names of the first two R authors and partly as a play on the name of S.[16] There are some important differences, but much of the code written for S runs unaltered.

R is a GNU project.[17][18] The source code for the R software environment is written primarily in C, Fortran, and R.[19] R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems. R uses a command line interface; there are also several graphical front-ends for it.

5.1.1 Statistical features

R and its libraries implement a wide variety of statistical and graphical techniques, including linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, and others. R is easily extensible through functions and extensions, and the R community is noted for its active contributions in terms of packages. Many of R's standard functions are written in R itself, which makes it easy for users to follow the algorithmic choices made. For computationally intensive tasks, C, C++, and Fortran code can be linked and called at run time. Advanced users can write C, C++,[20] Java,[21] .NET [22][23][24] or Python code to manipulate R objects directly.

R is highly extensible through the use of user-submitted packages for specific functions or specific areas of study. Due to its S heritage, R has stronger object-oriented programming facilities than most statistical computing languages. Extending R is also eased by its lexical scoping rules.[25] Another strength of R is static graphics, which can produce publication-quality graphs, including mathematical symbols. Dynamic and interactive graphics are available through additional packages. R has its own LaTeX-like documentation format,

which is used to supply comprehensive documentation, both on-line in a number of formats and in hard copy.[26]

5.1.2 Packages

The capabilities of R are extended through user-created packages, which allow specialized statistical techniques, graphical devices (ggplot2), import/export capabilities, reporting tools (knitr, Sweave), etc. These packages are developed primarily in R, and sometimes in Java, C, C++ and Fortran. A core set of packages is included with the installation of R, with more than 5,800 additional packages and 120,000 functions (as of June 2014) available at the Comprehensive R Archive Network (CRAN), Bioconductor, Omegahat, GitHub and other repositories.[12][27]

The "Task Views" page (subject list) on the CRAN website[28] lists a wide range of tasks (in fields such as Finance, Genetics, High Performance Computing, Machine Learning, Medical Imaging, Social Sciences and Spatial Statistics) to which R has been applied and for which packages are available. R has also been identified by the FDA as suitable for interpreting data from clinical research.[29]

5.1.3 Editors and IDEs

Text editors and Integrated development environments (IDEs) with some support for R include: ConTEXT, Eclipse (StatET),[30] Emacs (Emacs Speaks Statistics), LyX (modules for knitr and Sweave), Vim, jEdit,[31] Kate,[32] Revolution R Enterprise DevelopR (part of Revolution R Enterprise),[33] RStudio,[34] Sublime Text, TextMate, WinEdt (R Package RWinEdt), Tinn-R and Notepad++.[35]

5.2 RStudio

RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, and debugging and workspace management.

RStudio is available in open source and commercial editions and runs on the desktop (Windows, Mac, and Linux) or in a browser connected to RStudio Server or RStudio Server Pro (Debian/Ubuntu, RedHat/CentOS, and SUSE Linux). Its features include:

- Access RStudio locally
- Syntax highlighting, code completion, and smart indentation
- Execute R code directly from the source editor
- Quickly jump to function definitions
- Easily manage multiple working directories using projects
- Integrated R help and documentation
- Interactive debugger to diagnose and fix errors quickly
- Extensive package development tools
- All of the features of open source

Implementation of Algorithms

```
#----------------- Loading Libraries And Normalization of Data ------------------#
#Loding Libraries
library(class)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(C50)
library(gmodels)
library(stats)
library(cluster)
library(fpc)
library(plotly)
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(e1071)
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(caTools)
```

```
## Warning: package 'caTools' was built under R version 3.3.2
```

```
#----------------- load orignal dataset ------------------------------------#
data.liver <- read.csv("liver.csv",TRUE,)
str(data.liver)
```

```
## 'data.frame':    583 obs. of  11 variables:
##  $ Age           : int  65 62 62 58 72 46 26 29 17 55 ...
##  $ Gender        : Factor w/ 2 levels "Female","Male": 1 2 2 2 2 2 1 1 2 2 ...
##  $ TB            : num  0.7 10.9 7.3 1 3.9 1.8 0.9 0.9 0.9 0.7 ...
##  $ DB            : num  0.1 5.5 4.1 0.4 2 0.7 0.2 0.3 0.3 0.2 ...
##  $ Alkphos       : int  187 699 490 182 195 208 154 202 202 290 ...
##  $ Sgpt          : int  16 64 60 14 27 19 16 14 22 53 ...
##  $ Sgot          : int  18 100 68 20 59 14 12 11 19 58 ...
##  $ TP            : num  6.8 7.5 7 6.8 7.3 7.6 7 6.7 7.4 6.8 ...
##  $ ALB           : num  3.3 3.2 3.3 3.4 2.4 4.4 3.5 3.6 4.1 3.4 ...
##  $ A.G.Ratio     : num  0.9 0.74 0.89 1 0.4 1.3 1 1.1 1.2 1 ...
##  $ Selector.field: int  1 1 1 1 1 1 1 1 2 1 ...
```

```
# Removing Gender column
data.liver <- data.liver[,c(1,3:11)]


# selecting the column of missing values and defining ina variable
x <- c(data.liver$A.G.Ratio)
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.3000  0.7000  0.9471  0.9471  1.1000  2.8000
```

```
#------------------------ Mean without NA values ----------------------------#
result.mean<-mean(x,na.rm = TRUE)
print(result.mean)
```

```
## [1] 0.9470639
```

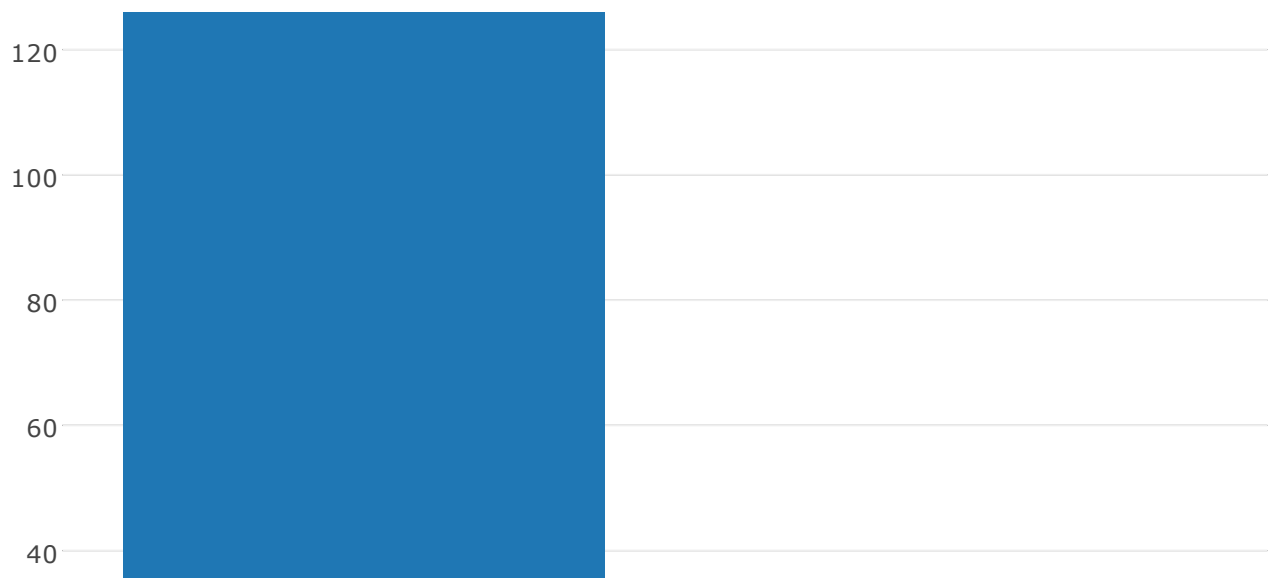
```
liver_normal <- data.liver[,c(1:9)]

# Function for Normalization
normalize <- function(w) {
  return((w-min(w)) / (max(w) - min(w)))
}

# Normalizing the dataset for calculations
liver_normal <- as.data.frame(lapply(liver_normal,normalize))
Selector.field <- as.factor(data.liver$Selector.field)
liver_normal<-cbind(liver_normal,Selector.field)
str(liver_normal)
```

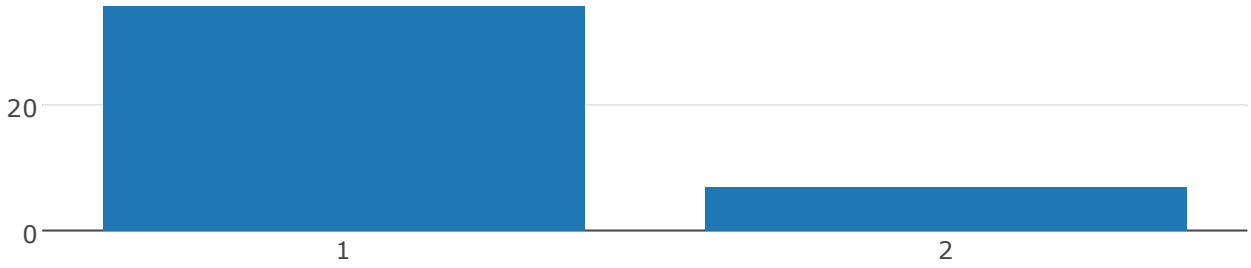
```
## 'data.frame':    583 obs. of  10 variables:
##  $ Age           : num  0.709 0.674 0.674 0.628 0.791 ...
##  $ TB            : num  0.00402 0.14075 0.09249 0.00804 0.04692 ...
##  $ DB            : num  0 0.2755 0.2041 0.0153 0.0969 ...
##  $ Alkphos       : num  0.0606 0.3107 0.2086 0.0581 0.0645 ...
##  $ Sgpt          : num  0.00302 0.02714 0.02513 0.00201 0.00854 ...
##  $ Sgot          : num  0.00163 0.0183 0.01179 0.00203 0.00996 ...
##  $ TP            : num  0.594 0.696 0.623 0.594 0.667 ...
##  $ ALB           : num  0.522 0.5 0.522 0.543 0.326 ...
##  $ A.G.Ratio     : num  0.24 0.176 0.236 0.28 0.04 0.4 0.28 0.32 0.36 0.28 ...
##  $ Selector.field: Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 2 1 ...
```

```
#--------------------------- Data Partitioning for KNN Algorithm  -------------------#
trainer <- liver_normal[1:450,c(1:9)]
tester <- liver_normal[451:583,c(1:9) ]
trainer_target <- liver_normal[1:450,10]
tester_target <- liver_normal[451:583,10]

#################################################################################
#                              Applying KNN Algorithm                           #
#################################################################################

knn_model <- knn(train = trainer, test = tester,
          cl=trainer_target, k=sqrt(583))

# Printing and plotting model results
print(knn_model)
```

```
##   [1] 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1
## [106] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1
## Levels: 1 2
```

```
x<-rchisq(100,5,0)
plot_ly(x=knn_model,type = 'histogram')
```






```
# Summary of the Algorithm
summary(knn_model)
```

```
##   1   2
## 126   7
```

```
#Finding Results using confusion matrix
confusionMatrix(knn_model, tester_target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 92 34
##          2  5  2
##
##                Accuracy : 0.7068
##                  95% CI : (0.6216, 0.7825)
##     No Information Rate : 0.7293
##     P-Value [Acc > NIR] : 0.7552
##
##                   Kappa : 0.0054
##  Mcnemar's Test P-Value : 7.34e-06
##
##             Sensitivity : 0.94845
##             Specificity : 0.05556
##          Pos Pred Value : 0.73016
##          Neg Pred Value : 0.28571
##              Prevalence : 0.72932
##          Detection Rate : 0.69173
##    Detection Prevalence : 0.94737
##       Balanced Accuracy : 0.50200
##
##        'Positive' Class : 1
##
```

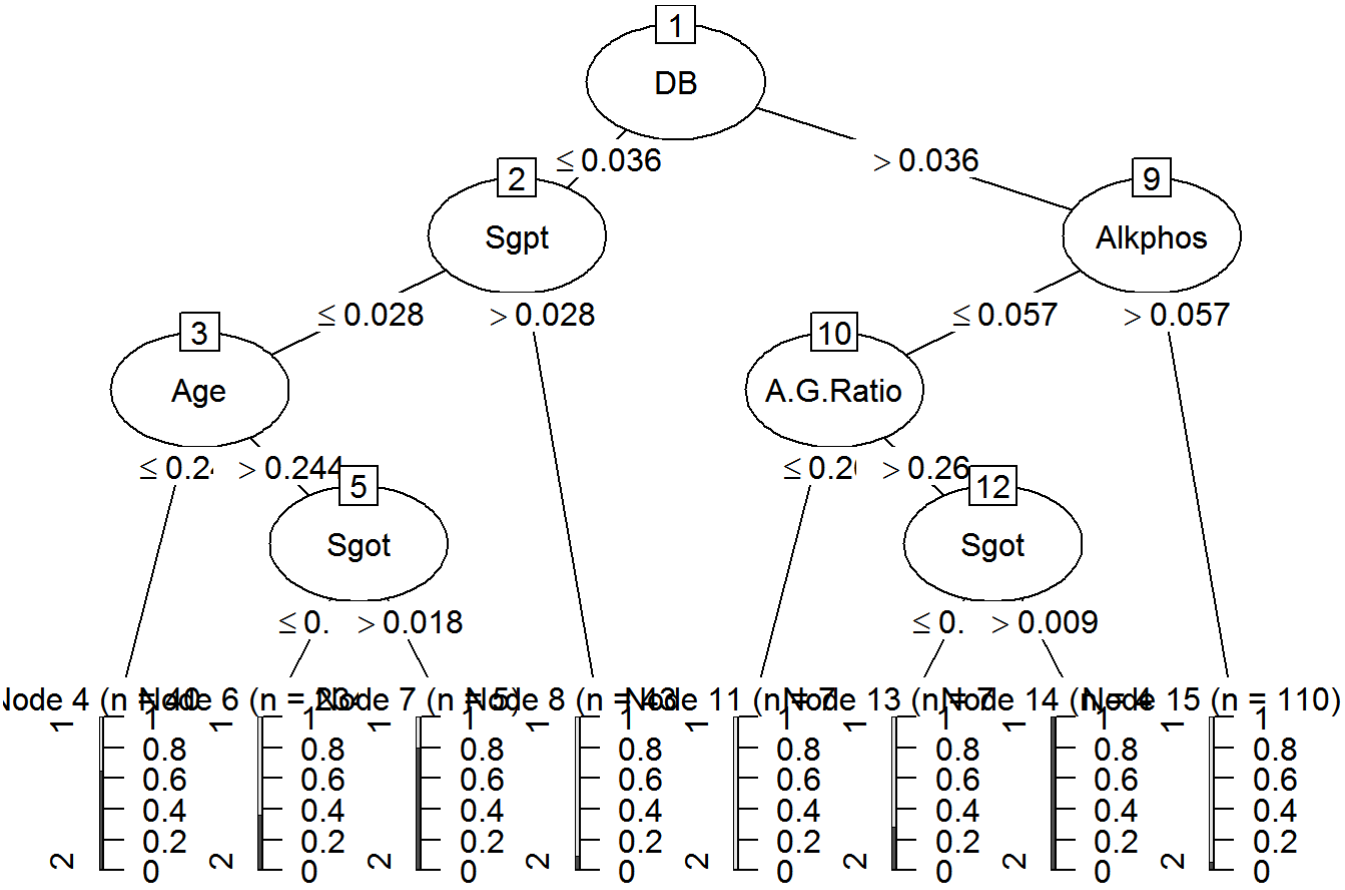
```
#--------------- Validating Data Partitioning for this C5.0 Algorithm ------------------#
trainc50 <- liver_normal[1:450,c(1:9)]
testc50 <- liver_normal[451:583,]
train_targetc50 <- liver_normal[1:450,10]
test_targetc50 <- liver_normal[451:583,10]


#############################################################################
#                                C50 ALGORITHM                              #
#############################################################################

C50_model<- C5.0(trainc50,train_targetc50)

# print result of Algorithm
print(C50_model)
```

```
##
## Call:
## C5.0.default(x = trainc50, y = train_targetc50)
##
## Classification Tree
## Number of samples: 450
## Number of predictors: 9
##
## Tree size: 8
##
## Non-standard options: attempt to group attributes
```

```
# Summary of the Algorithm
summary(C50_model)
```

```
##
## Call:
## C5.0.default(x = trainc50, y = train_targetc50)
##
##
## C5.0 [Release 2.07 GPL Edition]      Fri Dec 02 20:04:43 2016
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 450 cases (10 attributes) from undefined.data
##
## Decision tree:
##
## DB <= 0.03571429:
## :...Sgpt > 0.02763819: 1 (43/4)
## :   Sgpt <= 0.02763819:
## :   :...Age <= 0.244186: 2 (42/14)
## :       Age > 0.244186:
## :       :...Sgot <= 0.0182964: 1 (232/83)
## :           Sgot > 0.0182964: 2 (5/1)
## DB > 0.03571429:
## :...Alkphos > 0.0566683: 1 (110/6)
##     Alkphos <= 0.0566683:
##     :...A.G.Ratio <= 0.26: 1 (7)
##         A.G.Ratio > 0.26:
##         :...Sgot <= 0.008944907: 1 (7/2)
##             Sgot > 0.008944907: 2 (4)
##
##
## Evaluation on training data (450 cases):
##
##      Decision Tree
##    ----------------
##    Size      Errors
##
##       8  110(24.4%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     304    15    (a): class 1
##      95    36    (b): class 2
##
##
##  Attribute usage:
##
##  100.00% DB
##   71.56% Sgpt
##   62.00% Age
##   55.11% Sgot
##   28.44% Alkphos
##    4.00% A.G.Ratio
##
##
## Time: 0.0 secs
```

```
plot(C50_model,type="extended")
```




```
# Testing the model
pred_m<-predict(C50_model,testc50)
print(pred_m)
```

```
##   [1] 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
##  [36] 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [106] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1
## Levels: 1 2
```

```
# Making table of Confusion matrix
CrossTable(test_targetc50, pred_m,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
           dnn = c('actual Selector.Field', 'predicted Selector.field'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  133
##
##
##                      | predicted Selector.field
## actual Selector.Field |         1 |         2 | Row Total |
## ---------------------|-----------|-----------|-----------|
##                    1 |        91 |         6 |        97 |
##                      |     0.684 |     0.045 |           |
## ---------------------|-----------|-----------|-----------|
##                    2 |        32 |         4 |        36 |
##                      |     0.241 |     0.030 |           |
## ---------------------|-----------|-----------|-----------|
##          Column Total |       123 |        10 |       133 |
## ---------------------|-----------|-----------|-----------|
##
##
```

```
# Printing the Result using Confusion Matrix
confusionMatrix(test_targetc50,pred_m)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 91  6
##          2 32  4
##
##                Accuracy : 0.7143
##                  95% CI : (0.6295, 0.7892)
##     No Information Rate : 0.9248
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0637
##  Mcnemar's Test P-Value : 5.002e-05
##
##             Sensitivity : 0.7398
##             Specificity : 0.4000
##          Pos Pred Value : 0.9381
##          Neg Pred Value : 0.1111
##              Prevalence : 0.9248
##          Detection Rate : 0.6842
##    Detection Prevalence : 0.7293
##       Balanced Accuracy : 0.5699
##
##        'Positive' Class : 1
##
```

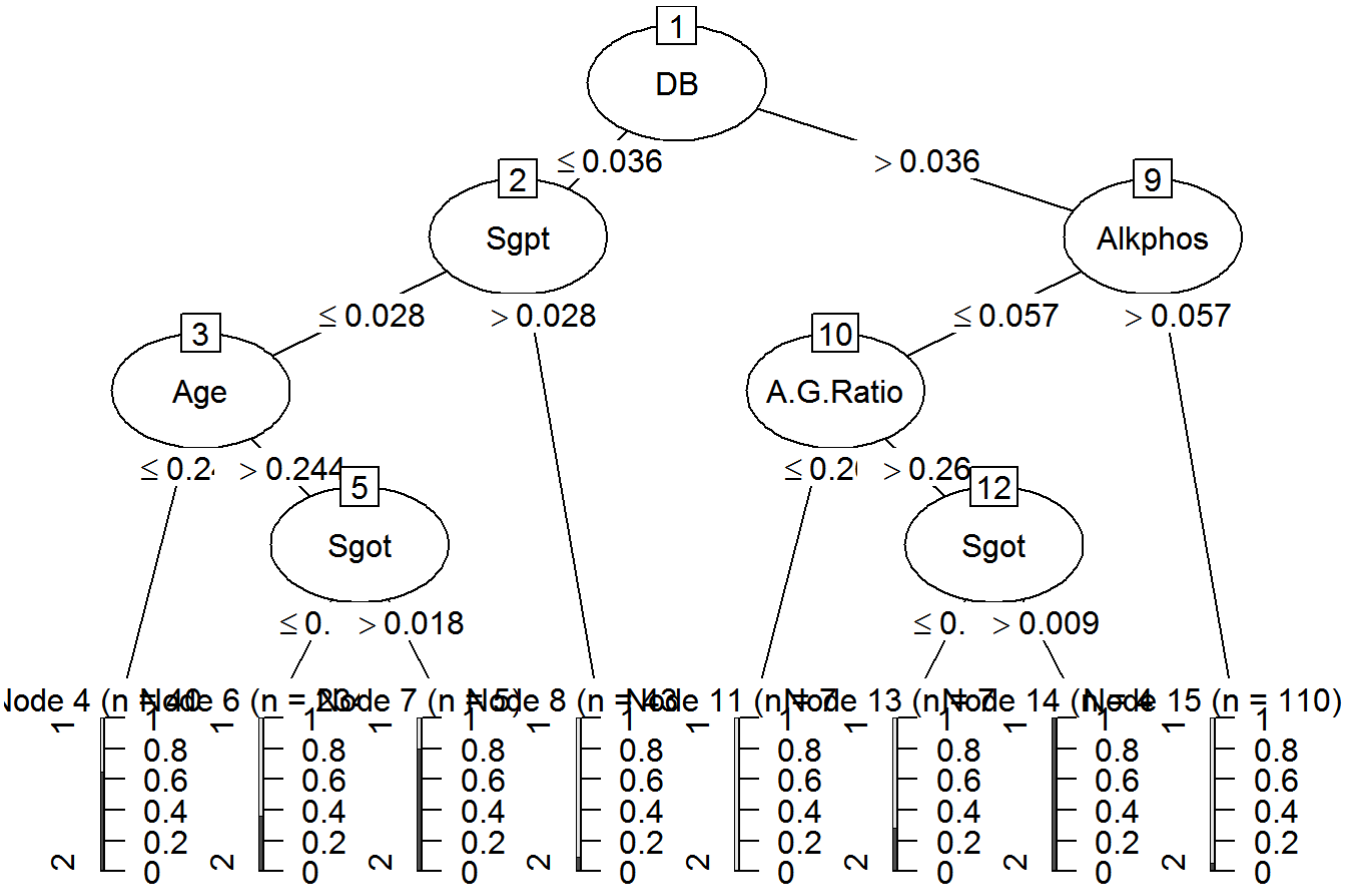
```
#-------------------------------------------------------------------------------#
# Improving the model with (adaptative) boosting
credit_boost10 <- C5.0(trainc50, train_targetc50,
                       trials = 10)
credit_boost10
```

```
##
## Call:
## C5.0.default(x = trainc50, y = train_targetc50, trials = 10)
##
## Classification Tree
## Number of samples: 450
## Number of predictors: 9
##
## Number of boosting iterations: 10 requested;  3 used due to early stopping
## Average tree size: 5.3
##
## Non-standard options: attempt to group attributes
```

```
credit_boost_pred10 <- predict(credit_boost10, testc50)
CrossTable(test_targetc50, credit_boost_pred10,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
           dnn = c('actual Selector.field', 'predicted Selector.field'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  133
##
##
##                       | predicted Selector.field
## actual Selector.field |         1 |         2 | Row Total |
## ----------------------|-----------|-----------|-----------|
##                     1 |        88 |         9 |        97 |
##                       |     0.662 |     0.068 |           |
## ----------------------|-----------|-----------|-----------|
##                     2 |        24 |        12 |        36 |
##                       |     0.180 |     0.090 |           |
## ----------------------|-----------|-----------|-----------|
##          Column Total |       112 |        21 |       133 |
## ----------------------|-----------|-----------|-----------|
##
##
```

```
plot(credit_boost10)
```




```
confusionMatrix(test_targetc50,credit_boost_pred10)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 88  9
##          2 24 12
##
##                Accuracy : 0.7519
##                  95% CI : (0.6696, 0.8226)
##     No Information Rate : 0.8421
##     P-Value [Acc > NIR] : 0.99749
##
##                   Kappa : 0.2768
##  Mcnemar's Test P-Value : 0.01481
##
##             Sensitivity : 0.7857
##             Specificity : 0.5714
##          Pos Pred Value : 0.9072
##          Neg Pred Value : 0.3333
##              Prevalence : 0.8421
##          Detection Rate : 0.6617
##    Detection Prevalence : 0.7293
##       Balanced Accuracy : 0.6786
##
##        'Positive' Class : 1
##
```

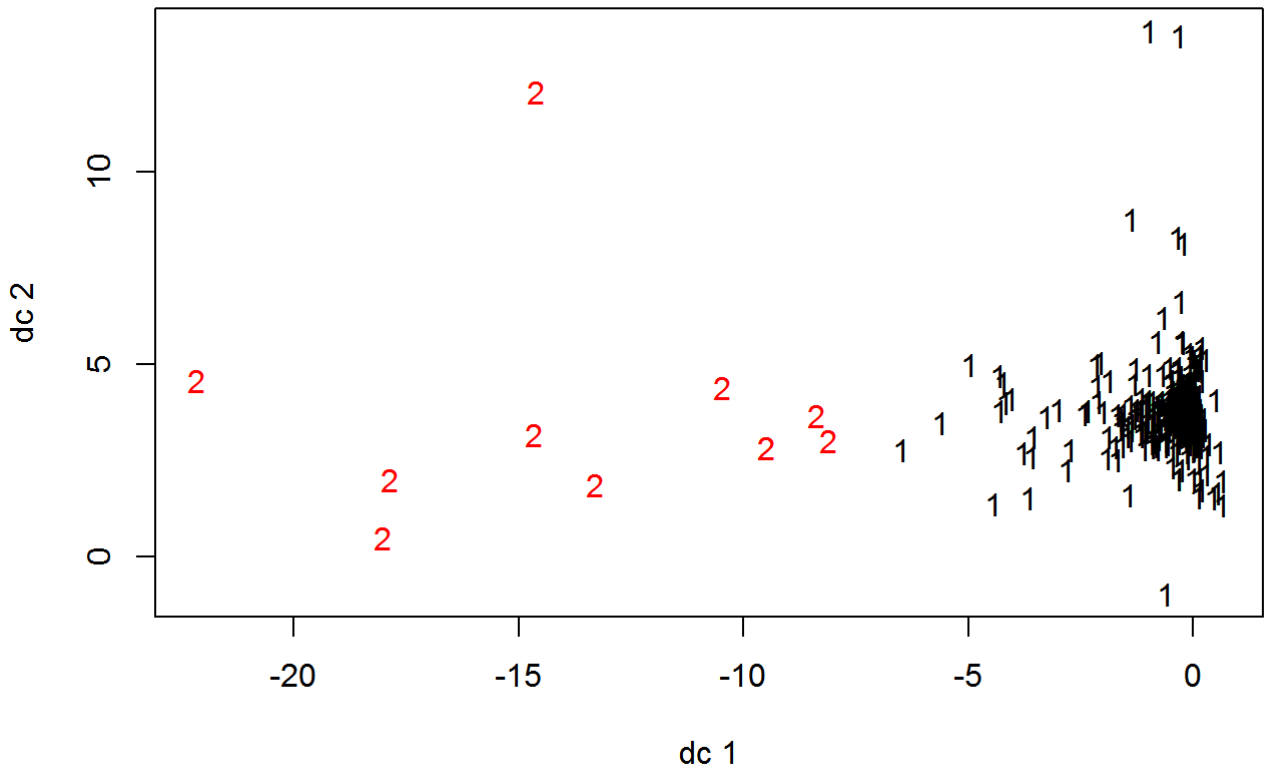
```
#------------------ Prepration of the Data to be implemented in kmeans algorithm ------------
------------#
data.liver1<-data.liver
data.liver1$Selector.field <- as.factor(data.liver1$Selector.field)
str(data.liver1)
```

```
## 'data.frame':    583 obs. of  10 variables:
##  $ Age           : int  65 62 62 58 72 46 26 29 17 55 ...
##  $ TB            : num  0.7 10.9 7.3 1 3.9 1.8 0.9 0.9 0.9 0.7 ...
##  $ DB            : num  0.1 5.5 4.1 0.4 2 0.7 0.2 0.3 0.3 0.2 ...
##  $ Alkphos       : int  187 699 490 182 195 208 154 202 202 290 ...
##  $ Sgpt          : int  16 64 60 14 27 19 16 14 22 53 ...
##  $ Sgot          : int  18 100 68 20 59 14 12 11 19 58 ...
##  $ TP            : num  6.8 7.5 7 6.8 7.3 7.6 7 6.7 7.4 6.8 ...
##  $ ALB           : num  3.3 3.2 3.3 3.4 2.4 4.4 3.5 3.6 4.1 3.4 ...
##  $ A.G.Ratio     : num  0.9 0.74 0.89 1 0.4 1.3 1 1.1 1.2 1 ...
##  $ Selector.field: Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 2 1 ...
```

```
#############################################################################################
#                                  K-means ALGORITHM                                        #
#############################################################################################

kmeans_model <- kmeans(x = subset(data.liver1, select = -Selector.field),
                       centers = 2)

# Printing and Ploting result of algorithm
str(kmeans_model)
```

```
## List of 9
##  $ cluster     : Named int [1:583] 1 1 1 1 1 1 1 1 1 1 ...
##   ..- attr(*, "names")= chr [1:583] "1" "2" "3" "4" ...
##  $ centers     : num [1:2, 1:9] 44.94 34.91 3.17 9.81 1.43 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "1" "2"
##   .. ..$ : chr [1:9] "Age" "TB" "DB" "Alkphos" ...
##  $ totss       : num 1.03e+08
##  $ withinss    : num [1:2] 4.4e+07 2.0e+07
##  $ tot.withinss: num 6.4e+07
##  $ betweenss   : num 38506270
##  $ size        : int [1:2] 572 11
##  $ iter        : int 1
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

```
plotcluster(data.liver1[,-10], kmeans_model$cluster)
```




```
# Summarizing the Algorithm's Result
summary(kmeans_model)
```

```
##              Length Class  Mode
## cluster      583    -none- numeric
## centers       18    -none- numeric
## totss          1    -none- numeric
## withinss       2    -none- numeric
## tot.withinss   1    -none- numeric
## betweenss      1    -none- numeric
## size           2    -none- numeric
## iter           1    -none- numeric
## ifault         1    -none- numeric
```

```
# Making table of Comfusion matrix
mtab <- table(data.liver1$Selector.field,kmeans_model$cluster)
CrossTable(kmeans_model$cluster, data.liver1$Selector.field,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
           dnn = c('actual Selector.Field', 'predicted Selector.field'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  583
##
##
##                      | predicted Selector.field
## actual Selector.Field |         1 |         2 | Row Total |
## ---------------------|-----------|-----------|-----------|
##                    1 |       405 |       167 |       572 |
##                      |     0.695 |     0.286 |           |
## ---------------------|-----------|-----------|-----------|
##                    2 |        11 |         0 |        11 |
##                      |     0.019 |     0.000 |           |
## ---------------------|-----------|-----------|-----------|
##          Column Total |       416 |       167 |       583 |
## ---------------------|-----------|-----------|-----------|
##
##
```

```
# Printing the result using confusion matrix
confusionMatrix(mtab)
```

```
## Confusion Matrix and Statistics
##
##
##       1   2
##   1 405  11
##   2 167   0
##
##                Accuracy : 0.6947
##                  95% CI : (0.6555, 0.7319)
##     No Information Rate : 0.9811
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.0367
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7080
##             Specificity : 0.0000
##          Pos Pred Value : 0.9736
##          Neg Pred Value : 0.0000
##              Prevalence : 0.9811
##          Detection Rate : 0.6947
##    Detection Prevalence : 0.7136
##       Balanced Accuracy : 0.3540
##
##        'Positive' Class : 1
##
```

```
#------------ Preparing Data to be implemented in Naive Bayes Algorithm ----------#
train_naive <- liver_normal[1:450,c(1:9)]
train_target_naive <- liver_normal[1:450,10]
tester_naive <- liver_normal[451:583,c(1:9) ]
tester_target_naive <- liver_normal[451:583,10]

##############################################################################
#                          Applying NAIVE BAYES Algorithm                    #
##############################################################################

bayes_model <- naiveBayes(x = train_naive, y = train_target_naive)

# Summary of the Algorithm
summary(bayes_model)
```

```
##         Length Class  Mode
## apriori 2      table  numeric
## tables  9      -none- list
## levels  2      -none- character
## call    3      -none- call
```

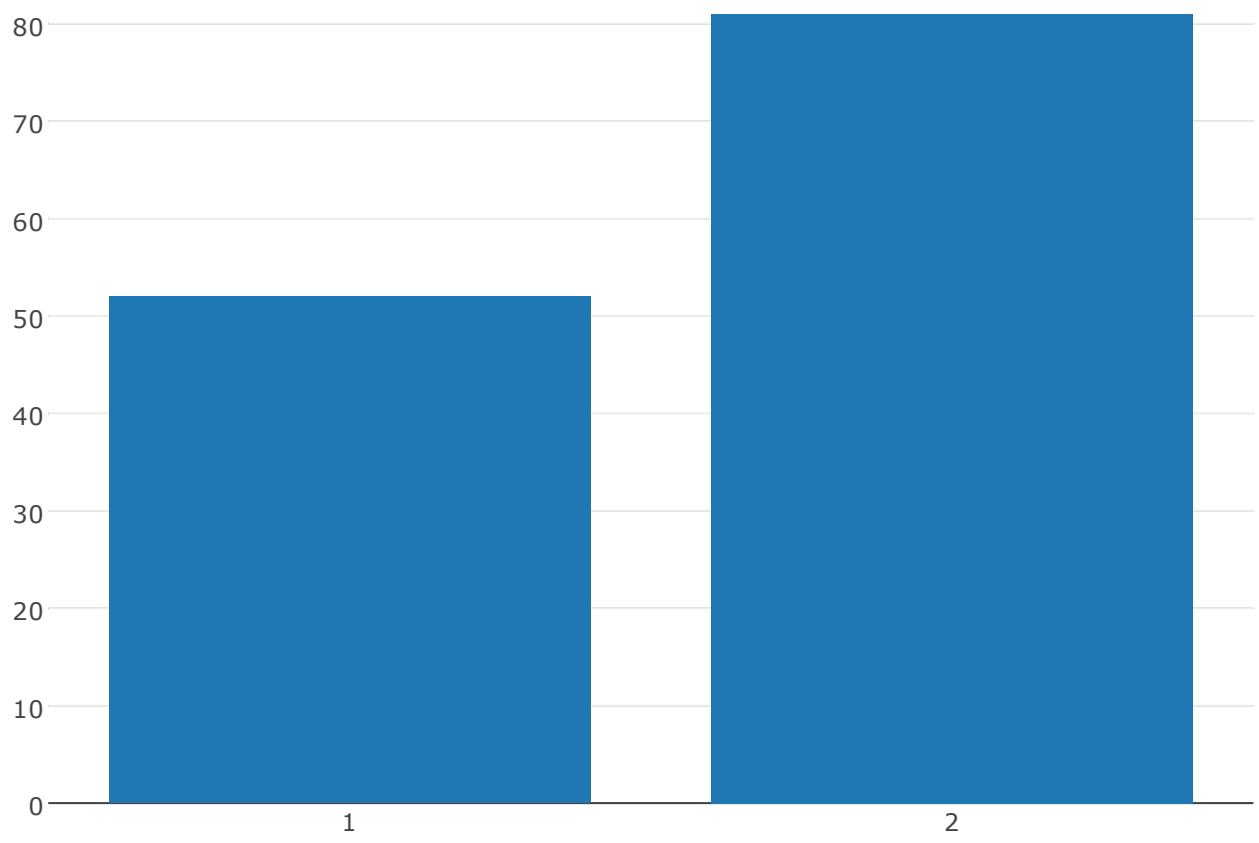
```
# structure result of Algorithm
str(bayes_model)
```

```
## List of 4
##  $ apriori: 'table' int [1:2(1d)] 319 131
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ train_target_naive: chr [1:2] "1" "2"
##  $ tables :List of 9
##   ..$ Age      : num [1:2, 1:2] 0.493 0.429 0.184 0.206
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ train_target_naive: chr [1:2] "1" "2"
##   .. .. ..$ Age               : NULL
##   ..$ TB       : num [1:2, 1:2] 0.0392 0.0109 0.0833 0.0148
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ train_target_naive: chr [1:2] "1" "2"
##   .. .. ..$ TB                : NULL
##   ..$ DB       : num [1:2, 1:2] 0.0698 0.0169 0.1257 0.0291
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ train_target_naive: chr [1:2] "1" "2"
##   .. .. ..$ DB                : NULL
##   ..$ Alkphos  : num [1:2, 1:2] 0.1317 0.0788 0.1422 0.075
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ train_target_naive: chr [1:2] "1" "2"
##   .. .. ..$ Alkphos           : NULL
##   ..$ Sgpt     : num [1:2, 1:2] 0.0472 0.0122 0.1175 0.0131
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ train_target_naive: chr [1:2] "1" "2"
##   .. .. ..$ Sgpt              : NULL
##   ..$ Sgot     : num [1:2, 1:2] 0.02631 0.00627 0.07636 0.00785
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ train_target_naive: chr [1:2] "1" "2"
##   .. .. ..$ Sgot              : NULL
##   ..$ TP       : num [1:2, 1:2] 0.539 0.551 0.161 0.153
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ train_target_naive: chr [1:2] "1" "2"
##   .. .. ..$ TP                : NULL
##   ..$ ALB      : num [1:2, 1:2] 0.477 0.539 0.172 0.167
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ train_target_naive: chr [1:2] "1" "2"
##   .. .. ..$ ALB               : NULL
##   ..$ A.G.Ratio: num [1:2, 1:2] 0.249 0.304 0.113 0.111
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ train_target_naive: chr [1:2] "1" "2"
##   .. .. ..$ A.G.Ratio         : NULL
##  $ levels : chr [1:2] "1" "2"
##  $ call   : language naiveBayes.default(x = train_naive, y = train_target_naive)
##  - attr(*, "class")= chr "naiveBayes"
```

```
# Testing the model
result<-predict(object = bayes_model,newdata = tester_naive)
print(result)
```

```
##   [1] 1 2 1 2 2 2 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 1 1 2 1
##  [36] 1 2 1 2 2 2 2 1 2 1 2 2 2 2 2 2 2 2 1 1 1 1 1 1 2 2 2 2 2 1 2 2 2 1
##  [71] 2 1 2 2 2 2 1 1 2 1 2 1 2 1 2 1 1 2 2 2 2 2 2 2 1 2 1 1 1 1 2 2 2 1 2
## [106] 1 2 2 1 1 1 1 1 2 2 1 2 2 1 1 1 2 1 1 1 1 1 1 2 2 2 2 2
## Levels: 1 2
```

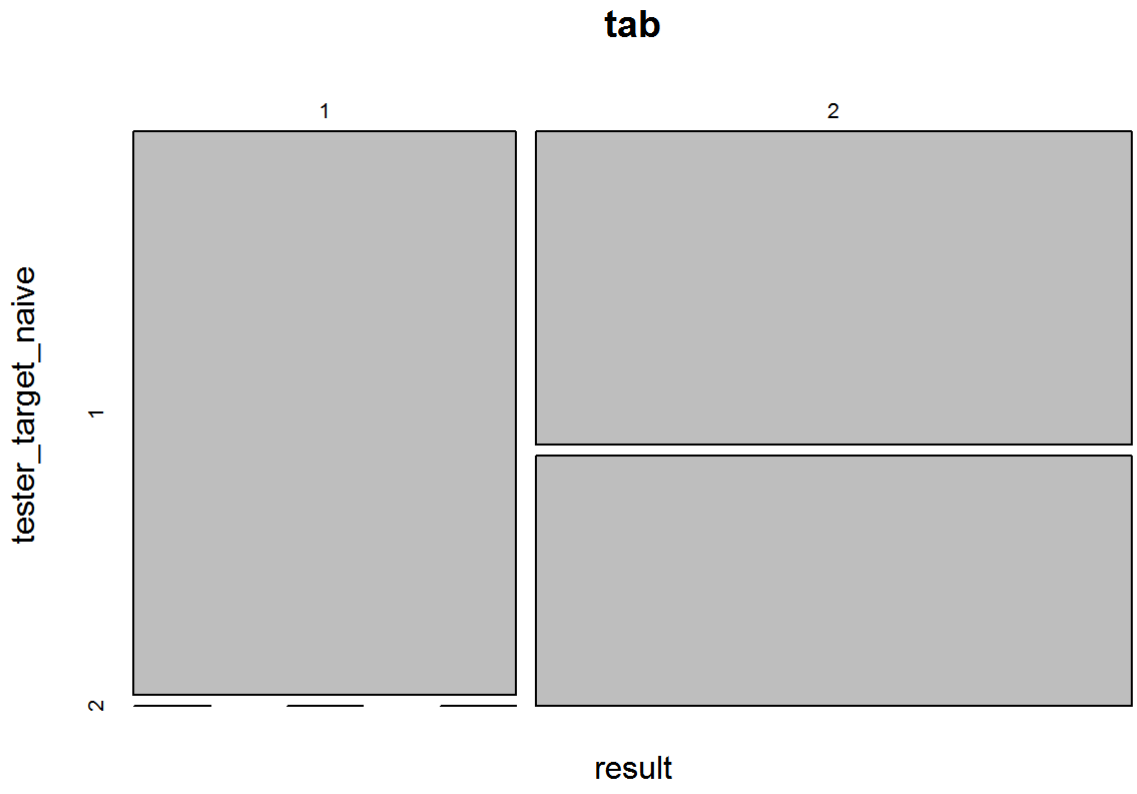
```
x<-rchisq(100,5,0)
plot_ly(x=result,type = 'histogram')
```




```
# Making table of Confusion matrix
tab<-table(result,tester_target_naive)
CrossTable(tester_target_naive, result,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
           dnn = c('actual Selector.Field', 'predicted Selector.field'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  133
##
##
##                      | predicted Selector.field
## actual Selector.Field |         1 |         2 | Row Total |
## ---------------------|-----------|-----------|-----------|
##                    1 |        52 |        45 |        97 |
##                      |     0.391 |     0.338 |           |
## ---------------------|-----------|-----------|-----------|
##                    2 |         0 |        36 |        36 |
##                      |     0.000 |     0.271 |           |
## ---------------------|-----------|-----------|-----------|
##          Column Total |        52 |        81 |       133 |
## ---------------------|-----------|-----------|-----------|
##
##
```

```
plot(tab)
```



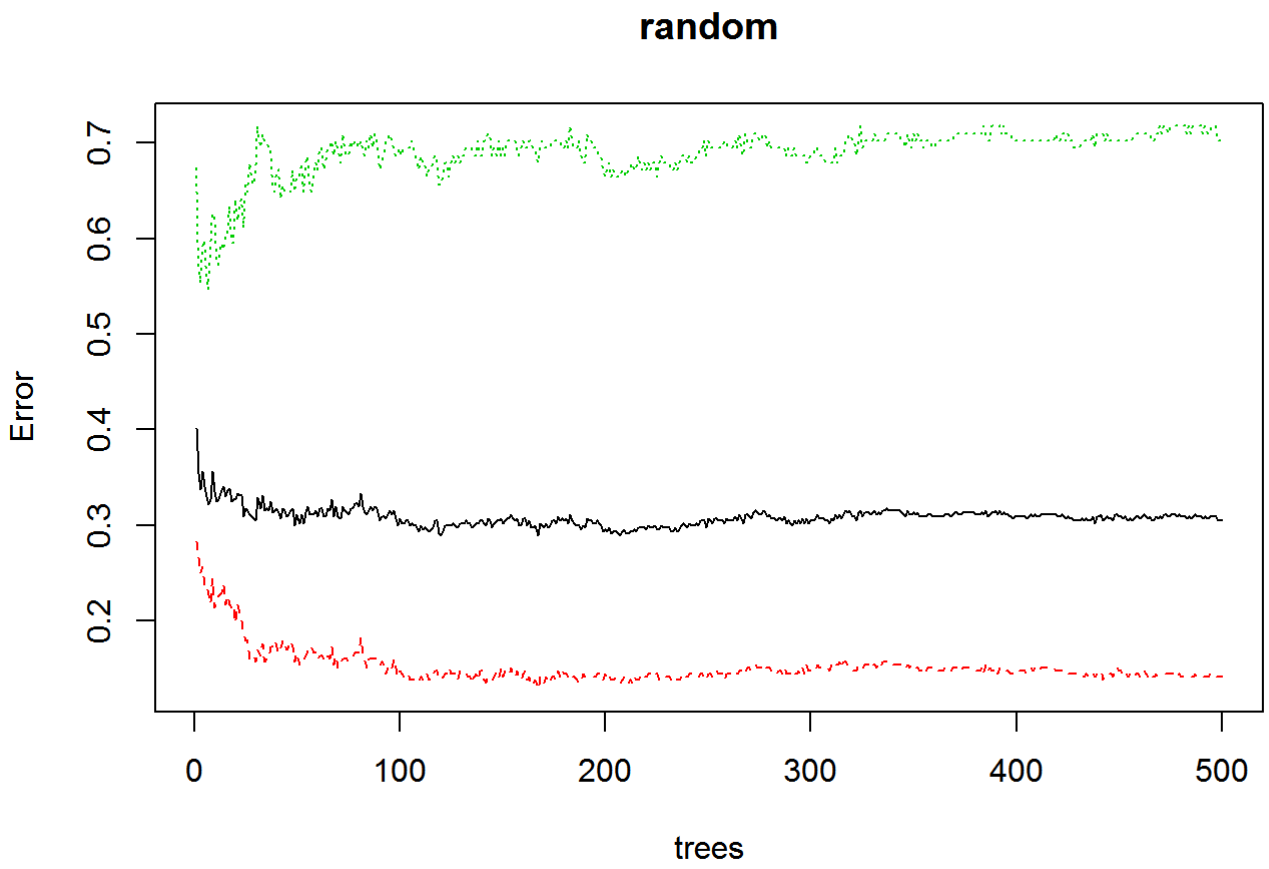
```
# Printing the result using confusion matrix
confusionMatrix(tab)
```

```
## Confusion Matrix and Statistics
##
##       tester_target_naive
## result  1  2
##      1 52  0
##      2 45 36
##
##                Accuracy : 0.6617
##                  95% CI : (0.5746, 0.7414)
##     No Information Rate : 0.7293
##     P-Value [Acc > NIR] : 0.9658
##
##                   Kappa : 0.3848
##  Mcnemar's Test P-Value : 5.412e-11
##
##             Sensitivity : 0.5361
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.4444
##              Prevalence : 0.7293
##          Detection Rate : 0.3910
##    Detection Prevalence : 0.3910
##       Balanced Accuracy : 0.7680
##
##        'Positive' Class : 1
##
```

```
#-----------------------  Prepairing Data to be implemented in this Algorithm --------------
------------#
train_rf <- liver_normal[1:450,c(1:9)]
test_rf <- liver_normal[451:583,]
train_target_rf <- liver_normal[1:450,10]
test_target_rf <- liver_normal[451:583,10]

#############################################################################
#                          Applying RANDOM FOREST Algorithm                 #
#############################################################################

random <- randomForest(train_rf,train_target_rf,ntree = 500)

# Printing and Ploting result of algorithm
print(random)
```

```
##
## Call:
##  randomForest(x = train_rf, y = train_target_rf, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 30.44%
## Confusion matrix:
##     1  2 class.error
## 1 274 45   0.1410658
## 2  92 39   0.7022901
```

```
plot(random)
```




```
# Summarizing the Algorithm's Result
summary(random)
```

```
##                 Length Class  Mode
## call               4   -none- call
## type               1   -none- character
## predicted        450   factor numeric
## err.rate        1500   -none- numeric
## confusion          6   -none- numeric
## votes            900   matrix numeric
## oob.times        450   -none- numeric
## classes            2   -none- character
## importance         9   -none- numeric
## importanceSD       0   -none- NULL
## localImportance    0   -none- NULL
## proximity          0   -none- NULL
## ntree              1   -none- numeric
## mtry               1   -none- numeric
## forest            14   -none- list
## y                450   factor numeric
## test               0   -none- NULL
## inbag              0   -none- NULL
```

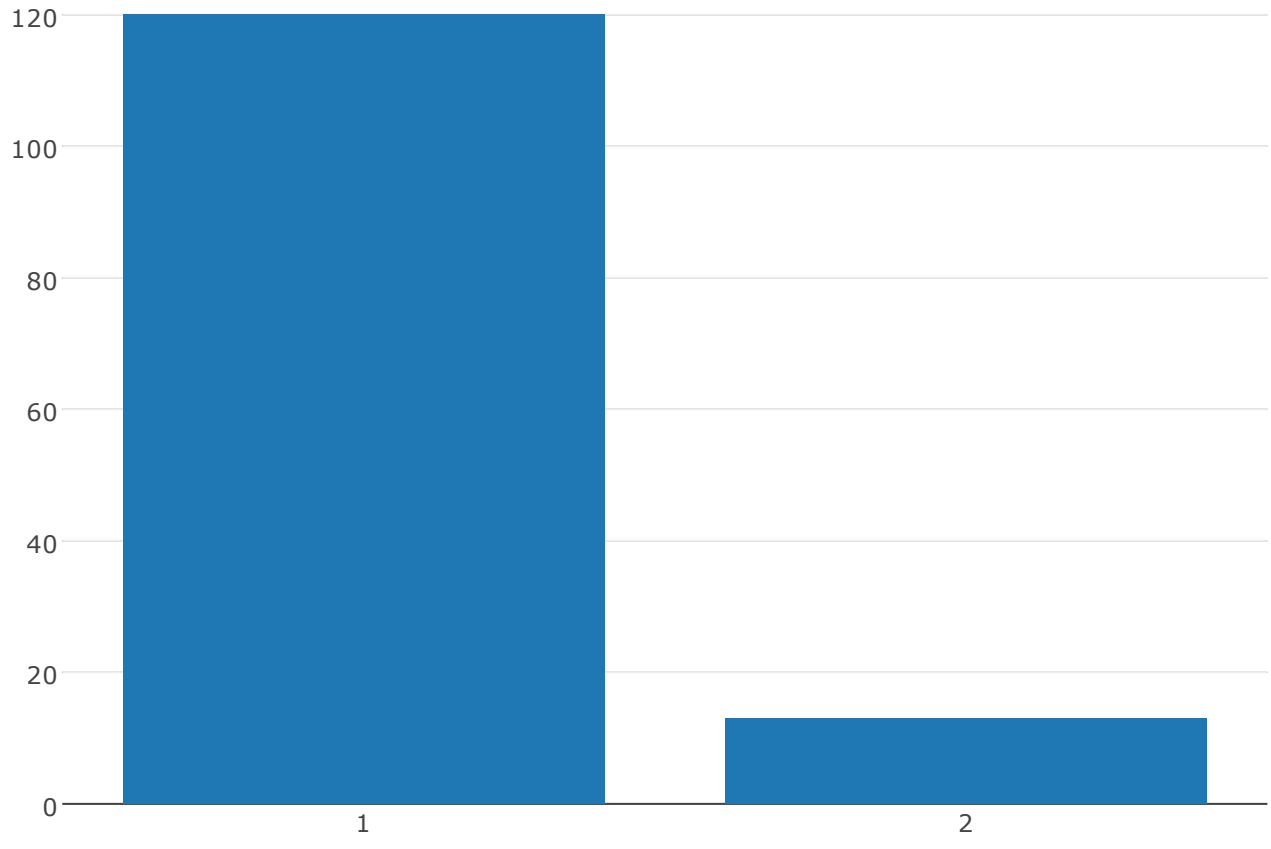
```
# Testing the model
predic <- predict(random,test_rf)
print(predic)
```

```
## 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468
##   1   1   1   1   1   2   1   1   1   1   1   1   1   2   1   1   1   1
## 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486
##   1   1   2   1   1   1   1   1   1   1   1   1   2   1   1   1   1   1
## 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504
##   1   1   1   2   1   2   1   1   1   1   2   1   2   1   1   1   1   1
## 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522
##   1   1   1   1   1   1   1   1   1   1   1   1   2   1   1   1   1   1
## 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540
##   1   2   1   1   1   1   1   1   2   1   1   1   1   1   1   1   1   1
## 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558
##   1   2   1   1   1   1   1   1   1   1   1   2   1   1   1   1   1   1
## 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## 577 578 579 580 581 582 583
##   1   1   1   1   1   1   1
## Levels: 1 2
```

```
# Making table of Confusion matrix
rtab<-table(predic,test_target_rf)
CrossTable(test_target_rf, predic,prop.chisq = FALSE,
           prop.c = FALSE,
           prop.r = FALSE,
           dnn = c('actual Selector.Field',
                   'predicted Selector.field'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  133
##
##
##                      | predicted Selector.field
## actual Selector.Field |         1 |         2 | Row Total |
## ----------------------|-----------|-----------|-----------|
##                     1 |        90 |         7 |        97 |
##                       |     0.677 |     0.053 |           |
## ----------------------|-----------|-----------|-----------|
##                     2 |        30 |         6 |        36 |
##                       |     0.226 |     0.045 |           |
## ----------------------|-----------|-----------|-----------|
##          Column Total |       120 |        13 |       133 |
## ----------------------|-----------|-----------|-----------|
##
##
```

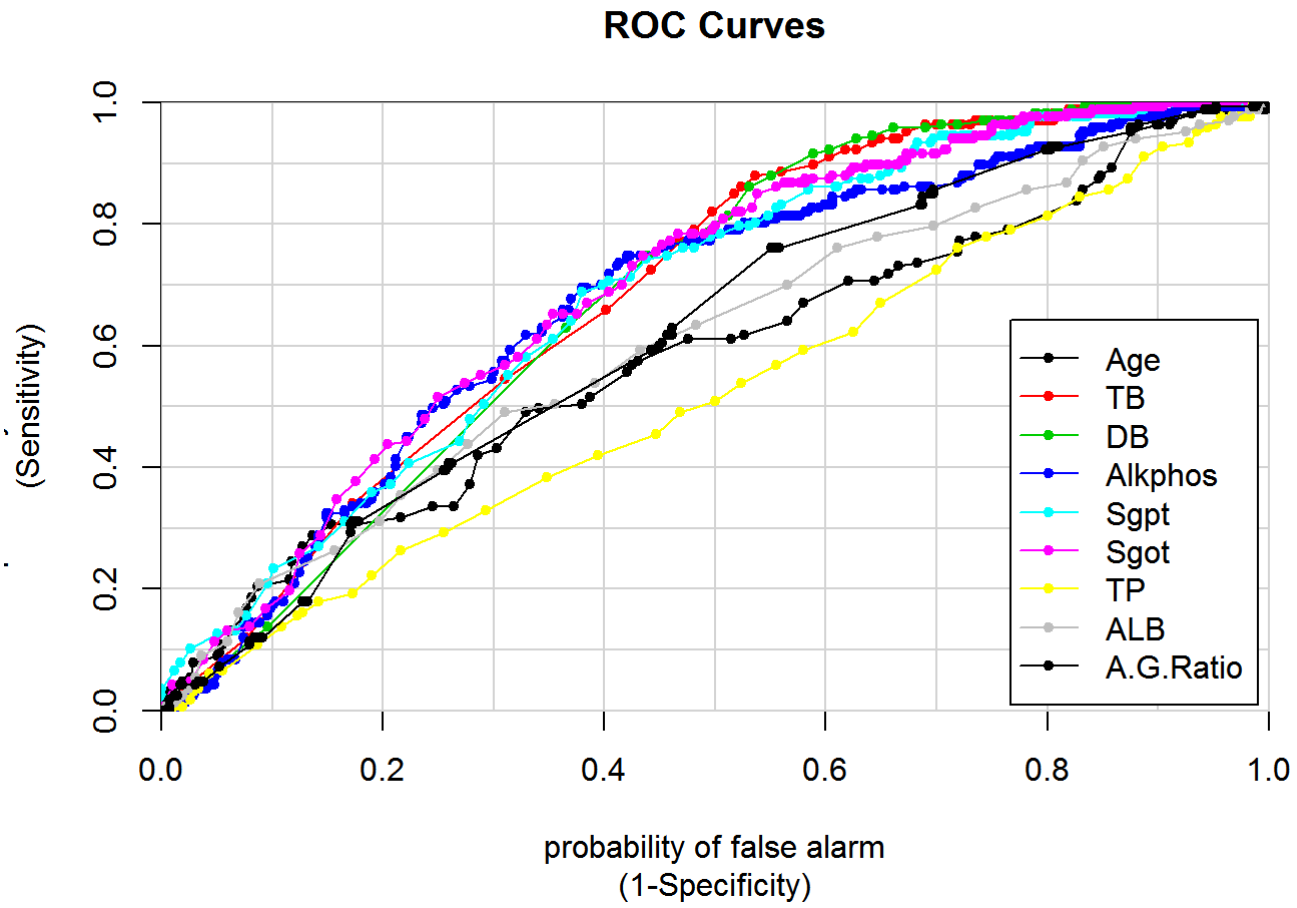
```
x<-rchisq(1000,5,0)
plot_ly(x=predic,type = 'histogram')
```



```
# Printing the result using confusion matrix
confusionMatrix(predic,test_target_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 90 30
##          2  7  6
##
##                Accuracy : 0.7218
##                  95% CI : (0.6375, 0.796)
##     No Information Rate : 0.7293
##     P-Value [Acc > NIR] : 0.6202821
##
##                   Kappa : 0.1183
##  Mcnemar's Test P-Value : 0.0002983
##
##             Sensitivity : 0.9278
##             Specificity : 0.1667
##          Pos Pred Value : 0.7500
##          Neg Pred Value : 0.4615
##              Prevalence : 0.7293
##          Detection Rate : 0.6767
##    Detection Prevalence : 0.9023
##       Balanced Accuracy : 0.5473
##
##        'Positive' Class : 1
##
```

```
#############################################################################################
# ROC curve for Whole Orignal dataset befor implementing algorithms.

colAUC(liver_normal[,-10], liver_normal[,10],
       plotROC=TRUE,alg=c("Wilcoxon","ROC"))
```




```
##               Age        TB       DB  Alkphos      Sgpt      Sgot
## 1 vs. 2 0.5827024 0.6932793 0.686075 0.674466 0.6855856 0.6972161
##                TP       ALB A.G.Ratio
## 1 vs. 2 0.5205622 0.6065897 0.6190264
```

```
cat("Total AUC: \n");
```

```
## Total AUC:
```

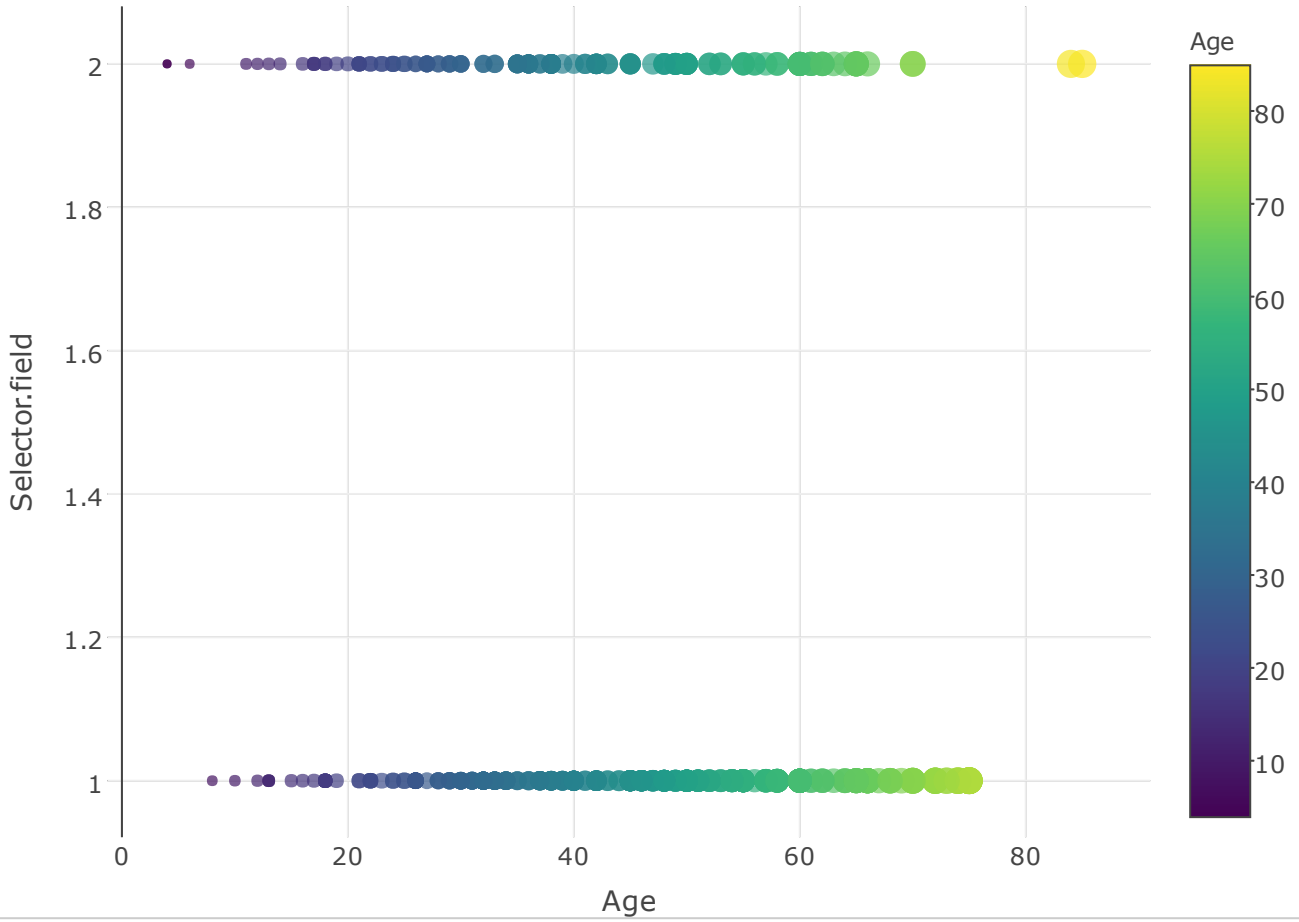
```
colMeans(colAUC(liver_normal[,-10], liver_normal[,10]))
```

```
##       Age        TB        DB    Alkphos      Sgpt      Sgot        TP
## 0.5827024 0.6932793 0.6860750 0.6744660 0.6855856 0.6972161 0.5205622
##       ALB A.G.Ratio
## 0.6065897 0.6190264
```

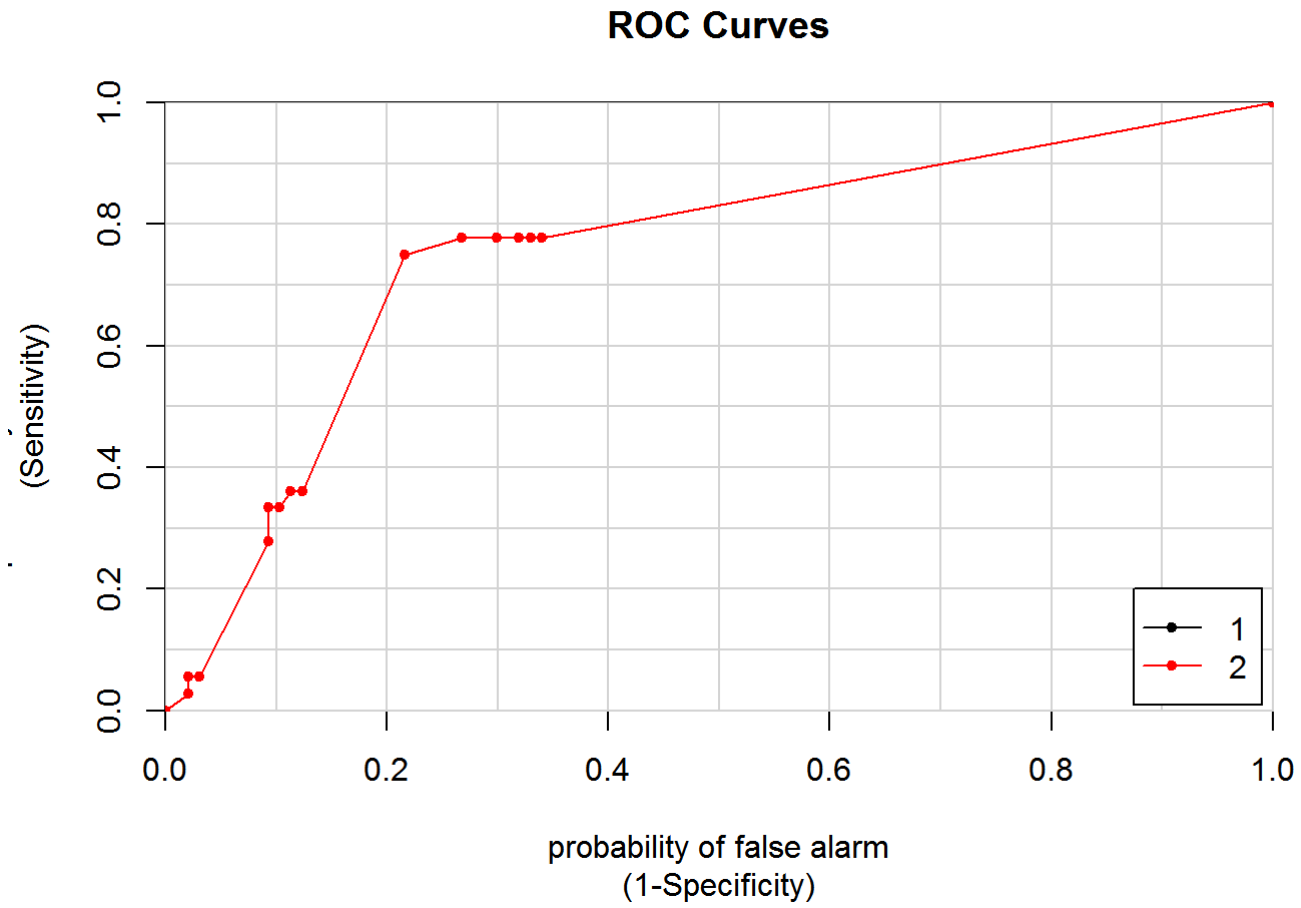
```
# Plot patients in there age groups
set.seed(100)
d <- data.liver[sample(nrow(data.liver), 500), ]
plot_ly(d, x = ~Age, y = ~Selector.field, color = ~Age,
        size = ~Age, text = ~paste("A.G.Ratio: ", A.G.Ratio))
```

```
## No trace type specified:
##   Based on info supplied, a 'scatter' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#scatter
```

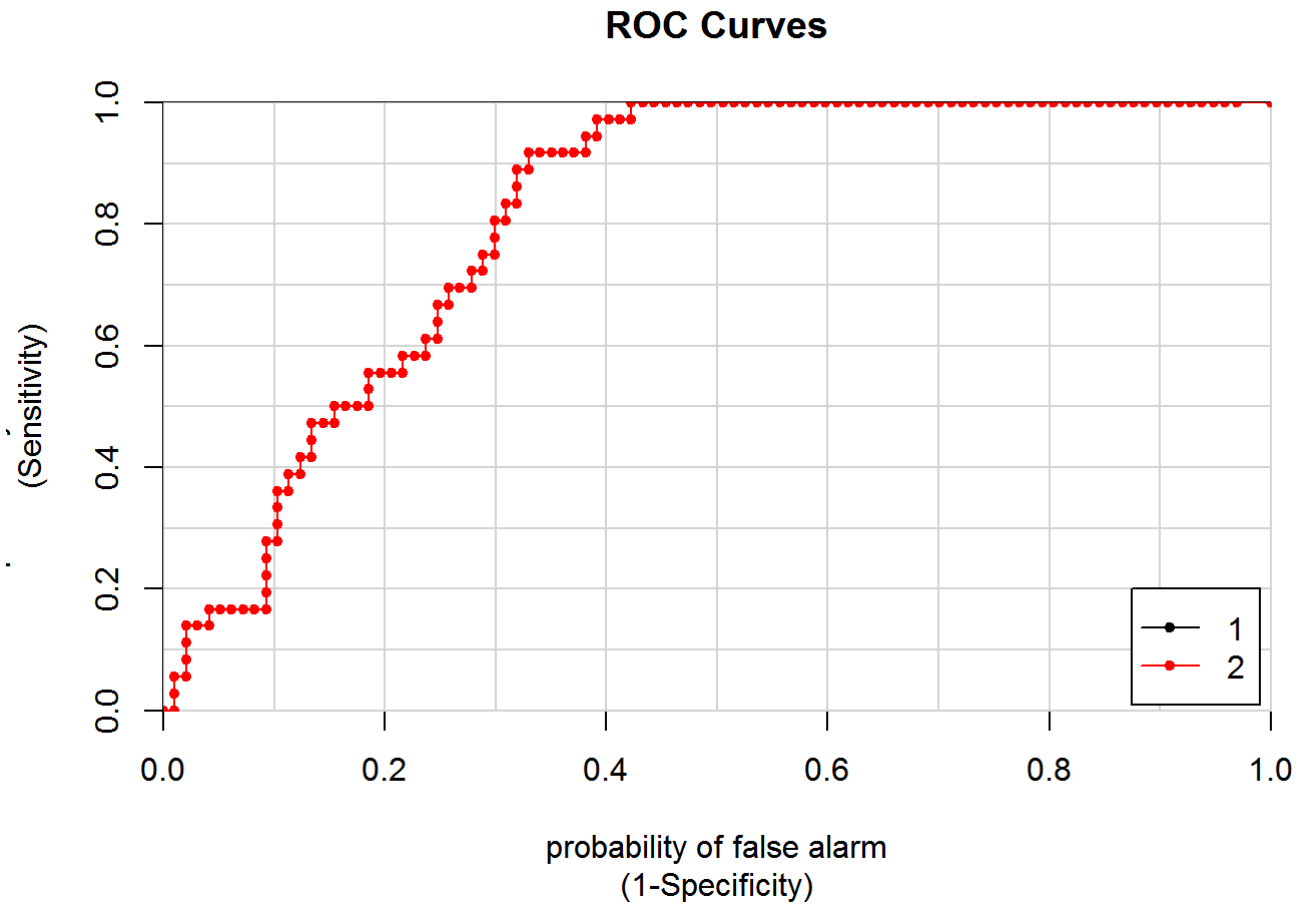
```
## No scatter mode specifed:
##   Setting the mode to markers
##   Read more about this attribute -> https://plot.ly/r/reference/#scatter-mode
```



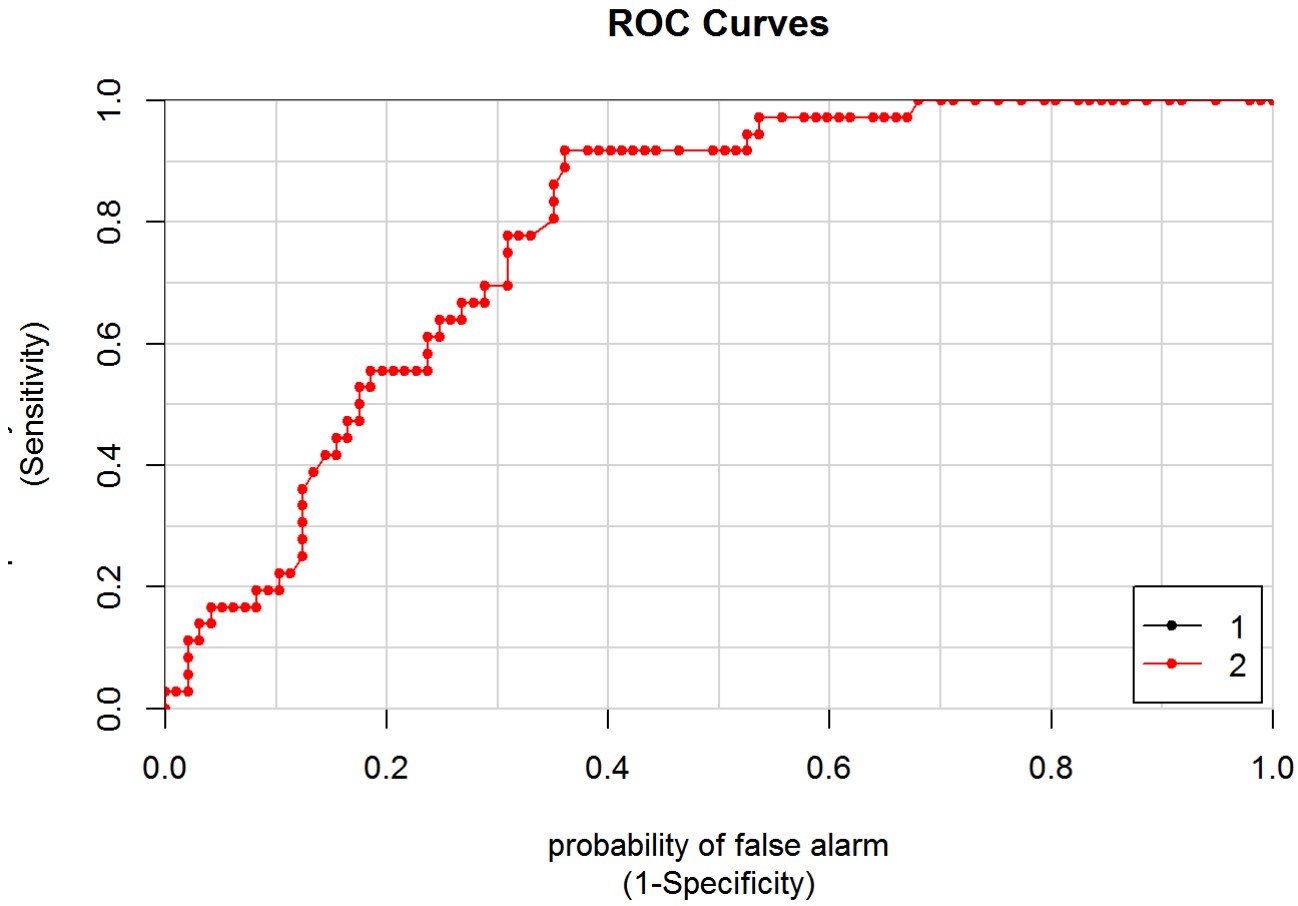

```
#--------------------- ROC Curve for C5.0 Algorithm ----------------------------#
c5.0_predict <- predict(credit_boost10,testc50,type="prob")
c5.0_ROC <- colAUC(c5.0_predict,testc50$Selector.field,
                   plotROC = TRUE,
                   alg=c("Wilcoxon","ROC"))
```




```
#--------------------- ROC curve for Naive bayes Algorithm ----------------------#
test_nb <- liver_normal[451:583,]
nb_predict <- predict(bayes_model,test_nb,type="raw")
nb_ROC <- colAUC(nb_predict,test_nb$Selector.field,
                 plotROC = TRUE,
                 alg=c("Wilcoxon","ROC"))
```




```
#--------------------- ROC curve for Random Forest -----------------------------#
rf_predict <- predict(random,test_rf,type="prob")
rf_ROC <- colAUC(rf_predict,test_rf$Selector.field,
                 plotROC = TRUE,
                 alg=c("Wilcoxon","ROC"))
```


ROC Curves
(Sensitivity)
0.0
0.2
0.4
0.6
0.8
1.0
1
2
probability of false alarm
(1-Specificity)

Chapter 7

Results

As we have applied Six different algorithms on the Liver Patient dataset taken from **UCI Repository**. On Some calculations these Algorithms provides **Accuracy**, **Precision**, and **Recall** values of the Implementation process, Which are tabulated as :

Classification Algorithm	Accuracy	Precision	Recall
KNN	70.68%	73.38%	93.81%
C5.0	71.43%	93.81%	73.98%
K-MEANS	69.47%	97.36%	70.80%
NAÏVE-BAYES	66.17%	1.00%	53.61%
RANDOM FOREST	72.18%	75.00%	92.78%
C5.0 with adaptive Boosting	75.19%	90.72%	78.57%

Conclusion

Finally, as the results in table shows that highest accuracy in given by the **Random forest** algorithm when applied all algorithms without any changes. But after adaptive boosting the **C5.0** algorithm it gives a more accurate result. The accuracy gained by our work is greater than the research paper which we have taken in study.

Chapter 8

Future Scope

8.1 Future scope on this Project

- More Algorithms can be applied on the same dataset to get good results.

- Combination of two or more algorithms of data mining are very helpful in determining the best results. This combination of algorithms is known as Hybrid Algorithm.

- Also by making clissifiers in present algorithms and applying new upcoming algorithms may be helpful for improvement.

- Choosing only those attributes which most affect the liver if slight change in quantity occurs in body. And applying Different approaches may improve the outcomes. This can be done by a lot of research on the liver patients, and doing surveys time to time while working in this topic.

REFERENCES

1. **Sina Bahramirad, Aida Mustapha, Maryam Eshraghi,** Classification of Liver Disease Diagnosis: A Comparative Study. Faculty of Computer Science and Information Technology, Universiti Putra Malaysia 0 ©2013 IEEE.

2. **B. V. Ramana, M. S. P. Babu and N. B. Venkateswarlu**, Liver Classification Using Modified Rotation Forest. International Journal of Engineering Research and Development, 6(1):17-24, 2012.

3. **R. H. Lin**, An Intelligent Model for Liver Disease Diagnosis. Artificial Intelligence in Medicine, 47(1):53-62, 2009.

4. **P. Rajeswari and G. Reena**, Analysis of Liver Disorder using Data mining Algorithm. Global Journal of Computer Science and Technology, 10(14):48-52, 2010.

5. **B. Chen and H. X. Zhang**, An Approach of Multiple Classifiers Ensemble based on Feature Selection. In Fifth International Conference on Fuzzy Systems and Knowledge Discovery, Vol. 2, pp. 390-394, 2008.

6. **S. Karthik, A. Priyadarishini, J. Anuradha and B. K. Tripathy**, Classification and Rule Extraction using Rough Set for Diagnosis of Liver Disease and Its Types. Advanced Applied Scientific Research, 2(3):334345, 2011.

7. **S. Ansari, I. Shafi, A. Ansari, J. Ahmad and S. I. Shah**, Diagnosis of Liver Disease Induced by Hepatitis Virus using Artificial Neural Networks. In 2011 IEEE International Multitopic Conference (INMIC), pp. 8-12, 2011.

8. **A.Frank and A. Asuncion**, UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, 2010.

9. **ILPD (Indian Liver Patient Dataset) Dataset**. UCI Repository of Machine Learning Databases. Available in http://archive.ics.uci.edu/ml/datasets/ILPD, last accessed: 15 December 2012.

10. **BUPA Liver Disorders Dataset**. UCI Repository of Machine Learning Databases. Available in http://archive.ics.uci.edu/ml/datasets/Liver+Disorders, last accessed: 15 December 2012.

11. **I Mierswa, M. Wurst, R. Klinkenberg, M. Scholz and T. Euler, YALE**: Rapid Prototyping for Complex Data Mining Tasks. In Proceedings of the 12th ACM

SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2006), pp. 935-940, 2006.

12. **A. Onisko, M. J. Druzdzel, and H. Wasyluk**, A Bayesian Network Model for Diagnosis of Liver Disorders. In Proceedings of the Eleventh Conference on Biocybernetics and Biomedical Engineering, Vol. 2, pp. 842-846, 1999.

13. **B. V. Ramana, M. S. P. Babu and N. B. Venkateswarlu**, A Critical Study of Selected Classification Algorithms for Liver Disease Diagnosis. International Journal of Database Management Systems, 3(2):101-114, 2011.

14. **B. V. Ramana, M. S. P. Babu and N. B. Venkateswarlu**, A Critical Evaluation of Bayesian Classifier for Liver Diagnosis using Bagging and Boosting Methods. International Journal of Engineering Science and Technology, 3(4):3422-3426, 2011.

15. **B. V. Ramana, M. S. P. Babu and N. B. Venkateswarlu**, A Critical Comparative Study of Liver Patients from USA and INDIA: An Exploratory Analysis. International Journal of Computer Science Issues, 9(3):506-516, 2012.