Shree Rahul Education Society's (Regd.)

SHREE L. R. TIWARI COLLEGE OF ENGINEERING

An Autonomous Institute Affiliated to University of Mumbai,

Approved by AICTE & DTE, Maharashtra State | NAAC Accredited, NBA Accredited Program | ISO 9001:2015 Certified | DTE Code No: 3423 | Recognized under Section 2(f) of the UGC Act 1956 | Minority Status (Hindi Linguistic)

DEPARTMENT OF COMPUTER ENGINEERING

CSL503 Data Warehousing & Mining Lab

Fifth Semester, 2025-2026 (Odd Semester)

Name of Student :

Roll No.

Batch :

Venue :

Experiment No.

Title of Experiment

Date of Conduction :

Date of Submission :

Particulars	Max. Marks	Marks Obtained
Preparedness and Efforts (PE)	**3**	
Knowledge of Tools (KT)	**3**	
Debugging and Results (DR)	**3**	
Documentation (DN)	**3**	
Punctuality & Lab Ethics (PL)	**3**	
Total	**15**	

Grades – Meet Expectations (3 Marks), Moderate Expectations (2 Marks), Below Expectations (1 Mark)

Checked and verified by

Name of Faculty :

Signature :

Date :

EXPERIMENT NO. 1

TITLE: One case study on building Data warehouse/Data Mart

AIM: Write Detailed Problem statement and design dimensional modelling (creation of star and snowflake schema)

THEORY:

Dimensional Modelling:

It is a data structure technique optimized for data storage in a Data warehouse. The purpose of dimensional modeling is to optimize the database for faster retrieval of data. The concept of Dimensional Modelling was developed by Ralph Kimball and consists of "fact" and "dimension" tables. A dimensional model in data warehouse is designed to read, summarize, analyze numeric information like values, balances, counts, weights, etc. in a data warehouse.

For instance, in the relational mode, normalization and ER models reduce redundancy in data. On the contrary, dimensional model in data warehouse arranges data in such a way that it is easier to retrieve information and generate reports.

Elements of Dimensional Data Model:

1) **Fact**:
   Facts are the measurements/metrics or facts from your business process. For a sales business process, a measurement would be quarterly sales number.

2) **Dimension:**
   Dimension provides the context surrounding a business process event. In simple terms, they give who, what, where of a fact. In the Sales business process, for the fact quarterly sales number, dimensions would be
   - Who – Customer Names
   - Where – Location
   - What – Product Name

   In other words, a dimension is a window to view information in the facts.

3) **Attributes:**
   The Attributes are the various characteristics of the dimension in dimensional data modeling. In the Location dimension, the attributes can be
   - State
   - Country
   - Zip code etc.

4) **Fact Table:**
A fact table is a primary table in dimension modeling Fact Table contains

- Measurements/facts
- Foreign key to dimension table

5) **Dimension Table:**
A dimension table contains dimensions of a fact. They are joined to fact table via a foreign key. Dimension tables are de-normalized tables. The Dimension Attributes are the various columns in a dimension table Dimensions offers descriptive characteristics of the facts with the help of their attributes No set limit set for given for number of dimensions The dimension can also contain one or more hierarchical relationships.

Steps of Dimensional Modelling

The accuracy in creating your Dimensional modeling determines the success of your data warehouse implementation. Here are the steps to create Dimension Model The model should describe the Why, How much, When/Where/Who and What of your business process.

Step 1) Identify the Business Process-Identifying the actual business process a data warehouse should cover. This could be Marketing, Sales, HR, etc. as per the data analysis needs of the organization. The selection of the Business process also depends on the quality of data available for that process. It is the most important step of the Data Modelling process, and a failure here would have cascading and irreparable defects.

Step 2) Identify the Grain-The Grain describes the level of detail for the business problem/solution. It is the process of identifying the lowest level of information for any table in your data warehouse. If a table contains sales data for every day, then it should be daily granularity. If a table contains total sales data for each month, then it has monthly granularity.

During this stage, you answer questions like:

Do we need to store all the available products or just a few types of products? This decision is based on the business processes selected for Data warehouse

Do we store the product sale information on a monthly, weekly, daily or hourly basis? This decision depends on the nature of reports requested by executives

How do the above two choices affect the database size?

Example of Grain: -The CEO at an MNC wants to find the sales for specific products in different locations on a daily basis. So, the grain is "product sale information by location by the day."

Step 3) Identify the Dimensions-Dimensions are nouns like date, store, inventory, etc. These dimensions are where all the data should be stored. For example, the date dimension may contain data like a year, month and weekday.

Example of Dimensions: -The CEO at an MNC wants to find the sales for specific products in different locations on a daily basis.

Dimensions: Product, Location and Time

Attributes: For Product: Product key (Foreign Key), Name, Type, Specifications

Hierarchies: For Location: Country, State, City, Street Address, Name

Step 4) Identify the Fact-This step is co-associated with the business users of the system because this is where they get access to data stored in the data warehouse. Most of the fact table rows are numerical values like price or cost per unit, etc.

Example of Facts: -The CEO at an MNC wants to find the sales for specific products in different locations on a daily basis. The fact here is Sum of Sales by product by location by time.

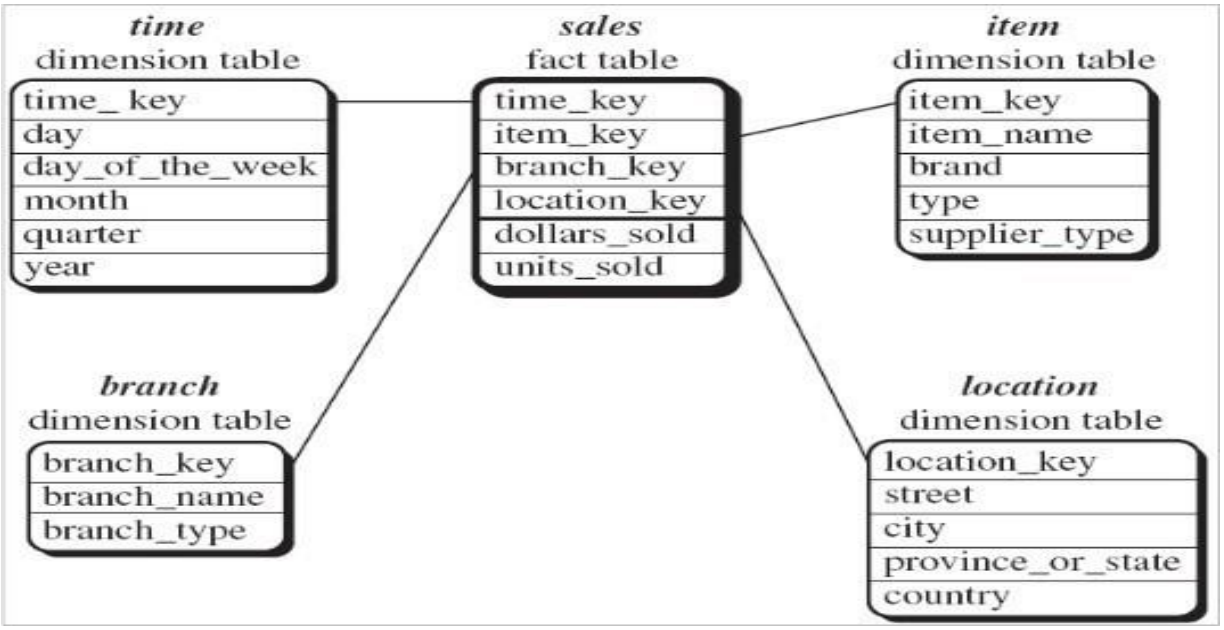
Step 5) Build Schema-In this step, you implement the Dimension Model. A schema is nothing but the database structure (arrangement of tables). There are two popular schemas

- Star Schema
- Snowflake Schema

Star Schema:

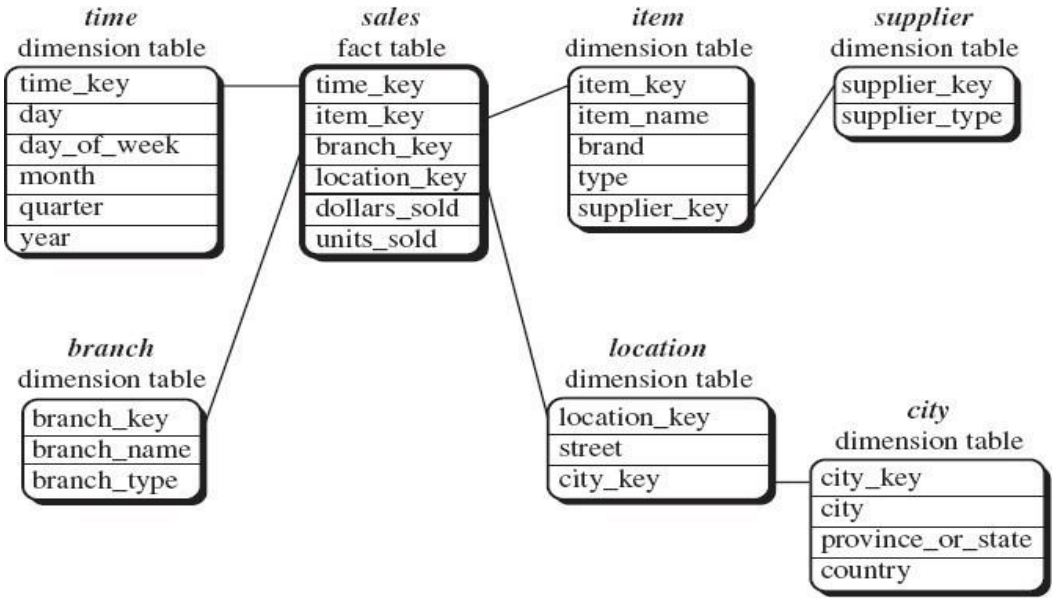
A dimensional model with fact table in the middle and dimension tables arranged around the fact table is called a star schema. Here, the fact table is at the core of the star and the dimension tables are along the spikes of the star. In this arrangement every attribute in the dimension table has the even chance to be participated in a query to analyze that attribute. Each dimension table is related to the fact table in a one-to- many relationship.

Star schema structure shows how the users normally view their metrics along their business dimensions. It can answer questions of what, when, by whom and whom etc. The answers are produced by combining one or more-dimension tables with the fact table. The relationship of a particular row in the fact tables with the rows in each dimension table. These relationships are shown as the spikes of the Star schema. Example: Star Schema for sales data



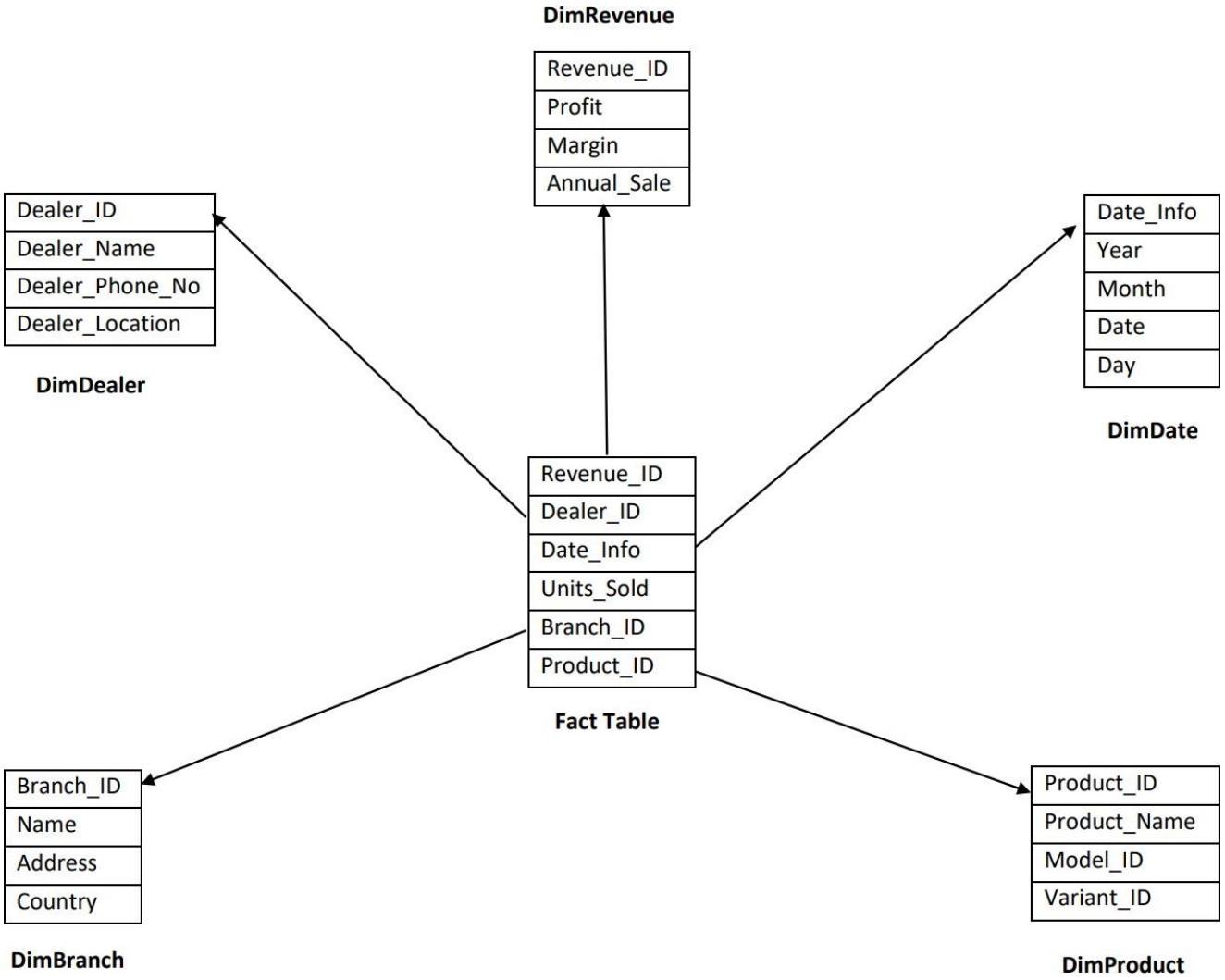
Snowflake schema:

A refinement of star schema where some dimensional hierarchy is further splitting (normalized) into a set of smaller dimension tables, forming a shape similar to snowflake. However, the snowflake structure can reduce the effectiveness of browsing, since more joins will be needed. The snowflake schema is a more complex data warehouse model than a star schema, and is a type of star schema. It is called a snowflake schema because the diagram of the schema resembles a snowflake.

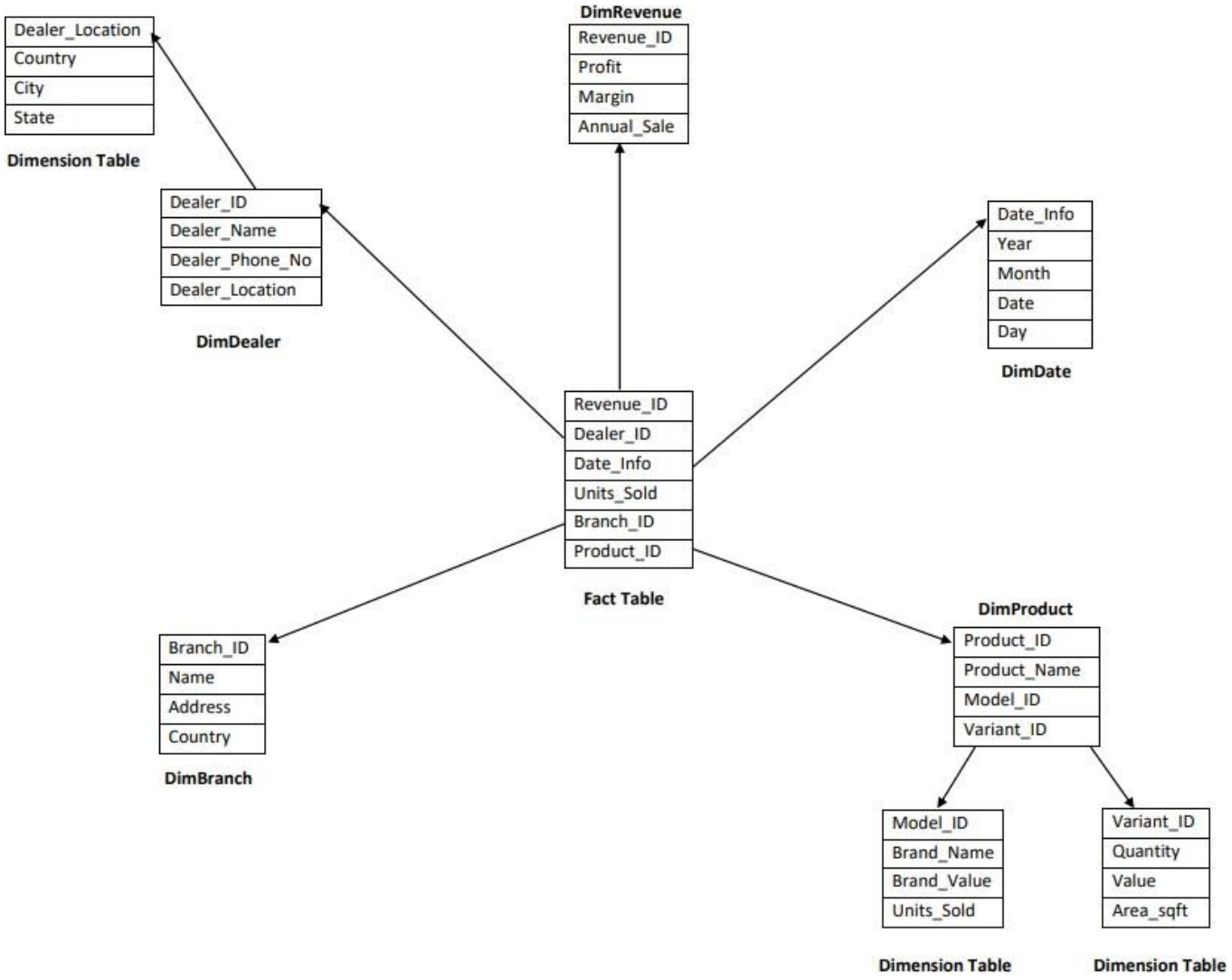


LAB EXERCISE: Students will select one case study on building Data warehouse/Data Mart. Write Detailed Problem statement and design dimensional modelling (creation of star and snowflake schema)

Problem Statement: Case Study on Real Estate Management System for creating Data Warehouse/Data Mart



Star Schema



Snowflake Schema

CONCLUSION: Hence, we have understood the concept of dimensional modelling and we have created the dimensional modelling design using star schema and snowflake schema for the selected case study for building Data Ware house/Data Mart.

EXPERIMENT NO. 2

TITLE: Implementation of all Dimension Table and Fact Table

AIM: Write an SQL query to **i**mplement Dimension Table and Fact Table based on Experiment 1 Case Study.

THEORY:

Fact Table:

In data warehousing, a fact table consists of the measurements, metrics or facts of a business process. It is located at the center of a star schema or a snowflake schema surrounded by dimension tables. Where multiple fact tables are used, these are arranged as a fact constellation schema. A fact table typically has two types of columns: those that contain facts and those that are foreign keys to dimension tables. The primary key of a fact table is usually a composite key that is made up of all of its foreign keys. Fact tables contain the content of the data warehouse and store different types of measures like additive, non- additive, and semi additive measures.
Fact tables provide the (usually) additive values that act as independent variables by which dimensional attributes are analyzed. Fact tables are often defined by their grain. The grain of a fact table represents the most atomic level by which the facts may be defined. The grain of a SALES fact table might be stated as "Sales volume by Day by Product by Store". Each record in this fact table is therefore uniquely defined by a day, product and store. Other dimensions might be members of this fact table (such as location/region) but these add nothing to the uniqueness of the fact records. These "affiliate dimensions" allow for additional slices of the independent facts but generally provide insights at a higher level of aggregation (a region contains many stores).

Dimension Table:

In data warehousing, a dimension table is one of the set of companion tables to a fact table. The fact table contains business facts (or measures), and foreign keys which refer to candidate keys (normally primary keys) in the dimension tables.

Contrary to fact tables, dimension tables contain descriptive attributes (or fields) that are typically textual fields (or discrete numbers that behave like text). These attributes are designed to serve two critical purposes: query constraining and/or filtering, and query result set labeling.

Dimension attributes should be:

- Verbose (labels consisting of full words)
- Descriptive
- Complete (having no missing values)
- Discretely valued (having only one value per dimension table row)
- Quality assured (having no misspellings or impossible values)

Dimension table rows are uniquely identified by a single key field. It is recommended that the key field be a simple integer because a key value is meaningless, used only for joining fields between the fact and dimension tables.

OUTPUT:

Fact and Dimension Table of Star and Snowflake Schema

Fact Table:

```
CREATE TABLE fact_table (
    Revenue_ID int,
    Dealer_ID int,
    Branch_ID int,
    Unit_Sold int,
    Date_Info varchar(255),
    Product_ID int
);
```

- DimDate

```
CREATE TABLE DimDate (
    Date_Info varchar(255) FOREIGN KEY REFERENCES fact_table(Date_Info),
    Year_Info int,
    Month_Info varchar(255),
    Day_Info varchar(255)
);
```

- DimProduct

```
CREATE TABLE DimProduct (
    Product_ID int FOREIGN KEY REFERENCES fact_table(Product_ID),
    Product_Name varchar(255),
    Model_ID int,
    Variant_ID int
);
```

- DimBranch

```
CREATE TABLE DimBranch (
    Branch_ID int FOREIGN KEY REFERENCES fact_table(Branch_ID),
    Name_Info varchar(255),
    Address_Info varchar(255),
    Country varchar(255)
);
```

- DimDealer

```
CREATE TABLE DimDealer (
    Dealer_ID int FOREIGN KEY REFERENCES fact_table(Dealer_ID),
    Dealer_Name varchar(255),
    Dealer_Phone_No int,
    Dealer_Location varchar(255)
);
```

- DimRevenue

```
CREATE TABLE DimRevenue (
    Revenue_ID int FOREIGN KEY REFERENCES fact_table(Dealer_ID),
    Profit int,
    Margin varchar(255),
    Annual_Sale int
);
```

Values of the Fact Table:

Results | Messages

	Revenue_ID	Dealer_ID	Branch_ID	Unit_Sold	Date_Info	Product_ID
1	1	49	305	500	22-07-2022	1049
2	2	56	310	1000	22-07-2022	1056
3	3	35	310	1500	22-07-2022	1035

Values of Dimension Table:

- DimDate

Results | Messages

	Date_Info	Year_Info	Month_Info	Day_Info
1	22-07-2022	2022	July	Monday

- DimProduct

Results | Messages

	Product_ID	Product_Name	Model_ID	Variant_ID
1	1	Rustom	56	100

- DimBranch

Results Messages

	Branch_ID	Name_Info	Address_Info	Country
1	1001	Block-A	Rizvi Complex OFF carter Road	India

- DimDealer

Results Messages

	Dealer_ID	Dealer_Name	Dealer_Phone_No	Dealer_Location
1	69	Anuj	123456789	Mumbai
2	78	Valay	123499789	Mumbai

- DimRevenue

Results Messages

	Revenue_ID	Profit	Margin	Annual_Sale
1	101	150000	10%	400

CONCLUSION: Hence, we have created and displayed Fact Table & Dimension Table.

EXPERIMENT NO. 3

TITLE: One case study on building OLAP operations

AIM: Write Detailed Problem statement to implementation of OLAP operations: Slice, Dice, Rollup, Drilldown and Pivot based on experiment 1

THEORY:

Online Analytical Processing Server (OLAP) is based on the multidimensional data model. It allows managers, and analysts to get an insight of the information through fast, consistent, and interactive access to information. This chapter cover the types of OLAP, operations on OLAP, difference between OLAP, and statistical databases and OLTP.

Types of OLAP Servers

We have four types of OLAP servers −

- Relational OLAP (ROLAP)
- Multidimensional OLAP (MOLAP)
- Hybrid OLAP (HOLAP)
- Specialized SQL Servers

OLAP Operations

Since OLAP servers are based on multidimensional view of data, we will discuss OLAP operations in multidimensional data.

Here is the list of OLAP operations −

- Roll-up
- Drill-down
- Slice and dice
- Pivot (rotate)

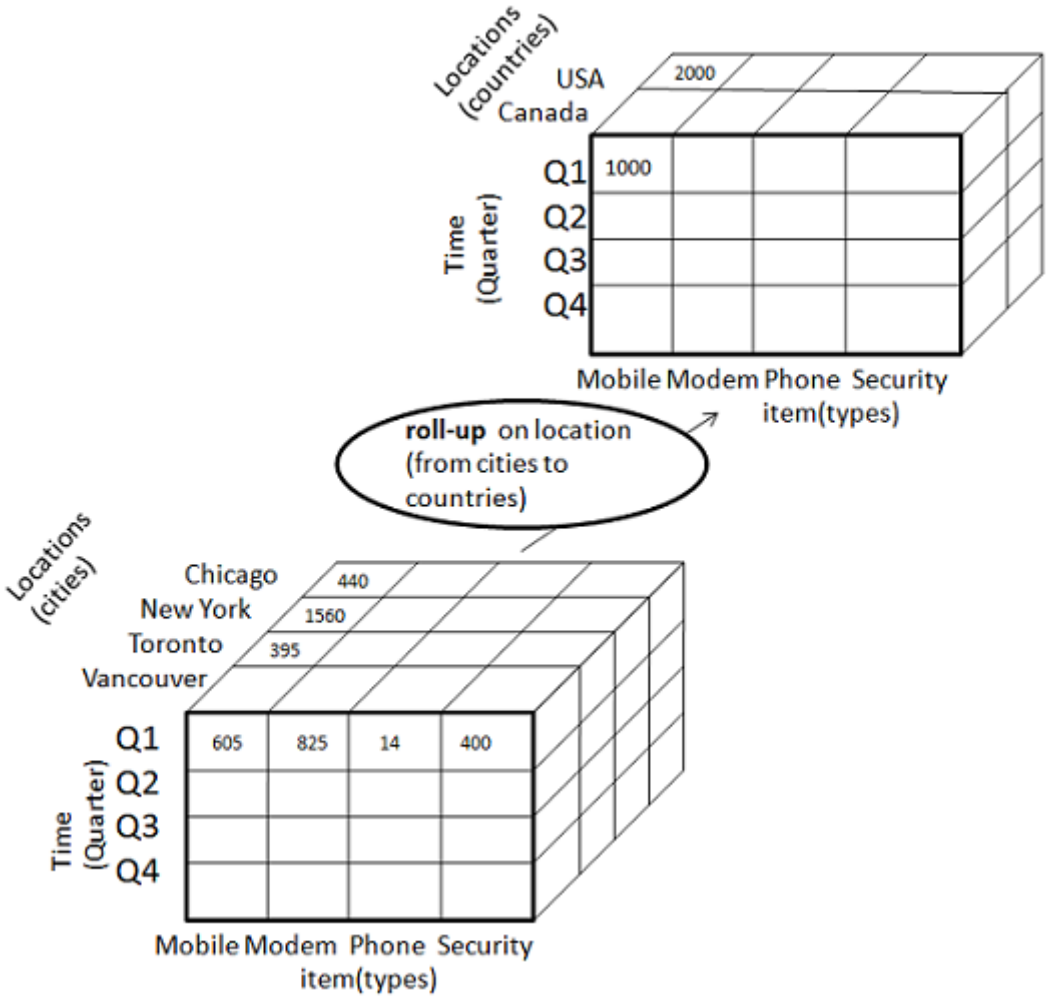
Roll-up

Roll-up performs aggregation on a data cube in any of the following ways −

- By climbing up a concept hierarchy for a dimension

- By dimension reduction

The following diagram illustrates how roll-up works.



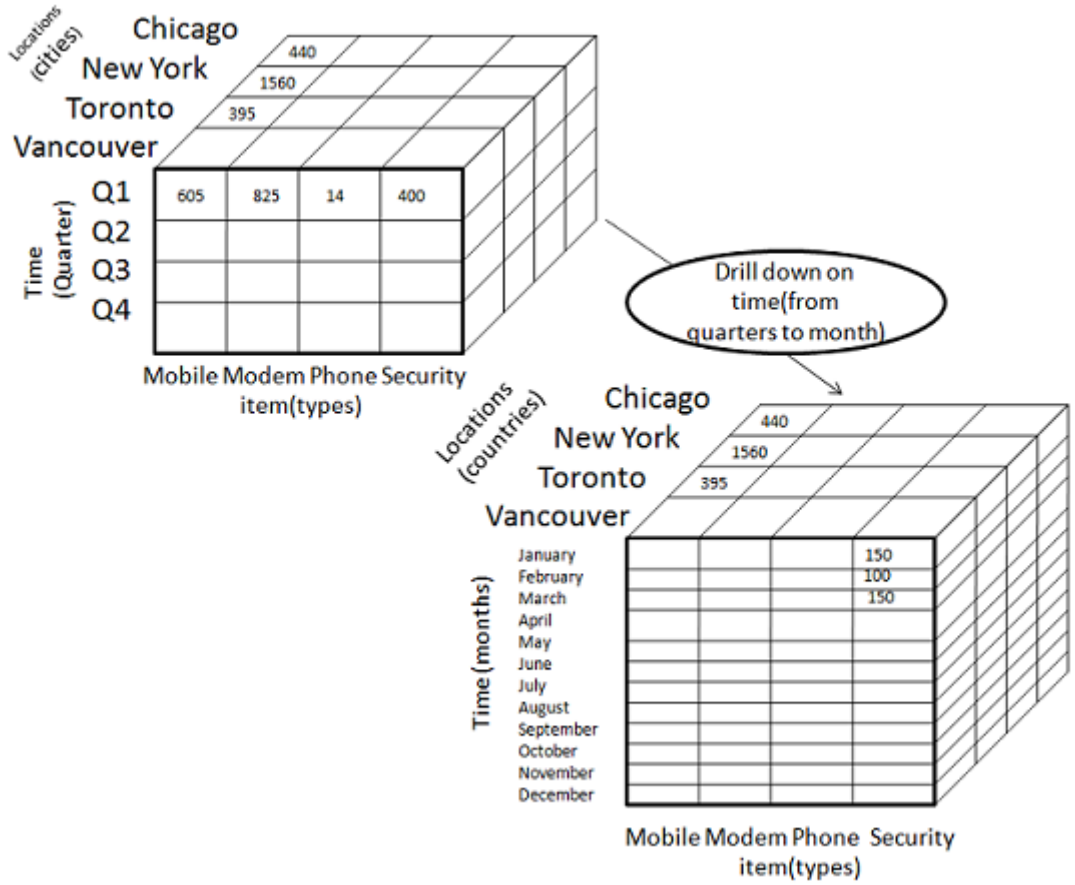
- Roll-up is performed by climbing up a concept hierarchy for the dimension location.
- Initially the concept hierarchy was "street < city < province < country".
- On rolling up, the data is aggregated by ascending the location hierarchy from the level of city to the level of country.
- The data is grouped into cities rather than countries.
- When roll-up is performed, one or more dimensions from the data cube are removed.

Drill-down

Drill-down is the reverse operation of roll-up. It is performed by either of the following ways −

- By stepping down a concept hierarchy for a dimension

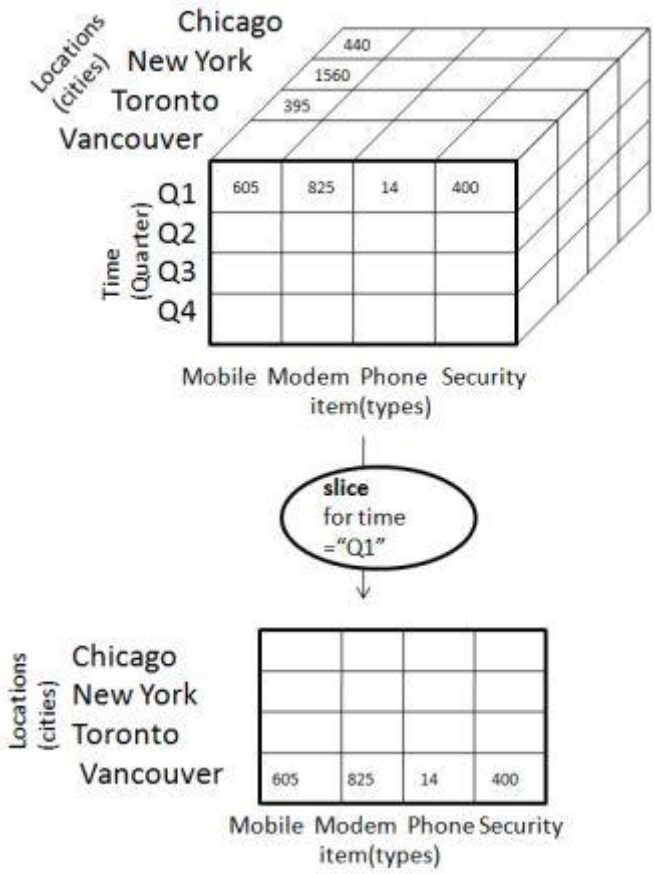
- By introducing a new dimension.



- Drill-down is performed by stepping down a concept hierarchy for the dimension time.
- Initially the concept hierarchy was "day < month < quarter &lt year."
- On drilling down, the time dimension is descended from the level of quarter to the level of month.
- When drill-down is performed, one or more dimensions from the data cube are added.
- It navigates the data from less detailed data to highly detailed data.

Slice

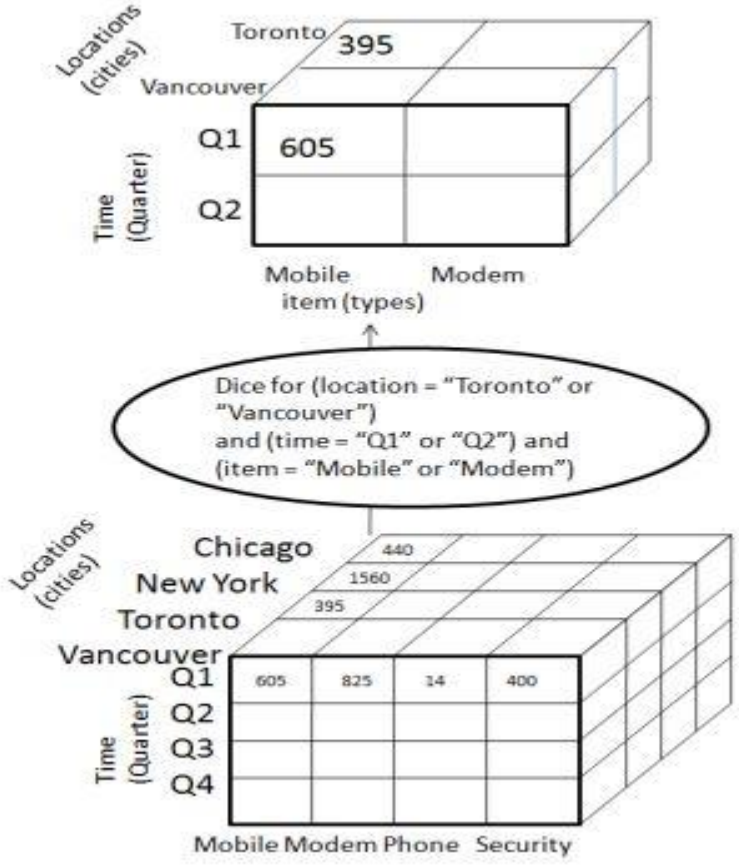
The slice operation selects one particular dimension from a given cube and provides a new sub-cube. Consider the following diagram that shows how slice works.



- Here Slice is performed for the dimension "time" using the criterion time = "Q1".
- It will form a new sub-cube by selecting one or more dimensions.

Dice

Dice selects two or more dimensions from a given cube and provides a new sub-cube. Consider the following diagram that shows the dice operation.

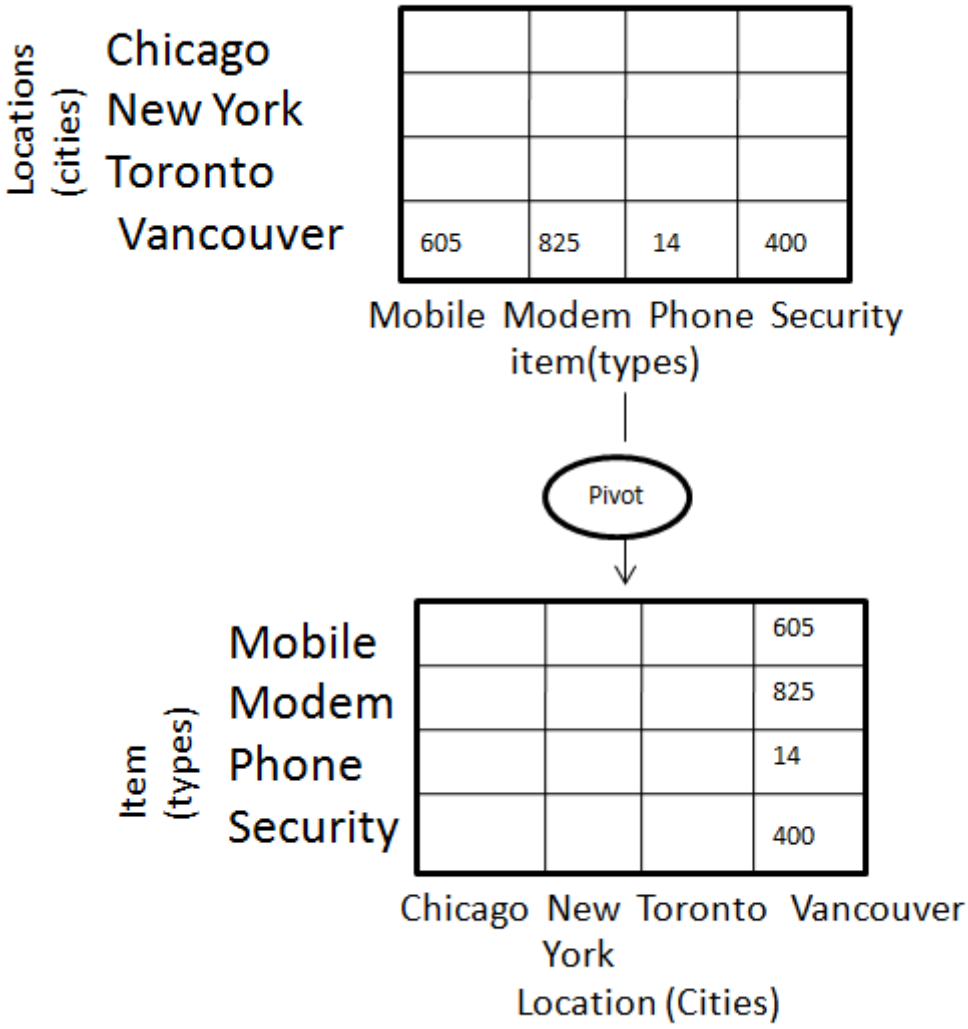


The dice operation on the cube based on the following selection criteria involves three dimensions.

- (location = "Toronto" or "Vancouver")
- (time = "Q1" or "Q2")
- (item =" Mobile" or "Modem")

Pivot

The pivot operation is also known as rotation. It rotates the data axes in view in order to provide an alternative presentation of data. Consider the following diagram that shows the pivot operation.



CONCLUSION: Hence, we have studied building different OLAP operations.

EXPERIMENT NO. 4

TITLE: Study on Data Mining tools like WEKA & XLMiner.

AIM: Write a program to implement Naive Bayes classifier using WEKA tool

THEORY:

WEKA:

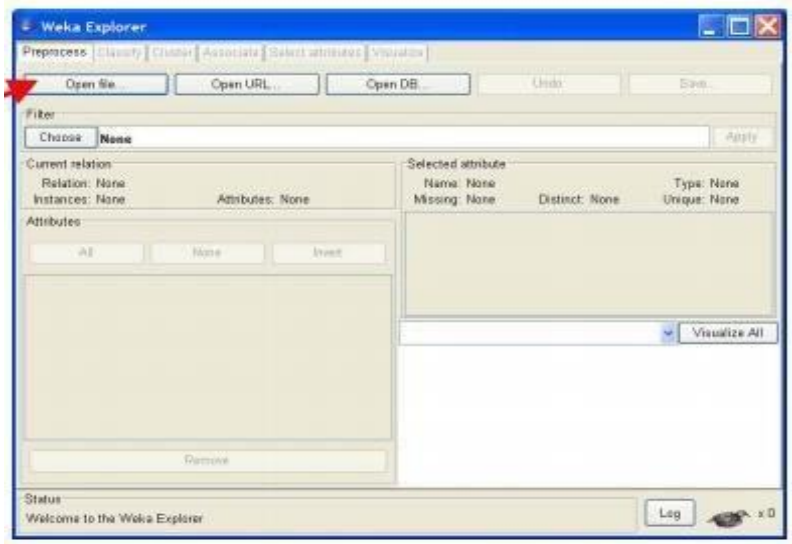
WEKA is a data mining system developed by the University of Waikato in New Zealand that implements data mining algorithms. WEKA is a state-of-the-art facility for developing machine learning (ML) techniques and their application to real-world data mining problems. It is a collection of machine learning algorithms for data mining tasks. The algorithms are applied directly to a dataset. WEKA implements algorithms for data preprocessing, classification, regression, clustering, association rules; it also includes a visualization tools. The new machine learning schemes can also be developed with this package. WEKA is open source software issued under the GNU General Public License. The goal of this Tutorial is to help you to learn WEKA Explorer. The tutorial will guide you step by step through the analysis of a simple problem using WEKA Explorer preprocessing, classification, clustering, association, attribute selection, and visualization tools. At the end of each problem there is a representation of the results with explanations side by side. Each part is concluded with the exercise for individual practice. By the time you reach the end of this tutorial, you will be able to analyze your data with WEKA Explorer using various learning schemes and interpret received results.:

WEKA is a collection of tools for:

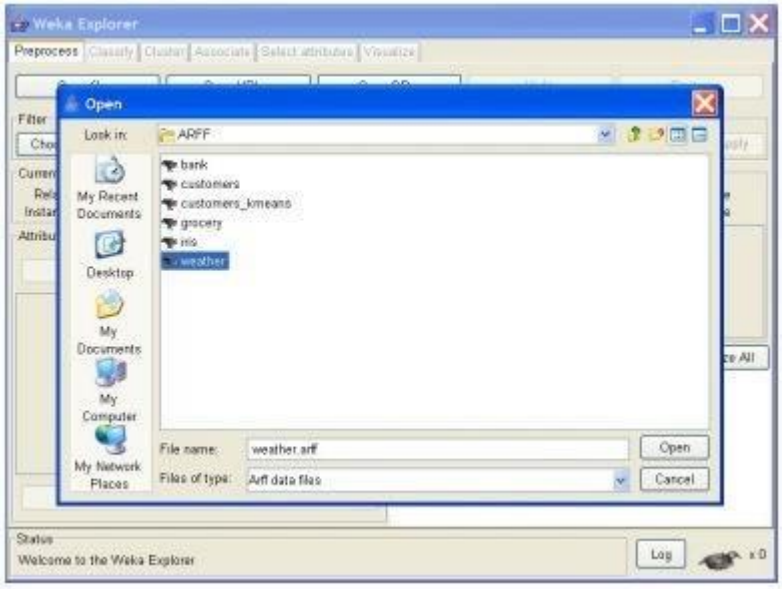
- Regression
- Clustering
- Association
- Data pre-processing
- Classification
- Visualization

Opening file from a local file system:

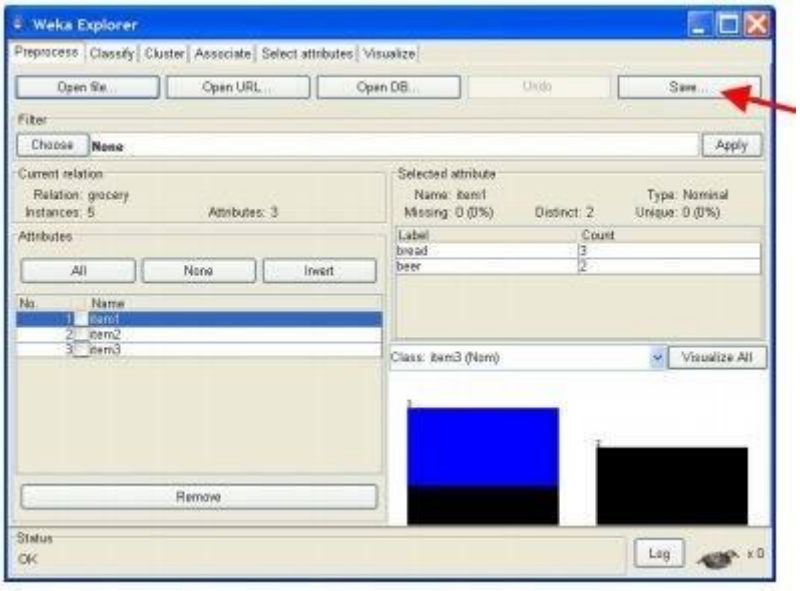
Click on 'Open file…' button.



It brings up a dialog box allowing you to browse for the data file on the local file system, choose "FileName.arff" file.



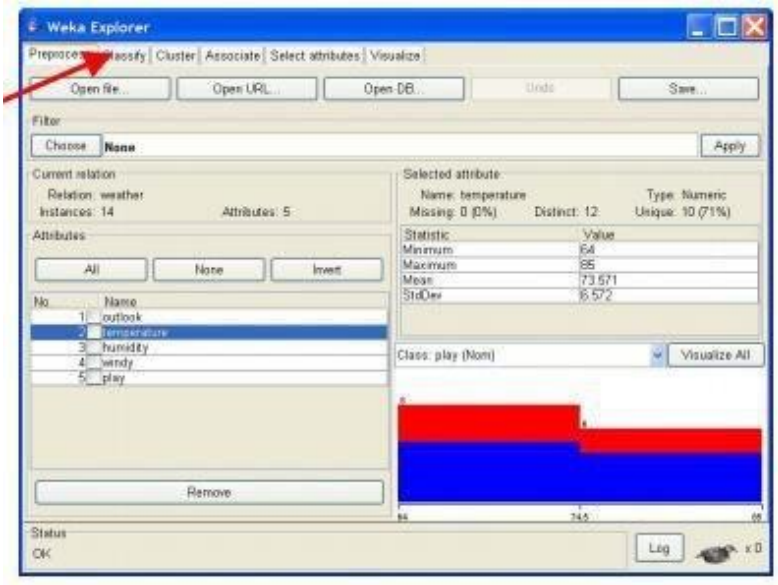
Some databases have the ability to save data in CSV format. In this case, you can select CSV file from the local filesystem. If you would like to convert this file into ARFF format, you can click on 'Save' button. WEKA automatically creates ARFF file from your CSV file.



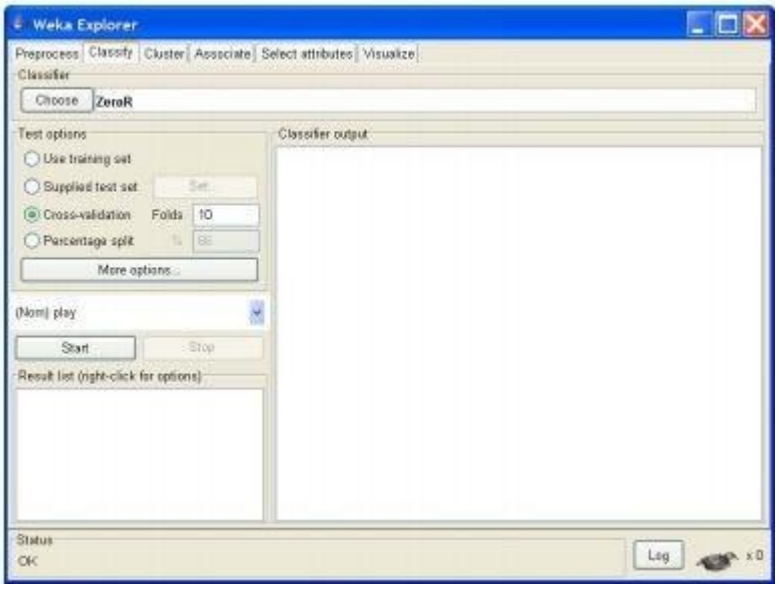
Building "Classifiers":

Classifiers in WEKA are the models for predicting nominal or numeric quantities. The learning schemes available in WEKA include decision trees and lists, instance-based classifiers, support vector machines, multi-layer perceptrons, logistic regression, and bayes' nets. "Meta"- classifiers include bagging, boosting, stacking, error-correcting output codes, and locally weighted learning.

Once you have your data set loaded, all the tabs are available to you. Click on the 'Classify' tab.



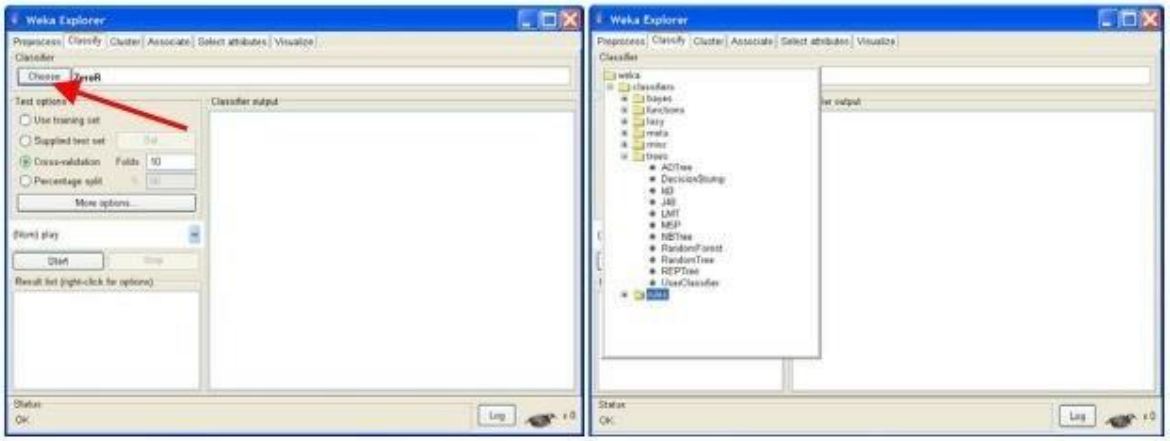
'Classify' window comes up on the screen.



Now you can start analyzing the data using the provided algorithms. In this exercise you will analyze the data with C4.5 algorithm using J48, WEKA's implementation of decision tree learner. The sample data used in this exercise is the weather data from the file "weather.arff". Since C4.5 algorithm can handle numeric attributes, in contrast to the ID3 algorithm from which C4.5 has evolved, there is no need to discretize any of the attributes. Before you start this exercise, make sure you do not have filters set in the 'Preprocess' window. Filter exercise in section 3.6 was just a practice.

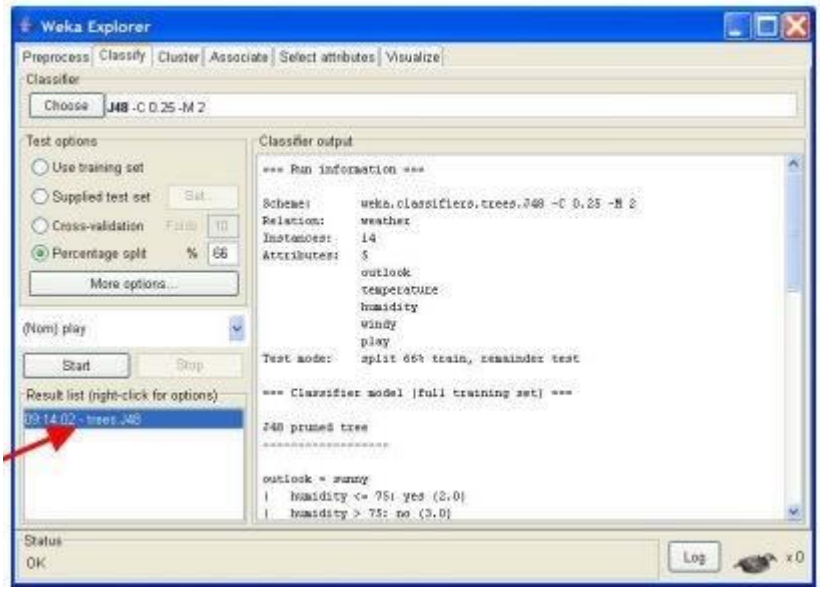
Choosing a Classifier:

Click on 'Choose' button in the 'Classifier' box just below the tabs and select C4.5 classifier WEKA Æ Classifiers Æ Trees Æ J48.

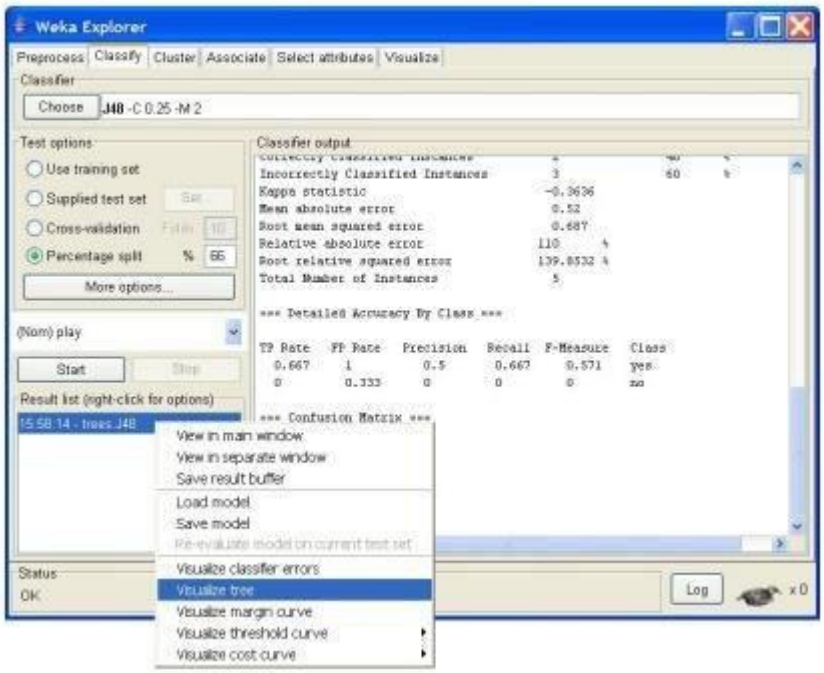


Visualization of Results:

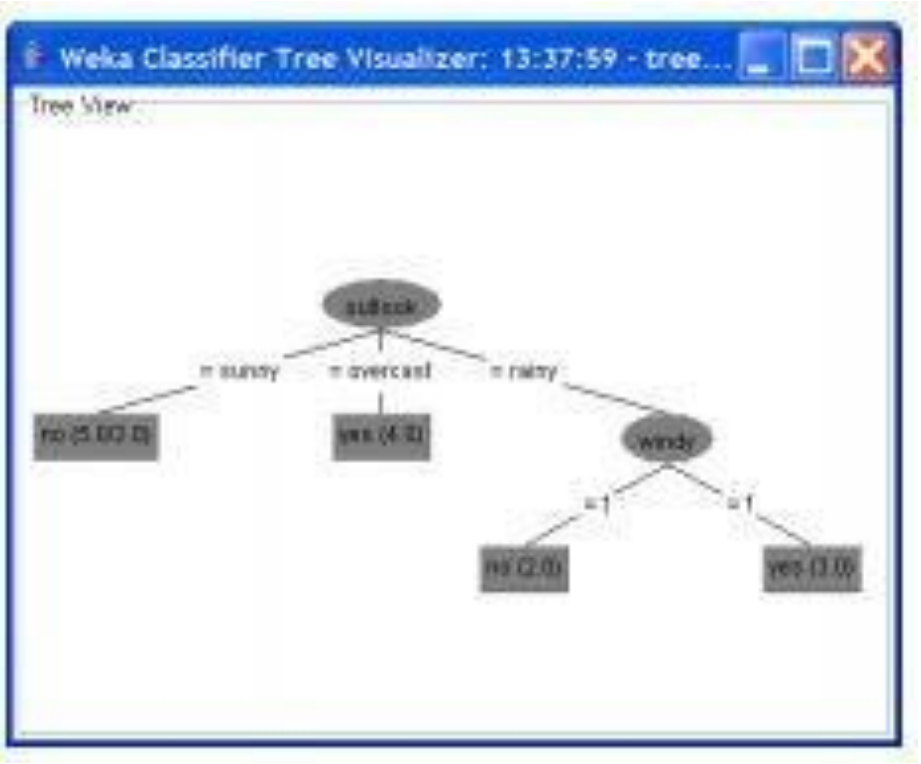
After training a classifier, the result list adds an entry.



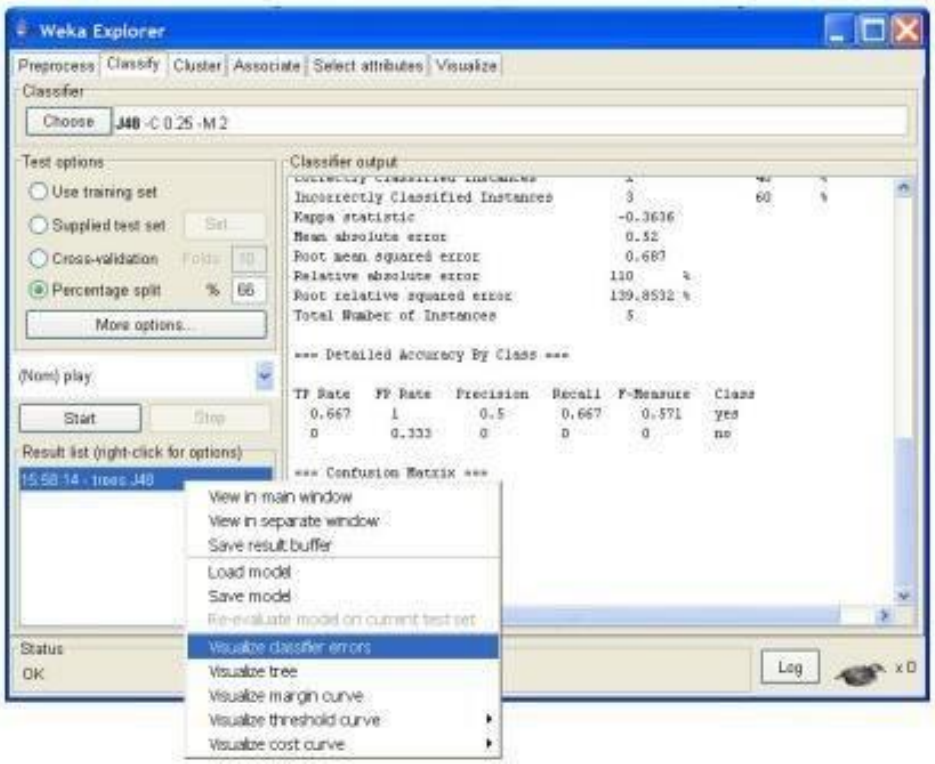
WEKA lets you to see a graphical representation of the classification tree. Right-click on the entry in 'Result list' for which you would like to visualize a tree. It invokes a menu containing the following items:



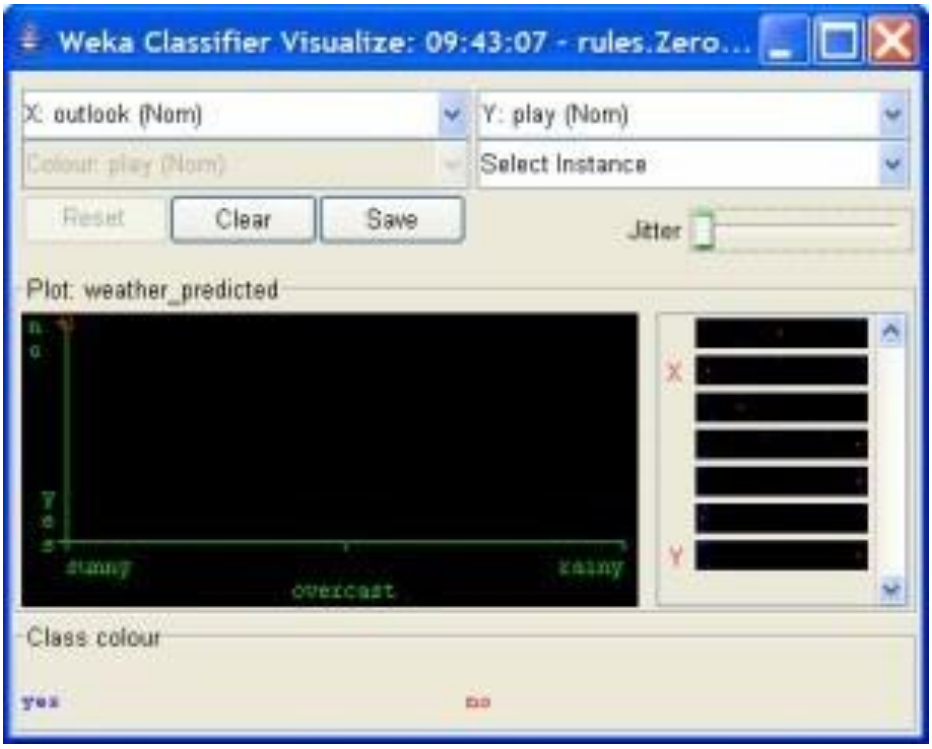
Select the item 'Visualize tree'; a new window comes up to the screen displaying the tree.



WEKA also lets you to visualize classification errors. Right-click on the entry in 'Result list' again and select 'Visualize classifier errors' from the menu:



Weka Classifier Visualize' window displaying graph appears on the screen.



XLMiner:

Data mining is a set of methods that helps an expert to obtain new, previously unknown significant information from available data. To apply data mining possibilities practically software that deploys data mining algorithms is needed. This software may also be integrated into other tools. One of these tools is the QuantLink XLMiner tool for Microsoft Excel. In this paper the data mining possibilities of XLMiner are examined. More precisely – this tool is used for extracting unknown client data, rules and patterns, as well as for extracting information that is significant for client relationship management.

Forecasting client activity class:

XLMiner offers to perform data sampling and partition using a data set that holds up to 60 000 records and 200 attributes. This includes data sampling to reduce the size of a data set (numbers of records) by choosing a subset of records that would represent the whole set. It can be done by random partitioning or partitioning with oversampling setting a desired support limit for a class.

Data set is larger than the limitations set by MS Excel or XLMiner; for this reason random partitioning was used to reduce the number of records. But the resulting data set would still have more attributes than it is allowed in this tool (30 attributes are allowed). Therefore a smaller subset of the attributes is needed, that doesn't lose much information in the reduction process. Therefore a tool named WEKA was used to pick a subset of 13 best attributes.

The data set was also divided into training and test data sets. Usually data sets are divided into training set holding 70% of the data and test set holding 30% of the data. This ratio was also used in this case study

wherever such division was necessary (e.g. classification and forecasting). XLMiner offers solving a classification task using the following methods: discriminant analysis, logistic regression (not to be confused with logic regression, which works with binary data and classifies data using the Boolean algebra approach), classification trees (CART), naive Bayes classifier, neural networks (multilayer feedforward architectures) and k-nearest neighbours. Discriminant analysis does not work with the needed amount of data and logistic regression does not support 5 classes in the data in this tool (it only classifies 2 classes), therefore this task was solved using other methods and attributes of data types that are supported by these methods (e.g. continuous data was not used with naive Bayes classifier). The best method for this task was chosen using the total classification error and the wrong clasifications into most important class (that is class 4) (see Table 1). Each client was assigned an activity class which shows how long a client will participate in the lottery, which means we can forecast whether this client will be a loyal customer and for how long he will participate). We consider the following classes that show 5 common patterns of client behaviour:

"0" no tickets were bought,

"1" only the first ticket was bought,

"2" no more than two tickets were bought,

"3" client bought all tickets for one season,

"4" client played during the first season and bought tickets for the second season (We consider this type of people 'loyal customers').

The importance of this task is its contribution to forecasting client behaviour for the company, it helps to forecast the number of players, money flow etc. If there is information about prospective customers (that are not clients of this lottery yet), it can also provide information for selective advertising campaigns, distributing the advertisements to prospective customers that will most likely become loyal customers.

An analysis of the classifiers' results enables us to evaluate the performance of the classifiers. It was done by evaluating the total classification error and the rates of the wrong classifications into each class (Table 1).

Classification errors (%)

Class	Classification tree	Naïve Bayes	*K*-nearest neighbours	Neural network
0	38.85	39.57	42.34	23.64
1	**100.00**	100.00	99.33	**100.00**
2	**100.00**	99.25	97.26	**100.00**
3	**100.00**	48.48	78.89	**100.00**
4	13.67	33.48	26.89	29.07
Total	45.44	47.75	49.24	49.24

Table 1

Some classifiers did not classify classes 1 to 3 but they were not the most important for this task. The most important was to determine the clients that will play in this lottery regularly (4th class). The classifier that had the lowest error for this class was classification tree. The same method also had the lowest total classification error. This means that the most appropriate method to solve this task is to build a classification tree. The use

of this method gives us 86 right answers out of 100 when we want to know whether the client will become a loyal customer.

In this case study there were several inconveniences caused by data set limitations posed by software. Not only was it the record number limitation by Microsoft Excel, but also some unexpected limitations from the XLMiner tool. The work with 60 000 records (XLMiner limit) could not be completed even if the data set had less than 200 attributes that were.

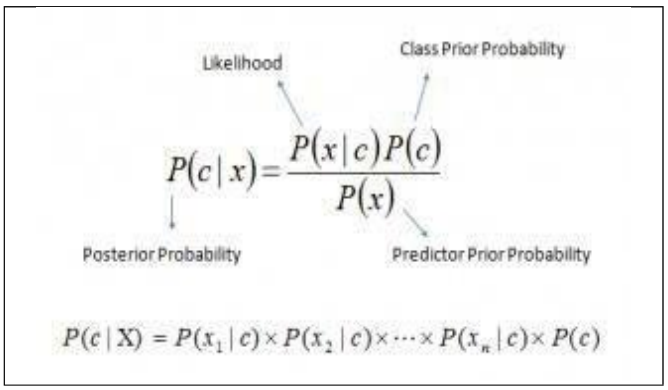
XLMiner solving data mining tasks, concentrating on tasks that are relevant for client relationship management. XLMiner can solve classification, forecasting, clustering, time series analysis and associative rules tasks. And the classification task is the one that is most emphasized. This task can be solved with various methods that work with both categorical (naive Bayes classifier) and continuous data (k- nearest neighbours and others) and the results are comprehensive and more processed. For time series analysis and associative rules there are some of the most popular methods but these tasks are not the main application of this tool, although they can be used successfully as secondary analysis tools. The same is with clustering – some of the methods are offered for this task but result analysis is quite poor and so is visualization. This tool also lacks some data pre-processing utilities. In this case study some tools for attribute set analysis and reduction (subset selection) would have been handy but they are offered only for some regression methods (as built in functions). All of the data set was not used in any task because of the XLMiner limitations. So we have to conclude that this tool is more suitable for tasks with smaller data sets and as an illustration in the learning process for the tasks described previously.

Naive Bayes Algorithm:

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:



- P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
- P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of predictor given class.
- P(x) is the prior probability of predictor

Working Naive Bayes algorithm

Let's understand it using an example. Below I have a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

Step 1: Convert the data set into a frequency table

Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Problem: Players will play if weather is sunny. Is this statement correct?

We can solve it using above discussed method of posterior probability. P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P (Sunny)

Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64 Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

Pros and Cons of Naive Bayes

Pros:

1. It is easy and fast to predict class of test data set. It also performs well in multi class prediction. When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.

2. It performs well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

1. If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.

2. On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.

Applications of Naive Bayes Algorithms

- Real time Prediction: Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.

- Multi class Prediction: This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.

- Text classification/ Spam Filtering/ Sentiment Analysis: Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)

- Recommendation System: Naïve Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not.

LAB EXERCISE:

1) **Java Code**

```
import java.io.*;
import java.util.*;

class element {
        double p[][];
```

```
        element(int n, int m) {
                p = new double[n][m];
        }
}

class Classifier {
        int no_attr = 0, no_rows = 0;
        String fileArray[][];
        String values[][];
        double class_p[];
        int count;
        element a[];

        void readFile(String fname) throws IOException {
                FileInputStream in = null;
                try {
                        File f = new File(fname);
                        in = new FileInputStream(f);
                } catch (Exception e) {
                        System.out.println("ERROR WHILE READING FILE");

                        System.exit(1);
                }
                BufferedReader br = new BufferedReader(new InputStreamReader(in));
                String input = "";
                input = br.readLine();
                StringTokenizer line = new StringTokenizer(input);
                no_attr = line.countTokens() - 1;
                fileArray = new String[100][no_attr + 1];
                for (int i = 0; i <= no_attr; i++)
                        fileArray[no_rows][i] = line.nextToken();
                while (true) {
                        input = br.readLine();
                        if (input == null)
                                break;
                        line = new StringTokenizer(input);
                        no_rows++;
                        for (int i = 0; i <= no_attr; i++)
                                fileArray[no_rows][i] = line.nextToken();
                }
                getAllvalues();
                createTable();
                newEntry();
        }
```

```
boolean in_values(int col_no, String temp) {
        for (int i = 0; i <= count; i++)
                if (values[i][col_no] != null && values[i][col_no].equals(temp))
                        return true;
                e
                l
                s
                e
                        return false
}
continue;
```

```
void getAllvalues() {
                values = new String[100][no_attr + 1];
                count = 0;
                String temp;
        //col major order traverasal for(int i=0;i<=no_attr;i++)
                {
                        for (int j = 1; j <= no_rows; j++) {
                                temp = fileArray[j][i];
                                if (in_values(i, temp))
                                        continue;
                                values[count++][i] = temp;
                        }
                        count = 0;
                }

        }

        int getlen(int col_no) // returns no of rows in particular col_no in array 'values'
        {
                int i = 0;
                while (values[i][col_no] != null)
                        i++;
                return i;
        }

        void display() {
                for (int i = 0; i <= no_rows; i++) {
                        for (int j = 0; j <= no_attr; j++) {
                                System.out.print(values[i][j] + "       ");
                                System.out.println();
                        }
                }
        }

        if(fileArray[i][col_no].equals(temp))tc++;
        return tc;

        void createTable() {
                int tp = getlen(no_attr);
                class_p = new double[tp];

                for (int i = 0; i < tp; i++) {
                        for (int j = 1; j <= no_rows; j++) {
                                if (values[i][no_attr].equals(fileArray[j][no_attr]))
                                        class_p[i]++;
```

```
                        }
                        class_p[i] /= no_rows;
                        System.out.println("P(" + values[i][no_attr] + ")=" + class_p[i]);
                }
                int tc = 0;
                a = new element[no_attr];
                for (int i = 0; i < no_attr; i++) {
                        a[i] = new element(getlen(i), getlen(no_attr));
                        for (int j = 0; j < getlen(i); j++) {
                                for (int k = 0; k < getlen(no_attr); k++) {
                                        for (int x = 1; x <= no_rows; x++) {
                                                if (values[j][i].equals(fileArray[x][i]) &&
                                                values[k][no_attr].equals(fileArray[x][no_attr]))
                                                        tc++;
                                        }
                                        a[i].p[j][k] = (double) tc / getcount(values[k][no_attr], no_attr);
                                        tc = 0;
                                }
                        }
                }
        }

        void newEntry()// takes sample input from user and processes it
        {
                Scanner s = new Scanner(System.in);
                System.out.println("\nEnter New Entry");
                String entry[] = new String[no_attr];
                double p_entry[] = new double[getlen(no_attr)];
                String X = "X=<";
                for (int i = 0; i < no_attr; i++) {
                        System.out.println("enter " + fileArray[0][i]);
                        entry[i] = s.next();
                        X += entry[i] + " ";
                }
                X += ">";
                System.out.println("\nThe Unseen Sample is" + X + "\n");
//Process
                double large = 0.0;
                int pos = -1;
                for (int i = 0; i < getlen(no_attr); i++) {
```

```
                        double product = 1.0;
                        for (int j = 0; j < no_attr; j++) {
                                product *= a[j].p[getindex(j, entry[j])][i];
                        }
                        p_entry[i] = class_p[i] * product;

                        System.out.println("P(X|" + values[i][no_attr] + ").P(" + values[i][no_attr] +
                        ")=" + p_entry[i]);
                        if (p_entry[i] > large) {
                                large = p_entry[i];
                                pos = i;
                        }
                }
                System.out.println("\nThe Decision is " + values[pos][no_attr]);
        }

        int getindex(int col_no, String temp) {
                for (int i = 0; i < getlen(col_no); i++) {
                        if (values[i][col_no].equals(temp))
                                return i;
                }
                System.out.println("Invalid Entry");
                return 32676;
        }
}

class Bayes {
        public static void main(String str[]) throws IOException {
                Scanner s = new Scanner(System.in);
                Classifier c = new Classifier();
                System.out.println("Enter the Name of the input file with its extension");
                c.readFile(s.next());
        }
}
```

OUTPUT:

```
Enter the Name of the input file with its extension
Bayesinput1.txt
P(SUNBURNED)=0.375
P(NONE)=0.625

Enter New Entry
enter HAIR
BROWN
enter HEIGHT
TALL
enter WEIGHT
AVERAGE
enter DUBLIN
NO

The Unseen Sample isX=<BROWN TALL AVERAGE NO >

P(X|SUNBURNED).P(SUNBURNED)=0.0
P(X|NONE).P(NONE)=0.03200000000000001

The Decision is NONE
```

2) WEKA Code

Weather DataSet

@relation weather.symbolic

@attribute outlook {sunny, overcast, rainy}
@attribute temperature {hot, mild, cool}
@attribute humidity {high, normal}
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data sunny,hot,high,FALSE,no
sunny,hot,high,TRUE,no
overcast,hot,high,FALSE,yes
rainy,mild,high,FALSE,yes
rainy,cool,normal,FALSE,yes
rainy,cool,normal,TRUE,no
overcast,cool,normal,TRUE,yes
sunny,mild,high,FALSE,no
sunny,cool,normal,FALSE,yes
rainy,mild,normal,FALSE,yes
sunny,mild,normal,TRUE,yes
overcast,mild,high,TRUE,yes

overcast,hot,normal,FALSE,yes
rainy,mild,high,TRUE,no

Weather test @relation
weather-test
@attribute outlook {sunny, overcast, rainy}
@attribute temperature {hot, mild, cool}
@attribute humidity {high, normal}
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}
@data
rainy,hot,high,FALSE,no
sunny,cool,high,TRUE,no

OUTPUT:

== Run information ===
Scheme: weka.classifiers.bayes.NaiveBayes
Relation: weather.symbolic
Instances: 14
Attributes: 5
outlook
temperature
humidity
windy
play
Test mode: user supplied test set: size unknown (reading incrementally)

=== Classifier model (full training set) ===
Naive Bayes Classifier

	Class	
Attribute	yes	no
	(0.63)	(0.38)
==========	==========	==========
outlook		
sunny	3.0	4.0
overcast	5.0	1.0
rainy4.0	3.0	
[total]	12.0	8.0
temperature		
hot	3.0	3.0
mild	5.0	3.0
cool	4.0	2.0
[total]	12.0	8.0

humidity		
high	4.0	5.0
normal	7.0	2.0
[total]	11.0	7.0
windy		
TRUE	4.0	4.0
FALSE	7.0	3.0
[total]	11.0	7.0

Test Instance
Time taken to build model: 0 seconds

=== Evaluation on test set ===
Time taken to test model on supplied test set: 0 seconds

=== Summary ===

Correctly Classified Instances	2	100	%
Incorrectly Classified Instances	0	0	%
Kappa statistic	1		
Mean absolute error	0.3461		
Root mean squared error	0.3555		
Relative absolute error	55.3714 %		
Root relative squared error	56.8819 %		
Total Number of Instances		2	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0.000	0.000	0.000	0.000	0.000	0.000	?	?	yes
1.000	0.000	1.000	1.000	1.000	0.000	?	1.000	no
Weighted Avg.		1.000	0.000	1.000	1.000	1.000	0.000	0.000	1.000

=== Confusion Matrix ===

a	b	<-- classified as
0	0	a = yes
0	2	b = no

CONCLUSION: Hence, we have studied Data Mining tools like WEKA & XLMiner and implemented classifier Naive Bayes using Java and WEKA.

EXPERIMENT NO.5

TITLE: Implementation of Decision Tree

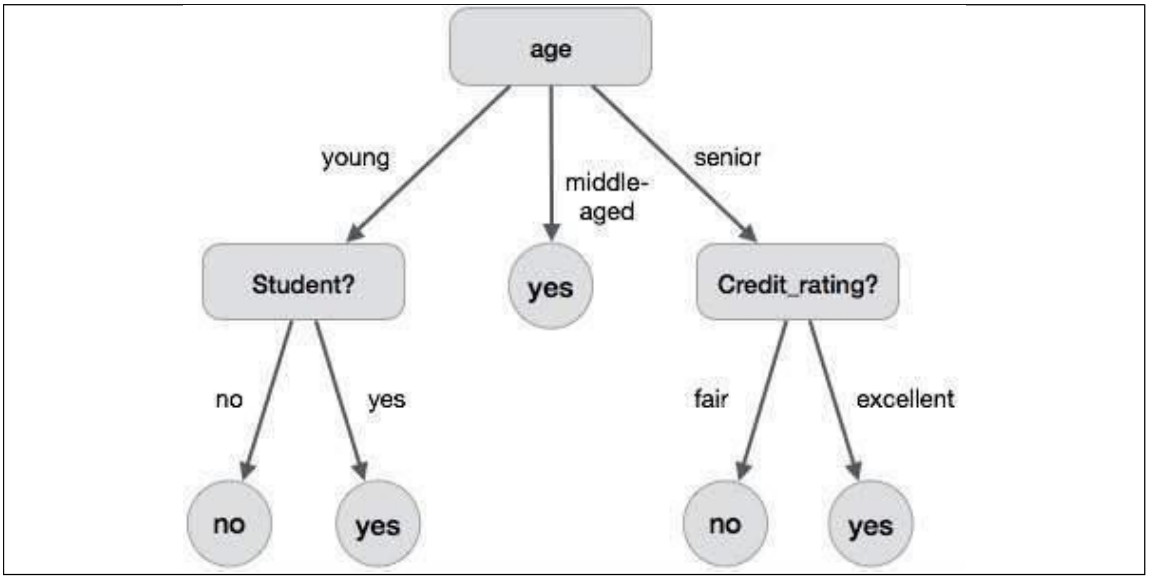
AIM: Write a program to implement Decision Tree using Java and WEKA.

THEORY:

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm.

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node.

The following decision tree is for the concept buy_computer that indicates whether a customer at a company is likely to buy a computer or not. Each internal node represents a test on an attribute. Each leaf node represents a class.



The benefits of having a decision tree are as follows −

- It does not require any domain knowledge.
- It is easy to comprehend.
- The learning and classification steps of a decision tree are simple and fast.

Decision Tree Induction Algorithm

A machine researcher named J. Ross Quinlan in 1980 developed a decision tree algorithm known as ID3 (Iterative Dichotomiser). Later, he presented C4.5, which was the successor of ID3. ID3 and C4.5 adopt a greedy approach. In this algorithm, there is no backtracking; the trees are constructed in a top-down recursive divide-and-conquer manner.

Generating a decision tree form training tuples of data partition D

Algorithm: Generate_decision_tree

Input:

Data partition, D, which is a set of training tuples and their associated class labels. attribute_list, the set of candidate attributes.

Attribute selection method, a procedure to determine the splitting criterion that best partitions that the data tuples into individual classes. This criterion includes a splitting_attribute and either a splitting point or splitting subset.

Output:

A Decision Tree

Method

create a node N;

if tuples in D are all of the same class, C then

return N as leaf node labeled with class C;

if attribute_list is empty then

return N as leaf node with labeled

with majority class in D;|| majority voting

apply attribute_selection_method(D, attribute_list) to find the best splitting_criterion;

label node N with splitting_criterion;

if splitting_attribute is discrete-valued and multiway splits allowed then

// no restricted to binary trees

attribute_list = splitting attribute; // remove splitting attribute

for each outcome j of splitting criterion

// partition the tuples and grow subtrees for each partition

let Dj be the set of data tuples in D satisfying outcome j; // a partition

if Dj is empty then

attach a leaf labeled with the majority class in D to node N;

else

attach the node returned by Generate decision tree(Dj, attribute list) to node N;

end for

return N;

LAB EXERCISE:

1) **Java Code**

```
package ID3;

import java.io.*;
import java.util.*;

public class ID3 {
    int numAttributes;
    String[] attributeNames;
    Vector[] domains;

    /* The class to represent a data point consisting of num Attributes values of attributes */
 class DataPoint {
        public int[] attributes;
```

```
        public DataPoint(int numattributes) {
                attributes = new int[numattributes];
        }
};
/* The class to represent a node in the decomposition tree. */
class TreeNode {
        public double entropy;
        public Vector data;
        public int decompositionAttribute;
        public int decompositionValue;
        public TreeNode[]  children;
        public TreeNode parent;

        public TreeNode() {
                data = new Vector();
        }
};
TreeNode root = new TreeNode();
public int getSymbolValue(int attribute, String symbol) {
        int index = domains[attribute].indexOf(symbol);
        if (index < 0) {
                domains[attribute].addElement(symbol);
                return domains[attribute].size() - 1;
        }
        return index;
}

public int[] getAllValues(Vector data, int attribute) {
        Vector values = new Vector();
```

```
        int num = data.size();
        for (int i = 0; i < num; i++) {
                DataPoint point = (DataPoint) data.elementAt(i);
                String symbol = (String) domains[attribute].elementAt(point.attributes[attribute]);
                int index = values.indexOf(symbol);
                if (index < 0) {
                        values.addElement(symbol);
                }
        }
        int[] array = new int[values.size()];
        for (int i = 0; i < array.length; i++) {
                String symbol = (String) values.elementAt(i);
                array[i] = domains[attribute].indexOf(symbol);
        }
        values = null;
        return array;
}
public Vector getSubset(Vector data, int attribute, int value) {
        Vector subset = new Vector();
        int num = data.size();
        for (int i = 0; i < num; i++) {
                DataPoint point = (DataPoint) data.elementAt(i);
                if (point.attributes[attribute] == value)
                        subset.addElement(point);
        }
        return subset;
}
public double calculateEntropy(Vector data) {
        int numdata = data.size();
```

```
        if (numdata == 0)
                return 0;
        int attribute = numAttributes - 1;
        int numvalues = domains[attribute].size();
        double sum = 0;
        for (int i = 0; i < numvalues; i++) {
                int count = 0;
                for (int j = 0; j < numdata; j++) {
                        DataPoint point = (DataPoint) data.elementAt(j);
                        if (point.attributes[attribute] == i)
                                count++;
                }
                double probability = 1 * count / numdata;
                if (count > 0)
                        sum += -probability * Math.log(probability);
        }
        return sum;
}
public boolean alreadyUsedToDecompose(TreeNode node, int attribute) {
        if (node.children != null) {
                if (node.decompositionAttribute == attribute)
                        return true;
        }
        if (node.parent == null)
                return false;
        return alreadyUsedToDecompose(node.parent, attribute);

}
```

```
public void decomposeNode(TreeNode node) {
double bestEntropy=0;
boolean selected=false; int selectedAttribute=0;
int numdata = node.data.size();
int numinputattributes = numAttributes-1;
node.entropy = calculateEntropy(node.data);
if (node.entropy == 0)
return;
for (int i=0; i< numinputattributes; i++) {
int numvalues = domains[i].size();
if ( alreadyUsedToDecompose(node, i) )
continue;
double averageentropy = 0;
for (int j=0; j< numvalues; j++) {
Vector subset = getSubset(node.data, i, j); if (subset.size() == 0) continue;
double subentropy = calculateEntropy(subset); averageentropy += subentropy *
subset.size();
}
averageentropy = averageentropy / numdata;
Taking the weighted average if (selected == false) {
selected = true;
bestEntropy = averageentropy; selectedAttribute = i;
}
else
{
if (averageentropy < bestEntropy) {
selected = true;
bestEntropy = averageentropy; selectedAttribute = i;
}
```

```
}
}
if(selected==false)return;
        intnumvalues = domains[selectedAttribute].size();node.decompositionAttribute=selectedAttribute;
node.children=new TreeNode[numvalues];
for(int j = 0;j<numvalues;j++)
        {
                node.children[j] = new TreeNode();
                node.children[j].parent = node;
                node.children[j].data = getSubset(node.data, selectedAttribute, j);
                node.children[j].decompositionValue = j;
        }
for(int j = 0;j<numvalues;j++)
        {
                decomposeNode(node.children[j]);
        }node.data=null;
}

public int readData(String filename) throws Exception {
FileInputStream in = null;t try
{
File inputFile = new File(filename); in = new FileInputStream(inputFile);
}
catch ( Exception e)
{
System.err.println( "Unable to open data file: " + filename + "\n" + e);

return 0;
}
```

```
BufferedReader bin = new BufferedReader(new InputStreamReader(in) ); String input;
while(true) {
input = bin.readLine(); if (input == null) {
System.err.println( "No data found in the data file: " + filename +"\n"); return 0;
}
if (input.startsWith("//"))
continue;
if (input.equals(""))
continue;
break;
}
StringTokenizer tokenizer = new StringTokenizer(input);
numAttributes = tokenizer.countTokens();
if (numAttributes <= 1) {
System.err.println( "Read line: " + input);
System.err.println( "Could not obtain the names of attributes in the line");
System.err.println( "Expecting at least one input attribute and one output attribute");
return 0;
}
domains = new Vector[numAttributes];
for (int i=0; i < numAttributes; i++)
domains[i] = new Vector();
attributeNames = new String[numAttributes];
for (int i=0; i < numAttributes; i++) {
attributeNames[i] = tokenizer.nextToken();
}
while(true) {
 input = bin.readLine();
if (input == null) break;
```

```
if (input.startsWith("//"))
continue;
if (input.equals(""))
continue;
tokenizer = new StringTokenizer(input);
int numtokens = tokenizer.countTokens();
if (numtokens != numAttributes) {
System.err.println( "Read " + root.data.size() + " data");
System.err.println( "Last line read: " + input);
System.err.println( "Expecting " + numAttributes + " attributes");
return 0;
}
DataPoint point = new DataPoint(numAttributes);
for (int i=0; i < numAttributes; i++) {
point.attributes[i] = getSymbolValue(i, tokenizer.nextToken());
}
root.data.addElement(point);
}
bin.close(); return 1;
}
        public void printTree(TreeNode node, String tab) {
                int outputattr = numAttributes - 1;
                if (node.children == null) {
                        int[] values = getAllValues(node.data, outputattr);
                        if (values.length == 1) {
                                System.out.println(tab + "\t" + attributeNames[outputattr] + " = \""
                                                + domains[outputattr].elementAt(values[0]) + "\";");
                                return;
                        }
```

```
                System.out.print(tab + "\t" + attributeNames[outputattr] + " = {");
                for (int i = 0; i < values.length; i++) {
                        System.out.print("\"" + domains[outputattr].elementAt(values[i]) + "\" ");
                        if (i != values.length - 1)
                                System.out.print(" , ");
                }
                System.out.println(" };");
                return;
        }
        int numvalues = node.children.length;
        for (int i = 0; i < numvalues; i++) {
                System.out.println(tab + "if( " + attributeNames[node.decompositionAttribute] + " ==
                \""+ domains[node.decompositionAttribute].elementAt(i) + "\") {");
                printTree(node.children[0], tab + "\t");
                if (i != numvalues - 1)
                        System.out.print(tab + "} else ");
                else
                        System.out.println(tab + "}");
        }
}

public void createDecisionTree() {
decomposeNode(root); printTree(root, "");
/* main function */
        public static void main(String[] args) throws Exception {
                ID3 me = new ID3();
                int status = me.readData("D:\\in.txt");
                if (status <= 0)
                        return;
                me.createDecisionTree();
```

```
        }
    }
```

OUTPUT:

```
run:
if( Outlook == "sunny") {
        if( Humidity == "high") {
                        Playball = "no";
        } else  if( Humidity == "normal") {
                        Playball = "no";
        }
} else if( Outlook == "overcast") {
        if( Humidity == "high") {
                        Playball = "no";
        } else  if( Humidity == "normal") {
                        Playball = "no";
        }
} else if( Outlook == "rainy") {
        if( Humidity == "high") {
                        Playball = "no";
        } else  if( Humidity == "normal") {
                        Playball = "no";
        }
}
BUILD SUCCESSFUL (total time: 6 seconds)
```

2) WEKA Code

=== Run information ===

Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2

Relation: weather.symbolic

Instances: 14

Attributes: 5

outlook

temperature

humidity

windy

play

Test mode:evaluate on training data

=== Classifier model (full training set) ===

J48 pruned tree

outlook = sunny

| humidity = high: no (3.0)

| humidity = normal: yes (2.0) outlook = overcast: yes (4.0) outlook = rainy

| windy = TRUE: no (2.0)

| windy = FALSE: yes (3.0)

Number of Leaves : 5

Size of the tree : 8

Time taken to build model: 0.02 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances	14	100	%
Incorrectly Classified Instances	0	0	%
Kappa statistic	1		
Mean absolute error	0		
Root mean squared error	0		
Relative absolute error	0	%	
Root relative squared error	0	%	
Total Number of Instances	14		

=== Detailed Accuracy By Class ===

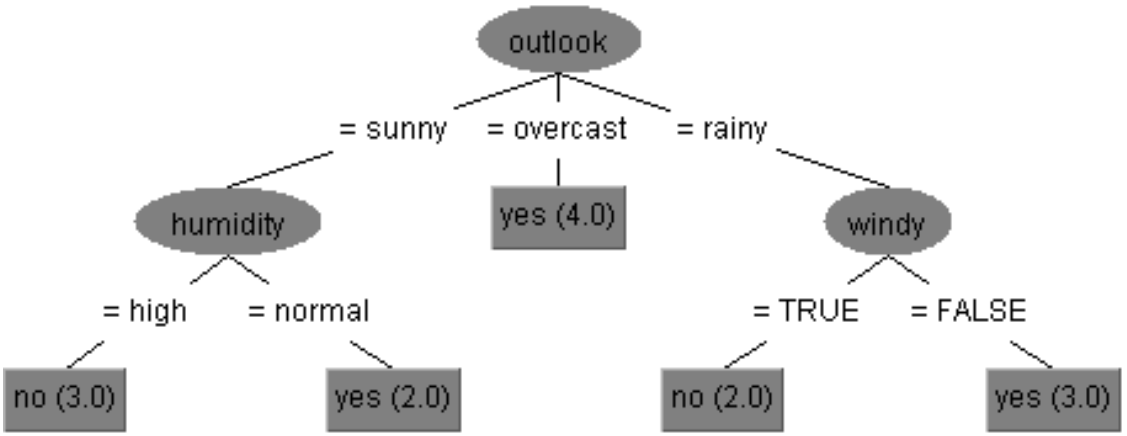
TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	yes
1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	no
Weighted Avg. 1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	

=== Confusion Matrix ===

a b <-- classified as

9 0 | a = yes

0 5 | b = no



CONCLUSION: Hence, we have implemented Decision Tree using Java and WEKA.

EXPERIMENT NO. 6

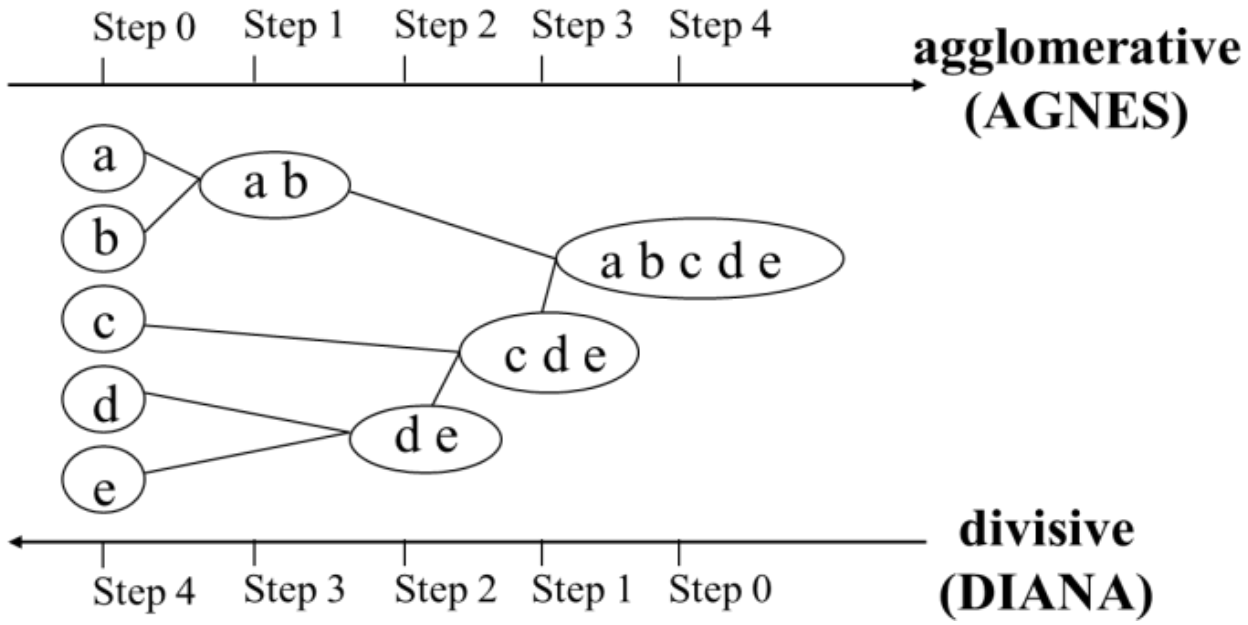
TITLE: Implementation of Agglomerative Hierarchical Clustering Algorithm.

AIM: Write a program to implement Agglomerative Hierarchical Clustering Algorithm using Java and WEKA tool.

THEORY:

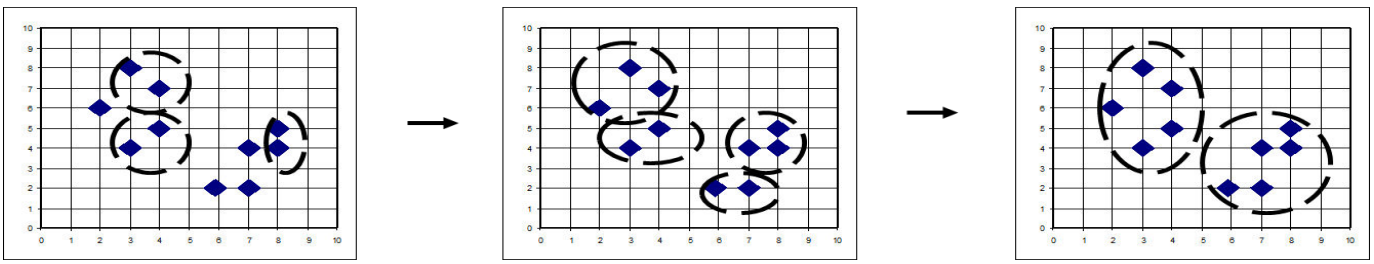
Hierarchical Clustering:

Use distance matrix as clustering criteria. This method does not require the number of clusters k as an input, but needs a termination condition.



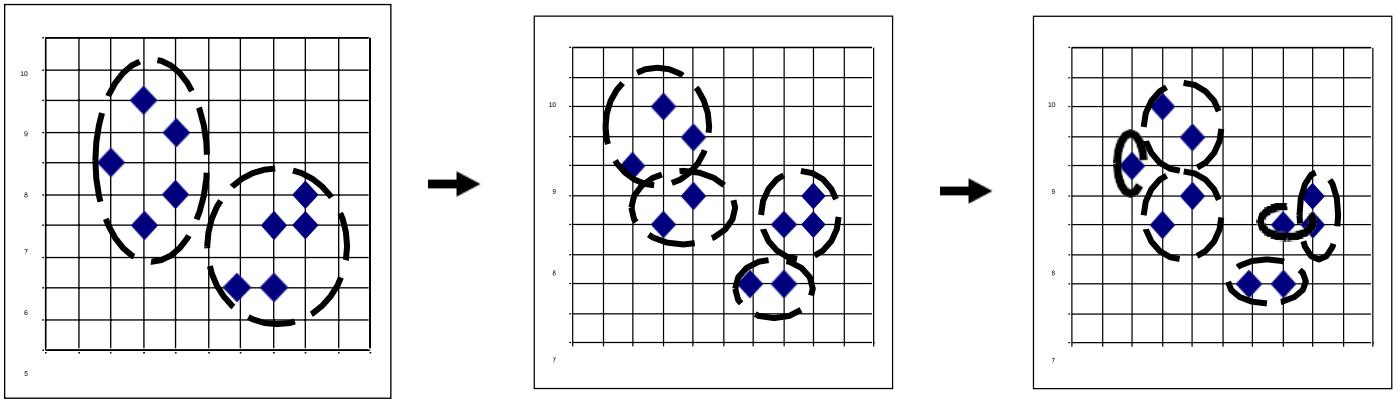
AGNES (Agglomerative Nesting):

Introduced in Kaufmann and Rousseeuw (1990). Implemented in statistical analysis packages, e.g., Splus. Use the Single-Link method and the dissimilarity matrix. Merge nodes that have the least dissimilarity. Go on in a non-descending fashion. Eventually all nodes belong to the same cluster.



DIANA (Divisive Analysis):

Introduced in Kaufmann and Rousseeuw (1990).Implemented in statistical analysis packages, e.g., Splus. Inverse order of AGNES. Eventually each node forms a cluster on its own.



Dendrogram:

Shows How the Clusters are Merged. Decompose data objects into a several levels of nested partitioning (tree of clusters), called a dendrogram. A clustering of the data objects is obtained by cutting the dendrogram at the desired level, then each connected component forms a cluster.

Major weakness of agglomerative clustering methods do not scale well: time complexity of at least O(n2), where n is the number of total objects can never undo what was done previously.

LAB EXERCISE:

1) **Java Code**

```
import java.util.*;

public class Agglo_clustering {
public static void main(string[]args)
{
        Scanner kbd=new Scanner(System.in);
        System.out.println("enter N -the no. of points");
        int N =kbd.nextlnt();
        string dummy =kbd.nextLine(); int M=2;
        double[][] data =new double[N][M]; String names[]=new String[N];
        int count=N;
for(int i=0;i<N;i++)
{
        names[i]="p"+(i+1); double[] d=new double[M];
        System.out.println("Enter the coordinates");
```

```
        for(int j =0; j<M; j++)
{
        d[j]=kbd.nextDouble();
        dummy=kbd.nextLine();
}
data[i]=d;
}
System.out.println("Data entered:");
System.out.print("Point \t X-coord Y-coord\n");
for(int i=0;i<N;i++)
{
System.out.print(names[i]+"\t");
for(int j=0;j<M;j++)
{
System.out.print(data[i][j]+"\t");
}
System.out.println();
}
System.out.println();
int choice=0;
System.out.println("Enter your choice 1.single linkage 2.complete linkage 3.average linkage");
choice=kbd.nextInt();
double INFINITY =Double.POSITIVE_INFINITY;
double[][]d =new double[N][N];
int[] dmin=new int[N];
 for(int i=0; i<N; i++)
{
for(int j=0;j<N;j++)
 {
if(i==j) d[i][j]=INFINITY;
else d[i][j]=distanceTo(data[i],data)[j]);
if(d[i][j]<d[i]dmin[i]]) dmin[i]=j;
}
}
System.out.println("Initial Distance Matrix:");
for(int j=0;j<=i;j++)
{
System.out.print(names[i]+"|");
for(int j=0;j<=i;j++)
{
System.out.print(ound2(d[i][j])+"");
if(d[i][j]!=INFINITY)
System.out.print("\t");
```

```
}
System.out.println();
}
System.out.println("--|");
System.out.println("\t");
for(int i=0;i<N;i++)
{
System.out.print(names[i]+" \t");
}
System.out.println();
 for(int s=0;s<N-1;s++)
{
for(int i=0;i<N;i++)
{
for(int j=0;j<N;j++)
{
if(i==j)d[i][j]=INFINITY;
if(d[i][j]<d[i][dmin[i]])dmin[i]=j;
}
}
int i1=0;
for(int i=0;i<N;i++)
if(d[i][dmin[i]]<d[i][dmin[i1]])
i1=1;
int i2=dmin[i1];
System.out.println("pts"+(i1+1)+"and"+(i2+1)+"are clustered hence new point is("+(i1+1)+","+(i2+1)+")");
if (choice==1)
{
for(int j=0;j<N;j++)
if(d[i2][j]<d[i1][j])
d[i1][j]=d[j][i1]=d[i2][j]; d[i1][i1]=INFINITY;
}
if(choice==2)

{
for(int j=0; j<N ;j++)
if(d[i2][j]>d[i1][j]  &&  d[i2][j]!=INFINITY || d[i1][j]==INFINITY && d[i2][j]!=INFINITY)
d[i1][j]=d[j][i1]=d[i2][j]; d[i1][i1]=INFINITY;
}
if(choice==3)
{
for(int j=0;j<N;j++)
```

```
d[i1][j]= d[j][i1]=(d[i1][j]+d[i2][j])/2;
d[i1][i1]=INFINITY;
}
for(int i=0;i<N;i++)
d[i2][i]=d[i][i2]=INFINITY;
for(int j=0;j<N;j++)
{
if(dmin[j]==i2); dmin[j]=i1;
if(d[i1][j]<d[i1][dmin[i1]])
dmin[i1]=j;
}
System.out.println("Distance matrix after iteration "+s+":");
for(int i=0;i<N;i++)
{
System.out.println(names[i]+"|");
for(int j=0;j<N;j++)
{
System.out.print(round2(d[i][j])+" ");
if(d[i][j]!=INFINITY)
System.out.print("\t");
}
System.out.println();
}
System.out.println(".....|       ");
System.out.println("\t");
for(int i==0;i<N;i++)
{
System.out.println(names[i]+" \t");
}
System.out.println("\n");
}
}
```

```
        static double distanceTo(double start[], double end[]) {
                double temp = Math.pow((end[0] - start[0]), 2) + Math.pow((end[1] - start[1]), 2);
                return Math.sqrt(temp);
        }

        static double round2(double a) {

                if (!(a == Double.POSITIVE_INFINITY)) {
                        int temp = (int) (a * 100);
                        double dbl = ((double) temp) / 100;
```

```
                return dbl;
            } else
                return Double.POSITIVE_INFINITY;

        }

        private static class strings {

            public strings() {
            }
        }

    }
```

OUTPUT:

Enter the number of elements 8

Enter 8 elements: 2 3 6 8 12 15 18 22

Enter the number of clusters: 3

At this step Value of clusters

K1{ 2 }

K2{ 3 }

K3{ 6 8 12 15 18 22 }

Value of m

m1=2.0 m2=3.0 m3=13.5

At this step Value of clusters

K1{ 2 }

K2{ 3 6 8 }

K3{ 12 15 18 22 }

Value of m

m1=2.0 m2=5.666666666666667 m3=16.75

At this step Value of clusters

K1{ 2 3 }

K2{ 6 8 }

K3{ 12 15 18 22 }

Value of m

m1=2.5 m2=7.0 m3=16.75

At this step Value of clusters

K1{ 2 3 }

K2{ 6 8 }

K3{ 12 15 18 22 }

Value of m

m1=2.5 m2=7.0 m3=16.75

The Final Clusters By Agglomerative are as follows:

K1{ 2 3 }

K2{ 6 8 }

K3{ 12 15 18 22 }

2) WEKA Code

=== Run information ===

Scheme: weka.clusterers.HierarchicalClusterer -N 2 -L SINGLE -P -A "weka.core.EuclideanDistance-R first-last"

Relation:iris

Instances: 150

Attributes: 5

sepallength

sepalwidth

petallength

petalwidth class

Test mode: evaluate on training data

=== Clustering model (full training set) ===

Cluster 0

((((((((((((((((((((0.0:0.03254,0.0:0.03254):0.00913,(0.0:0.03254,0.0:0.03254):0.00913):0.00332,((0.0:0.02778,0.0:0.02778):0.00476,0.0:0.03254):0.01244):0,0.0:0.04498):0.0051,0.0:0.05008):0.00364,0.0:0.05371):0.00437,(0.0:0.05085,0.0:0.05085):0.00724):0.01535,(0.0:0.06731,0.0:0.06731):0.00612):0.00188,0.0:0.07531):0.00196,0.0:0.07728):0.00536,((((((0.0:0.04383,0.0:0.04383):0.00625,0.0:0.05008):0,0.0:0.05008):0.00279,(((((0.0:0.03254,0.0:0.03254):0.01129,0.0:0.04383):0.00116,0.0:0.04498):0.0051,0.0:0.05008):0.00279,((0.0:0,0.0:0):0,0.0:0):0.05287):0):0.00522,0.0:0.05808):0.01919,((0.0:0.04498,0.0:0.04498):0.01549,0.0:0.06047):0.0168):0.00536):0.00165,0.0:0.08429):0.00356,(((0.0:0.02778,0.0:0.02778):0.04371,((0.0:0.04498,0.0:0.04498):0.01394,0.0:0.05893):0.01256):0.00809,0.0:0.07958):0.00826):0.00212,0.0:0.08996):0.00321,0.0:0.09317):0.00598,(0.0:0.0678,0.0:0.0678):0.03135):0.00292,0.0:0.10206):0.01316,0.0:0.11523):0.01375,(0.0:0.12263,(0.0:0.10346,0.0:0.10346):0.01917):0.00634):0.00241,0.0:0.13139):0.12414,0.0:0.25553)

Cluster 1

((((((((((((((((1.0:0.07344,(((1.0:0.06508,1.0:0.06508):0.00066,(1.0:0.05008,1.0:0.05008):0.01566):0.00224,1.0:0.06798):0.00546):0.00188,(1.0:0.07137,(1.0:0.05556,1.0:0.05556):0.01581):0.00395):0.00733,(1.0:0.07137,((1.0:0.04498,1.0:0.04498):0.01549,1.0:0.06047):0.01089):0.01127):0.00515,1.0:0.08779):0.00538,1.0:0.09317):0.00405,1.0:0.09722):0.0004,(1.0:0.05556,1.0:0.05556):0.04207):0.00152,(1.0:0.07344,1.0:0.07344):0.02571):0,1.0:0.09914):0.00432,(((((1.0:0.08333,1.0:0.08333):0.00613,((((1.0:0.06574,((1.0:0.05287,1.0:0.05287):0,(1.0:0.05287,(1.0:0.04498,1.0:0.04498):0.00789):0):0.01287):0.0077,(1.0:0.04498,1.0:0.04498):0.02845):0,1.0:0.07344):0.0093,(1.0:0.05287,(1.0:0.04498,1.0:0.04498):0.00789):0.02987):0.00672):0.0005,1.0:0.08996):0.00406,1.0:0.09402):0.00041,1.0:0.09443):0.00902):0.00268,1.0:0.10614):0.02,1.0:0.12614):0.00518,1.0:0.13132):0.0066,(1.0:0.12951,1.

0:0.12951):0.00841):0.00697,(1.0:0.09869,1.0:0.09869):0.0462):0.01518,(((1.0:0.05008,1.0:0.05008):0.04555,1.0:0.09562):0.03389,1.0:0.12951):0.03056):0.00969,1.0:0.16976):0.83409,((((((((((((2.0:0.08983,(2.0:0.06047,2.0:0.06047):0.02935):0.01175,2.0:0.10158):0.01245,(2.0:0.10743,((((2.0:0.071

48,(2.0:0.05008,2.0:0.05008):0.02141):0.02614,(2.0:0.08504,2.0:0.08504):0.01258):0.00852,(((2.0:0.05287,2.0:0.05287):0.04475,((((2.0:0.04383,2.0:0.04383):0.03881,2.0:0.08264):0.00719,(2.0:0.07148,2.0:0.07148):0.01834):0.00487,2.0:0.0947):0.00292):0.00534,2.0:0.10296):0.00318):0,((2.0:0.09415,(2.0:0.04167,2.0:0.04167):0.05249):0.01199,(((2.0:0.08429,(2.0:0.05287,2.0:0.05287):0.03142):0.00518,((2.0:0.03254,2.0:0.03254):0.0254,2.0:0.05794):0.03152):0.00524,2.0:0.0947):0.01144):0):0.00129):0.0066):0.02063,(((2.0:0,2.0:0):0.08779,2.0:0.08779):0.01089,2.0:0.09869):0.03597):0.00139,2.0:0.13605):0.0016,2.0:0.13765):0.01061,(((2.0:0.09869,2.0:0.09869):0.02337,2.0:0.12206):0.01586,((2.0:0.07344,2.0:0.07344):0.05554,(2.0:0.12263,2.0:0.12263):0.00634):0.00895):0.01034):0.00275,2.0:0.15102):0.00299,2.0:0.15401):0.02491,2.0:0.17892):0.01985,2.0:0.19878):0.00279,2.0:0.20156):0.02691,(2.0:0.11232,2.0:0.11232):0.11615):0.0402,2.0:0.26868):0.73517)

Time taken to build model (full training data) : 0.08 seconds

=== Model and evaluation on training set ===

Clustered Instances

0 50 (33%)

1 100 (67%)

CONCLUSION: Hence, we have implemented Agglomerative Clustering Algorithm using WEKA and Java.

EXPERIMENT NO. 7

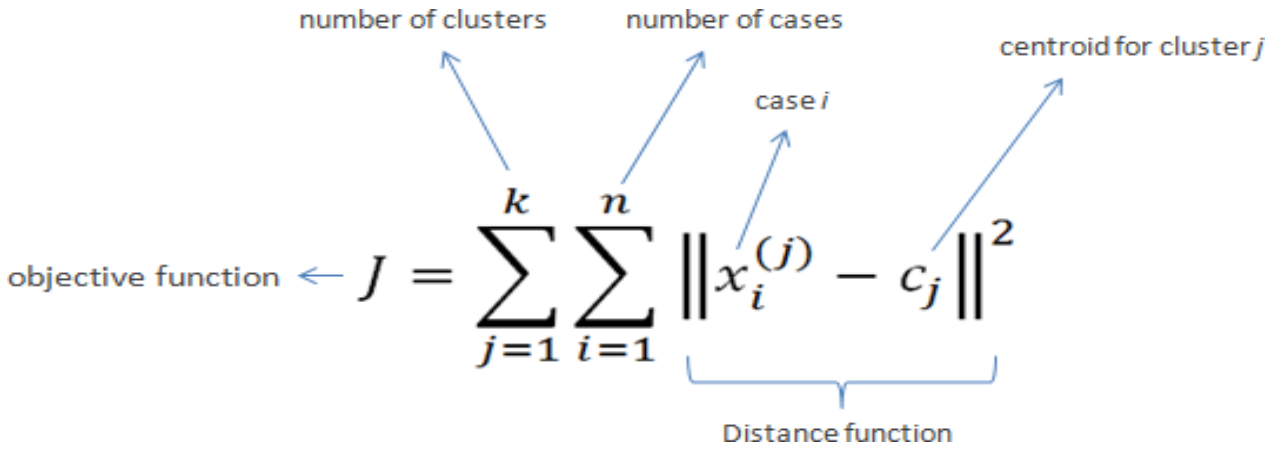
TITLE: Implementation of Clustering algorithm.

AIM: Write a program **t**o implement K-Mean clustering algorithm using WEKA and Java.

THEORY:

K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

K-Means clustering intends to partition n objects into k clusters in which each object belongs to the cluster with the nearest mean. This method produces exactly k different clusters of greatest possible distinction. The best number of clusters k leading to the greatest separation (distance) is not known as a priori and must be computed from the data. The objective of K- Means clustering is to minimize total intra-cluster variance, or, the squared error function:

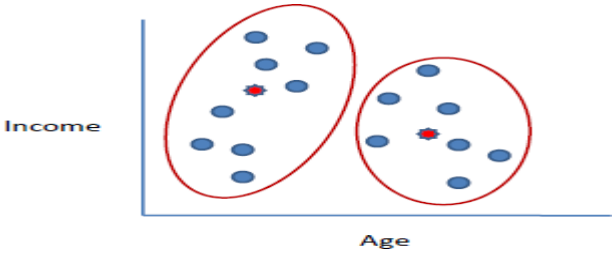


Algorithm

Clusters the data into k groups where k is predefined. Select k points at random as cluster centers.

Assign objects to their closest cluster center according to the Euclidean distance function. Calculate the centroid or mean of all objects in each cluster.

Repeat steps 2, 3 and 4 until the same points are assigned to each cluster in consecutive rounds.



K-Means is relatively an efficient method. However, we need to specify the number of clusters, in advance and the final results are sensitive to initialization and often terminates at a local optimum. Unfortunately, there is no global theoretical method to find the optimal number of clusters. A practical approach is to compare the outcomes of multiple runs with different k and choose the best one based on a predefined criterion. In general, a large k probably decreases the error but increases the risk of overfitting.

Example:

Suppose we want to group the visitors to a website using just their age (a one-dimensional space) as follows:

15,15,16,19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65

Initial clusters:

Centroid (C1) = 16 [16]

Centroid (C2) = 22 [22]

Iteration 1:

C1 = 15.33 [15,15,16]

C2 = 36.25 [19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65]

Iteration 2:

C1 = 18.56 [15,15,16,19,19,20,20,21,22]

C2 = 45.90 [28,35,40,41,42,43,44,60,61,65]

Iteration 3:

C1 = 19.50 [15,15,16,19,19,20,20,21,22,28]

C2 = 47.89 [35,40,41,42,43,44,60,61,65]

Iteration 4:

C1 = 19.50 [15,15,16,19,19,20,20,21,22,28]

C2 = 47.89 [35,40,41,42,43,44,60,61,65]

No change between iterations 3 and 4 has been noted. By using clustering, 2 groups have been identified 15-28 and 35-65. The initial choice of centroids can affect the output clusters, so the algorithm is often run multiple times with different starting conditions in order to get a fair view of what the clusters should be.

LAB EXERCISE:

1) **Java Code**

```
import java.util.*;

class k_means {
        static int count1, count2, count3;
        static int d[];
        static int k[][];
        static int tempk[][];
        static double m[];
        static double diff[];
        static int n, p;

        static int cal_diff(int a) // This method will determine the cluster in which an element go at a particular step
        {
                int temp1 = 0;
                for (int i = 0; i < p; ++i) {
                        if (a > m[i])
                                diff[i] = a - m[i];
                        else
                                diff[i] = m[i] - a;
                }
                int val = 0;
                double temp = diff[0];
                for (int i = 0; i < p; ++i) {
                        if (diff[i] < temp) {
                                temp = diff[i];
                                val = i;
                        }
                } // end of for loop return val;
        }

        static void cal_mean() // This method will determine intermediate mean values
        {
                for (int i = 0; i < p; ++i)
                        m[i] = 0; // initializing means to 0 int cnt=0;
                for (int i = 0; i < p; ++i) {
                        cnt = 0;
                        for (int j = 0; j < n - 1; ++j) {
                                if (k[i][j] != -1) {
                                        m[i] += k[i][j];
                                        ++cnt;
```

```
                    }
                }
                m[i] = m[i] / cnt;
        }
}
```

static int check1() // This checks if previous k ie. tempk and current k are same. Used as terminating case.

```
{
        for (int i = 0; i < p; ++i)
                for (int j = 0; j < n; ++j)
                        if (tempk[i][j] != k[i][j]) {
                                return 0;
                        }
        return 1;
}
```

```
public static void main(String args[]) {
        Scanner scr = new Scanner(System.in);
        /* Accepting number of elements */
System.out.println("Enter the number of elements ");
        n = scr.nextInt();
        d = new int[n];
        /* Accepting elements */
System.out.println("Enter " + n + " elements: ");
        for (int i = 0; i < n; ++i)
                d[i] = scr.nextInt();
        /* Accepting num of clusters */
System.out.println("Enter the number of clusters: ");
        p = scr.nextInt();
        /* Initialising arrays */
        k = new int[p][n];
        tempk = new int[p][n];
        m = new double[p];
        diff = new double[p];
        /* Initializing m */
        for (int i = 0; i < p; ++i)
                m[i] = d[i];
        int temp = 0;
        int flag = 0;
        do {
                for (int i = 0; i < p; ++i)
                        for (int j = 0; j < n; ++j) {
```

```
                        k[i][j] = -1;
                    }
                for (int i = 0; i < n; ++i) // for loop will cal cal_diff(int) for every element.
                {
                    temp = cal_diff(d[i]);
                    if (temp == 0)

                        k[temp][count1++] = d[i];
                    else if (temp == 1)
                        k[temp][count2++] = d[i];
                    else if (temp == 2)
                        k[temp][count3++] = d[i];
                }
                cal_mean();
                // call to method which will calculate mean at this step. flag=check1(); //
                // check if terminating condition is satisfied. if(flag!=1)
                /*Take backup of k in tempk so that you can check for equivalence in next step*/
                for (int i = 0; i < p; ++i)
                    for (int j = 0; j < n; ++j)
                        tempk[i][j] = k[i][j];
                System.out.println("\n\nAt this step");
                System.out.println("\nValue of clusters");
                for (int i = 0; i < p; ++i) {
                    System.out.print("K" + (i + 1) + "{ ");
                    for (int j = 0; k[i][j] != -1 && j < n - 1; ++j)
                        System.out.print(k[i][j] + " ");
                    System.out.println("}");
                } // end of for loop System.out.println("\nValue of m "); for(int i=0;i<p;++i)
                System.out.print("m" + (i + 1) + "=" + m[i] + " ");
                count1 = 0;
                count2 = 0;
                count3 = 0;
            } while (flag == 0);
            System.out.println("\n\n\nThe Final Clusters By Kmeans are as follows: ");
            for (int i = 0; i < p; ++i) {
                System.out.print("K" + (i + 1) + "{ ");
                for (int j = 0; k[i][j] != -1 && j < n - 1; ++j)
                    System.out.print(k[i][j] + " ");
                System.out.println("}");
            }
        }
}
```

OUTPUT:

Enter the number of elements 8

Enter 8 elements:

2 3 6 8 12 15 18 22

Enter the number of clusters:

3

At this step Value of clusters K1{ 2 }

K2{ 3 }

K3{ 6 8 12 15 18 22 }

Value of m

m1=2.0 m2=3.0 m3=13.5

At this step Value of clusters K1{ 2 }

K2{ 3 6 8 }

K3{ 12 15 18 22 }

Value of m

m1=2.0 m2=5.666666666666667 m3=16.75

At this step Value of clusters K1{ 2 3 }

K2{ 6 8 }

K3{ 12 15 18 22 }

Value of m

m1=2.5 m2=7.0 m3=16.75

At this step Value of clusters K1{ 2 3 }

K2{ 6 8 }

K3{ 12 15 18 22 }

Value of m

m1=2.5 m2=7.0 m3=16.75

The Final Clusters By Kmeans are as follows: K1{ 2 3 }

K2{ 6 8 }

K3{ 12 15 18 22 }

2) WEKA Code

=== Run information ===

Scheme: weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -S 10

Relation: iris

Instances: 150

Attributes: 5

sepallength

sepalwidth

petallength

petalwidth

class

Test mode: evaluate on training data

=== Clustering model (full training set) ===

kMeans

======

Number of iterations: 7

Within cluster sum of squared errors: 62.1436882815797

Initial starting points (random):

Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor

Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor

Missing values globally replaced with mean/mode

Final cluster centroids:

	Cluster#		
Attribute	Full Data	0	1
	(150.0)	(100.0)	(50.0)
sepallength	5.8433	6.262	5.006
sepalwidth	3.054	2.872	3.418
petallength	3.7587	4.906	1.464
petalwidth	1.1987	1.676	0.244
class	Iris-setosa	Iris-versicolor	Iris-setosa

Time taken to build model (full training data) : 0.01 seconds

=== Model and evaluation on training set ===

Clustered Instances

0 100 (67%)

1 50 (33%)

CONCLUSION: Hence, we have implemented K-Mean using WEKA and Java.

EXPERIMENT NO.8

TITLE: Implementation of Association mining tool algorithm.

AIM: Write a program **t**o implement Frequent Pattern (FP) Tree using Java and Weka Tool.

THEORY:

The FP-Growth Algorithm is an alternative algorithm used to find frequent item sets. FP stands for frequent pattern. It is vastly different from the Apriori Algorithm explained in previous sections in that it uses a FP-tree to encode the data set and then extract the frequent itemsets from this tree. This section is divided into two main parts, the first deals with the representation of the FP-tree and the second details how frequent itemset generation occurs using this tree and its algorithm.

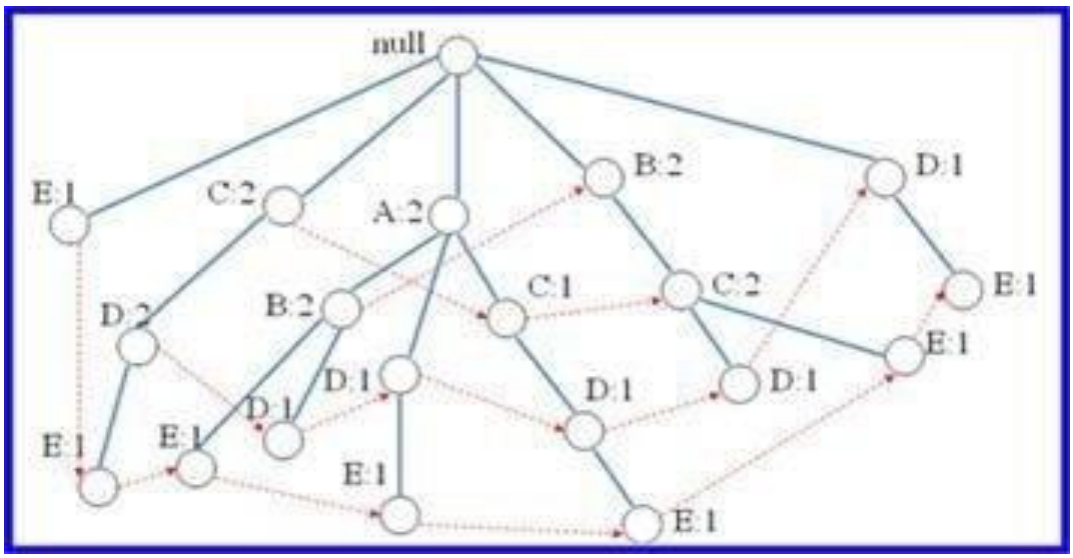
In the first pass, the algorithm counts occurrence of items (attribute-value pairs) in the dataset, and stores them to 'header table'. In the second pass, it builds the FP-tree structure by inserting instances. Items in each instance have to be sorted by descending order of their frequency in the dataset, so that the tree can be processed quickly. Items in each instance that do not meet minimum coverage threshold are discarded. If many instances share most frequent items, FP-tree provides high compression close to tree root.

Recursive processing of this compressed version of main dataset grows large item sets directly, instead of generating candidate items and testing them against the entire database. Growth starts from the bottom of the header table (having longest branches), by finding all instances matching given condition. New tree is created, with counts projected from the original tree corresponding to the set of instances that are conditional on the attribute, with each node getting sum of its children counts. Recursive growth ends when no individual items conditional on the attribute meet minimum support threshold, and processing continues on the remaining header items of the original FP-tree. Once the recursive process has completed, all large item sets with minimum coverage have been found, and association rule creation begins.

FP-Tree Representation

A FP-tree is a compact data structure that represents the data set in tree form. Each transaction is read and then mapped onto a path in the FP-tree. This is done until all transactions have been read. Different transactions that have common subsets allow the tree to remain compact because their paths overlap. The diagram to the right is an example of a best-case scenario that occurs when all transactions have exactly the same itemset; the size of the FP-tree will be only a single branch of nodes.

The worst case scenario occurs when every transaction has a unique itemset and so the space needed to store the tree is greater than the space used to store the original data set because the FP-tree requires additional space to store pointers between nodes and also the counters for each item. The diagram below is an example of how a worst case scenario FP-tree might appear. As you can see, the complexity of the tree grows with the uniqueness of each transaction



Construction:

The construction of a FP-tree is subdivided into three major steps:

1) Scan the data set to determine the support count of each item, discard the infrequent items and sort the frequent items in decreasing order.

2) Scan the data set one transaction at a time to create the FP-tree. For each transaction:
   a. If it is a unique transaction form a new path and set the counter for each node to 1.

   b. If it shares a common prefix itemset then increment the common itemset node counters and create new nodes if needed.

3) Continue this until each transaction has been mapped unto the tree.

LAB EXERCISE:

1) **Java Code**

```
import java.io.*;

class FP_Tree {
        public static void main(String arg[]) throws IOException {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                System.out.print("Enter number of transactions:");
                int no_t = Integer.parseInt(br.readLine());
                System.out.print("enter the no. of items in the Itemeset:");
                int no_i = Integer.parseInt(br.readLine());
                System.out.print("enter the minimum support:");
                double min_sup = Double.parseDouble(br.readLine());
```

```
System.out.println("enter the item set(should be strictly small case alphabets) and Enter 0 once you finish");
String[][][] d = new String[no_t][2][no_i];
for (int i = 0; i < no_t; i++) {
        System.out.println("TID no: " + (i + 1));
        for (int k = 0; k < no_i; k++) {
                String s = br.readLine();
                d[i][1][k] = s;
                if (d[i][1][k].equals("0"))
                        break;
        }
}

char item_set[] = new char[no_i];
System.out.println("Item Set:");
for (int i = 0; i < item_set.length; i++)

{
        item_set[i] = (char) (i + 97);
        System.out.print(" " + item_set[i]);
}
int sup[] = new int[item_set.length];
int s = 0;

System.out.println("\nCorresponding supports");
for (int j = 0; j < sup.length; j++) {
        s = 0;

        for (int i = 0; i < no_t; i++) {
                for (int k = 0; k < no_i; k++) {
                        if (d[i][1][k].equals("0"))
                                break;
                        if ((d[i][1][k].charAt(0)) == (item_set[j]))
                                s++;
                        sup[j] = s;
                }
        }
        System.out.print(" " + sup[j]);
}
char item_set_new[] = new char[item_set.length];
int count = 0;
for (int k = 0; k < sup.length; k++) {
        if (sup[k] >= min_sup) {
                item_set_new[k] = item_set[k];
```

```
                    count++;
                }
            }
            System.out.println();
            for (int i = 0; i < sup.length; i++) {
                for (int j = 0; j < sup.length - 1; j++) {
                    if (sup[j] < sup[j + 1]) {
                        int temp = sup[j];
                        sup[j] = sup[j + 1];
                        sup[j + 1] = temp;
                        char temp1 = item_set_new[j];
                        item_set_new[j] = item_set_new[j + 1];
                        item_set_new[j + 1] = temp1;
                    }
                }
            }
            char is_final[] = new char[count];
            int sup_final[] = new int[count];
            int cnt = 0;
            for (int i = 0; i < no_i; i++) {
                if ((Character.isLetter(item_set_new[i])))
                {
                    is_final[cnt] = item_set_new[i];
                    sup_final[cnt] = sup[i];
                    cnt++;
                }
            }
            for (int i = 0; i < is_final.length; i++) {
                System.out.println(is_final[i] + "\t" + sup_final[i]);
            }
            for (int i = 0; i < no_t; i++) {
                System.out.println("transaction No :" + (i + 1));
                System.out.print("Root");

                for (int m = 0; m < count; m++) {
                    for (int k = 0; k < no_i; k++) {
                        if (d[i][1][k].equals("0"))
                            break;
                        if (((d[i][1][k].charAt(0)) == (is_final[m])) && (sup_final[m] >
                        0)) {
                            System.out.print("\n|\n" + is_final[m]);
                            sup_final[m]--;
                            break;
                        }
```

```
                        }
                    }
                    System.out.println("\n\n");
                }
            }
        }
```

OUTPUT:

Enter number of transactions: 5

Enter the no. of items in the Itemeset:16

Enter the minimum support: 3

Enter the item set(should be strictly small case alphabets) and Enter 0 once you finish

TID no: 1

f a c d g i m p 0t

TID no: 2

a b c f l m o 0

TID no: 3

b f h j o 0

TID no: 4

b c k s p 0

TID no: 5

a f c e l p m n 0

Item Set:

a b c d e f g h i j k l m n o p

Corresponding supports

3 3 4 1 1 4 1 1 1 1 1 2 3 1 2 3

c 4

f 4

a 3

b 3

m 3

p 3

transaction No :1

Root

| c | f | a | m | p

transaction No :2

Root

| c | f | a | b | m

transaction No :3

Root

| f | b

transaction No :4

Root

| c | b | p

transaction No :5

Root

| c | f | a | m | p

2) WEKA Code

Scheme: weka.clusterers.HierarchicalClusterer -N 2 -L SINGLE -P -A"weka.core.EuclideanDistance -R first-last"

Relation: iris

Instances: 150

Attributes: 5

sepallength

sepalwidth

petallength

petalwidth class

Test mode: evaluate on training data

=== Clustering model (full training set) ===

Cluster 0

((((((((((((((((((((0.0:0.03254,0.0:0.03254):0.00913,(0.0:0.03254,0.0:0.03254):0.00913):0.00332,((0.0:0.02778,0.0:0.02778):0.00476,0.0:0.03254):0.01244):0,0.0:0.04498):0.0051,0.0:0.05008):0.00364,0.0:0.05371):0.00437,(0.0:0.05085,0.0:0.05085):0.00724):0.01535,(0.0:0.06731,0.0:0.06731):0.00612):0.00188,0.0:0.07531):0.00196,0.0:0.07728):0.00536,((((((0.0:0.04383,0.0:0.04383):0.00625,0.0:0.05008):0,0.0:0.05008):0.00279,(((((0.0:0.03254,0.0:0.03254):0.01129,0.0:0.04383):0.00116,0.0:0.04498):0.0051,0.0:0.05008):0.00279,((0.0:0,0.0:0):0,0.0:0):0.05287):0):0.00522,0.0:0.05808):0.01919,((0.0:0.04498,0.0:0.04498):0.01549,0.0:0.06047):0.0168):0.00536):0.00165,0.0:0.08429):0.00356,(((0.0:0.02778,0.0:0.02778):0.04371,((0.0:0.04498,0.0:0.04498):0.01394,0.0:0.05893):0.01256):0.00809,0.0:0.07958):0.00826):0.00212,0.0:0.08996):0.00321,0.0:0.09317):0.00598,(0.0:0.0678,0.0:0.0678):0.03135):0.00292,0.0:0.10206):0.01316,0.0:0.11523):0.01375,(0.0:0.12263,(0.0:0.10346,0.0:0.10346):0.01917):0.00634):0.00241,0.0:0.13139):0.12414,0.0:0.25553)

Cluster 1

((((((((((((((((1.0:0.07344,(((1.0:0.06508,1.0:0.06508):0.00066,(1.0:0.05008,1.0:0.05008):0.01566):0.00224,1.0:0.06798):0.00546):0.00188,(1.0:0.07137,(1.0:0.05556,1.0:0.05556):0.01581):0.00395):0.00733,(1.0:0.07137,((1.0:0.04498,1.0:0.04498):0.01549,1.0:0.06047):0.01089):0.01127):0.00515,1.

0:0.08779):0.00538,1.0:0.09317):0.00405,1.0:0.09722):0.0004,(1.0:0.05556,1.0:0.05556):0.04207):0.00152,(1.0:0.07344,1.0:0.07344):0.02571):0,1.0:0.09914):0.00432,(((((1.0:0.08333,1.0:0.08333):0.00613,((((1.0:0.06574,((1.0:0.05287,1.0:0.05287):0,(1.0:0.05287,(1.0:0.04498,1.0:0.04498):0.00789):0):0.01287):0.0077,(1.0:0.04498,1.0:0.04498):0.02845):0,1.0:0.07344):0.0093,(1.0:0.05287,(1.0:0.04498,1.0:0.04498):0.00789):0.02987):0.00672):0.0005,1.0:0.08996):0.00406,1.0:0.09402):0.00041,1.0:0.09443):0.00902):0.00268,1.0:0.10614):0.02,1.0:0.12614):0.00518,1.0:0.13132):0.0066,(1.0:0.12951,1.0:0.12951):0.00841):0.00697,(1.0:0.09869,1.0:0.09869):0.0462):0.01518,(((1.0:0.05008,1.0:0.05008):0.04555,1.0:0.09562):0.03389,1.0:0.12951):0.03056):0.00969,1.0:0.16976):0.83409,((((((((((((2.0:0.08983,(2.0:0.06047,2.0:0.06047):0.02935):0.01175,2.0:0.10158):0.01245,(2.0:0.10743,((((2.0:0.07148,(2.0:0.05008,2.0:0.05008):0.02141):0.02614,(2.0:0.08504,2.0:0.08504):0.01258):0.00852,(((2.0:0.05287,2.0:0.05287):0.04475,((((2.0:0.04383,2.0:0.04383):0.03881,2.0:0.08264):0.00719,(2.0:0.07148,2.0:0.07148):0.01834):0.00487,2.0:0.0947):0.00292):0.00534,2.0:0.10296):0.00318):0,((2.0:0.09415,(2.0:0.04167,2.0:0.04167):0.05249):0.01199,(((2.0:0.08429,(2.0:0.05287,2.0:0.05287):0.03142):0.00518,((2.0:0.03254,2.0:0.03254):0.0254,2.0:0.05794):0.03152):0.00524,2.0:0.0947):0.01144):0):0.00129):0.0066):0.02063,(((2.0:0,2.0:0):0.08779,2.0:0.08779):0.01089,2.0:0.09869):0.03597):0.00139,2.0:0.13605):0.0016,2.0:0.13765):0.01061,(((2.0:0.09869,2.0:0.09869):0.02337,2.0:0.12206):0.01586,((2.0:0.07344,2.0:0.07344):0.05554,(2.0:0.12263,2.0:0.12263):0.00634):0.00895):0.01034):0.00275,2.0:0.15102):0.00299,2.0:0.15401):0.02491,2.0:0.17892):0.01985,2.0:0.19878):0.00279,2.0:0.20156):0.02691,(2.0:0.11232,2.0:0.11232):0.11615):0.0402,2.0:0.26868):0.73517)

Time taken to build model (full training data) : 0.08 seconds

=== Model and evaluation on training set ===

Clustered Instances

0 50 (33%)

1 100 (67%)

CONCLUSION: Hence, we have implemented FP Tree using Java and Weka Tool.

DEPARTMENT OF COMPUTER ENGINEERING

EXPERIMENT NO. 9

AIM: Implementation of association mining apriori using Java and WEKA.

OBJECTIVE: To understand the Implementation of association mining apriori using Java and WEKA.

THEORY:

Apriori Algorithm

The apriori principle can reduce the number of itemsets we need to examine. Put simply, the apriori principle states that if an itemset is infrequent, then all its subsets must also be infrequent. This means that if {beer} was found to be infrequent, we can expect {beer, pizza} to be equally or even more infrequent. So in consolidating the list of popular itemsets, we need not consider {beer, pizza}, nor any other itemset configuration that contains beer.

Finding itemsets with high support

Using the apriori principle, the number of itemsets that have to be examined can be pruned, and the list of popular itemsets can be obtained in these steps:

Step 0. Start with itemsets containing just a single item, such as {apple} and {pear}.

Step 1. Determine the support for itemsets. Keep the itemsets that meet your minimum support threshold, and remove itemsets that do not.

Step 2. Using the itemsets you have kept from Step 1, generate all the possible itemset configurations.

Step 3. Repeat Steps 1 & 2 until there are no newer itemsets.

Example

Assume that a large supermarket tracks sales data by stock-keeping unit (SKU) for each item: each item, such as "butter" or "bread", is identified by a numerical SKU. The supermarket has a database of transactions where each transaction is a set of SKUs that were bought together.

Let the database of transactions consist of following itemsets:

Itemsets
{1,2,3,4}
{1,2,4}
{1,2}
{2,3,4}
{2,3}
{3,4}
{2,4}

DEPARTMENT OF COMPUTER ENGINEERING

We will use Apriori to determine the frequent item sets of this database. To do so, we will say that an item set is frequent if it appears in at least 3 transactions of the database: the value 3 is the support threshold.The first step of Apriori is to count up the number of occurrences, called the support, of each member item separately. By scanning the database for the first time, we obtain the following result

Item	Support
{1}	3
{2}	6
{3}	4
{4}	5

All the itemsets of size 1 have a support of at least 3, so they are all frequent.
The next step is to generate a list of all pairs of the frequent items.
For example, regarding the pair {1,2}: the first table of Example 2 shows items 1 and 2 appearing together in three of the itemsets; therefore, we say item {1,2} has support of three.

Item	Support
{1,2}	3
{1,3}	1
{1,4}	2
{2,3}	3
{2,4}	4
{3,4}	3

The pairs {1,2}, {2,3}, {2,4}, and {3,4} all meet or exceed the minimum support of 3, so they are frequent. The pairs {1,3} and {1,4} are not. Now, because {1,3} and {1,4} are not frequent, any larger set which contains {1,3} or {1,4} cannot be frequent. In this way, we can prune sets: we will now look for frequent triples in the database, but we can already exclude all the triples that contain one of these two pairs:

Item	Support
{2,3,4}	2

In the example, there are no frequent triplets -- {2,3,4} is below the minimal threshold, and the other triplets were excluded because they were super sets of pairs that were already below the threshold.

We have thus determined the frequent sets of items in the database, and illustrated how some items were not counted because one of their subsets was already known to be below the threshold.

Limitations

- Computationally Expensive.:Even though the apriori algorithm reduces the number of candidate itemsets to consider, this number could still be huge when store inventories are large or when the support threshold is low. However, an alternative solution would be to reduce the number of comparisons by using advanced data structures, such as hash tables, to sort candidate itemsets more efficiently.
- Spurious Associations: Analysis of large inventories would involve more itemset configurations, and the support threshold might have to be lowered to detect certain associations. However, lowering the support threshold might also increase the number of spurious associations detected. To ensure that identified associations are generalizable, they could first be distilled from a training dataset, before having their support and confidence assessed in a separate test dataset.

Weka:

=== Run information ===

Scheme: weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation: supermarket
Instances: 4627
Attributes: 217
[list of attributes omitted]
=== Associator model (full training set) ===

Apriori
========
Minimum support: 0.15 (694 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

Size of set of large itemsets L(1): 44
Size of set of large itemsets L(2): 380
Size of set of large itemsets L(3): 910
Size of set of large itemsets L(4): 633
Size of set of large itemsets L(5): 105
Size of set of large itemsets L(6): 1

Best rules found:

1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723 <conf:(0.92)> lift:(1.27) lev:(0.03) [155] conv:(3.35)

2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696 <conf:(0.92)> lift:(1.27) lev:(0.03) [149] conv:(3.28)

3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705 <conf:(0.92)> lift:(1.27) lev:(0.03) [150] conv:(3.27)

4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746 <conf:(0.92)> lift:(1.27) lev:(0.03) [159] conv:(3.26)

5. party snack foods=t fruit=t total=high 854 ==> bread and cake=t 779 <conf:(0.91)> lift:(1.27) lev:(0.04) [164] conv:(3.15)

6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725 <conf:(0.91)> lift:(1.26) lev:(0.03) [151] conv:(3.06)

7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701 <conf:(0.91)> lift:(1.26) lev:(0.03) [145] conv:(3.01)

8. biscuits=t fruit=t total=high 954 ==> bread and cake=t 866 <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(3)

9. frozen foods=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757 <conf:(0.91)> lift:(1.26) lev:(0.03) [156] conv:(3)

10. frozen foods=t fruit=t total=high 969 ==> bread and cake=t 877 <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(2.92)

Java:

```
import java.io.*;
import java.util.*;

public class Apriori {
   public static void main(String[] args) {
      AprioriCalculation ap = new AprioriCalculation();
      ap.aprioriProcess();
   }
}

class AprioriCalculation {
   Vector<String> candidates = new Vector<String>(); // the current candidates
   String configFile = "C:\\Users\\HP\\Desktop\\Config.txt"; // configuration file
   String transaFile = "C:\\Users\\HP\\Desktop\\transa.txt"; // transaction file
   String outputFile = "apriori-output.txt"; // output file
   int numItems; // number of items per transaction
   int numTransactions; // number of transactions
   double minSup; // minimum support for a frequent itemset
   String oneVal[]; // array of values per column that will be treated as a '1'
```

```
String itemSep = " "; // the separator value for items in the database

public void aprioriProcess() {
   Date d; // date object for timing purposes
   long start, end; // start and end time
   int itemsetNumber = 0; // the current itemset being looked at

   // get config
   getConfig();
   System.out.println("Apriori algorithm has started.\n");

   // start timer
   d = new Date();
   start = d.getTime();

   // while not complete
   do {
      // increase the itemset that is being looked at
      itemsetNumber++;
      // generate the candidates
      generateCandidates(itemsetNumber);
      // determine and display frequent itemsets
      calculateFrequentItemsets(itemsetNumber);

      if (candidates.size() != 0) {
         System.out.println("Frequent " + itemsetNumber + "-itemsets");
         System.out.println(candidates);
      }

      // if there are <=1 frequent items, then it's the end
      // This prevents reading through the database again when there is only one frequent itemset.
   } while (candidates.size() > 1);

   // end timer
   d = new Date();
   end = d.getTime();

   // display the execution time
   System.out.println("Execution time is: " + ((double) (end - start) / 1000) + " seconds.");
}

public static String getInput() {
   String input = "";
   // read from System.in
   BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
   // try to get the user's input, if there is an error, print the message
```

```
    try {
        input = reader.readLine();
    } catch (Exception e) {
        System.out.println(e);
    }
    return input;
}

private void getConfig() {
    FileWriter fw;
    BufferedWriter file_out;
    String input = "";

    // ask if want to change the config
    System.out.println("Default Configuration: ");
    System.out.println("\tRegular transaction file with '" + itemSep + "' item separator.");
    System.out.println("\tConfig File: " + configFile);
    System.out.println("\tTransa File: " + transaFile);
    System.out.println("\tOutput File: " + outputFile);
    System.out.println("\nPress 'C' to change the item separator, configuration file and transaction files");
    System.out.print("or any other key to continue. ");
    input = getInput();

    if (input.compareToIgnoreCase("c") == 0) {
        System.out.print("Enter new transaction filename (return for '" + transaFile + "'): ");
        input = getInput();
        if (input.compareToIgnoreCase("") != 0)
            transaFile = input;

        System.out.print("Enter new configuration filename (return for '" + configFile + "'): ");
        input = getInput();
        if (input.compareToIgnoreCase("") != 0)
            configFile = input;

        System.out.print("Enter new output filename (return for '" + outputFile + "'): ");
        input = getInput();
        if (input.compareToIgnoreCase("") != 0)
            outputFile = input;

        System.out.println("Filenames changed");
        System.out.print("Enter the separating character(s) for items (return for '" + itemSep + "'): ");
        input = getInput();
        if (input.compareToIgnoreCase("") != 0)
            itemSep = input;
    }
```

```
    try {
      FileInputStream file_in = new FileInputStream(configFile);
      BufferedReader data_in = new BufferedReader(new InputStreamReader(file_in));
      // number of items
      numItems = Integer.valueOf(data_in.readLine()).intValue();

      // number of transactions
      numTransactions = Integer.valueOf(data_in.readLine()).intValue();
      // minsup
      minSup = (Double.valueOf(data_in.readLine()).doubleValue());
      // output config info to the user
      System.out.print("\nInput configuration: " + numItems + " items, " + numTransactions + " transactions,
");
      System.out.println("minsup = " + minSup + "%");
      System.out.println();
      minSup /= 100.0;
      oneVal = new String[numItems];
      System.out.print("Enter 'y' to change the value each row recognizes as a '1':");
      if (getInput().compareToIgnoreCase("y") == 0) {
        for (int i = 0; i < oneVal.length; i++) {
          System.out.print("Enter value for column #" + (i + 1) + ": ");
          oneVal[i] = getInput();
        }
      } else {
        for (int i = 0; i < oneVal.length; i++)
          oneVal[i] = "1";
      }
      // create the output file
      fw = new FileWriter(outputFile);
      file_out = new BufferedWriter(fw);
      // put the number of transactions into the output file
      file_out.write(numTransactions + "\n");
      file_out.write(numItems + "\n******\n");
      file_out.close();
    } catch (IOException e) {
      System.out.println(e);
    }
  }

  private void generateCandidates(int n) {
    Vector<String> tempCandidates = new Vector<String>(); // temporary candidate string vector
    String str1, str2; // strings that will be used for comparisons
    StringTokenizer st1, st2; // string tokenizers for the two itemsets being compared

    // if it's the first set, candidates are just the numbers
    if (n == 1) {
```

```
        for (int i = 1; i <= numItems; i++) {
            tempCandidates.add(Integer.toString(i));
        }
    } else if (n == 2) { // the second itemset is just all combinations of itemset 1
        // add each itemset from the previous frequent itemsets together
        for (int i = 0; i < candidates.size(); i++) {
            st1 = new StringTokenizer(candidates.get(i));
            str1 = st1.nextToken();
            for (int j = i + 1; j < candidates.size(); j++) {
                st2 = new StringTokenizer(candidates.elementAt(j));
                str2 = st2.nextToken();
                tempCandidates.add(str1 + " " + str2);
            }
        }
    } else {
        // for each itemset
        for (int i = 0; i < candidates.size(); i++) {
            // compare it to the next itemset
            for (int j = i + 1; j < candidates.size(); j++) {
                // create the strings
                str1 = new String();
                str2 = new String();
                // create the tokenizers
                st1 = new StringTokenizer(candidates.get(i));
                st2 = new StringTokenizer(candidates.get(j));
                // make a string of the first n-2 tokens of the strings
                for (int s = 0; s < n - 2; s++) {
                    str1 = str1 + " " + st1.nextToken();
                    str2 = str2 + " " + st2.nextToken();
                }
                // if they have the same n-2 tokens, add them together
                if (str2.compareToIgnoreCase(str1) == 0)
                    tempCandidates.add((str1 + " " + st1.nextToken() + " " + st2.nextToken()).trim());
            }
        }
    }
    // clear the old candidates
    candidates.clear();
    // set the new ones
    candidates = new Vector<String>(tempCandidates);
    tempCandidates.clear();
}

  private void calculateFrequentItemsets(int n) {
    Vector<String> frequentCandidates = new Vector<String>(); // the frequent candidates for the current
itemset
```

```
FileInputStream file_in; // file input stream
BufferedReader data_in; // data input stream
FileWriter fw;
BufferedWriter file_out;
StringTokenizer st, stFile; // tokenizer for candidate and transaction
boolean match; // whether the transaction has all the items in an itemset
boolean trans[] = new boolean[numItems]; // array to hold a transaction so that can be checked
int count[] = new int[candidates.size()]; // the number of successful matches

try {
   // output file
   fw = new FileWriter(outputFile, true);
   file_out = new BufferedWriter(fw);
   // load the transaction file
   file_in = new FileInputStream(transaFile);
   data_in = new BufferedReader(new InputStreamReader(file_in));
   // for each transaction
   for (int i = 0; i < numTransactions; i++) {
      stFile = new StringTokenizer(data_in.readLine(), itemSep); // read a line from the file to the tokenizer
      // put the contents of that line into the transaction array
      for (int j = 0; j < numItems; j++) {
         trans[j] = (stFile.nextToken().compareToIgnoreCase(oneVal[j]) == 0); // if it is not a 0, assign the
value to true
      }
      // check each candidate
      for (int c = 0; c < candidates.size(); c++) {
         match = false; // reset match to false
         // tokenize the candidate so that we know what items need to be present for a match
         st = new StringTokenizer(candidates.get(c));
         // check each item in the itemset to see if it is present in the transaction
         while (st.hasMoreTokens()) {
            match = (trans[Integer.valueOf(st.nextToken()) - 1]);
            if (!match) // if it is not present in the transaction, stop checking
               break;
         }
         if (match) // if at this point it is a match, increase the count
            count[c]++;
      }
   }
   for (int i = 0; i < candidates.size(); i++) {
      // if the count% is larger than the minSup%, add the candidate to the frequent candidates
      if ((count[i] / (double) numTransactions) >= minSup) {
         frequentCandidates.add(candidates.get(i));
         // put the frequent itemset into the output file
         file_out.write(candidates.get(i) + "," + count[i] / (double) numTransactions + "\n");
      }
```

```
            }
            file_out.write("-\n");
            file_out.close();
        } catch (IOException e) {
            System.out.println(e);
        }
        // clear old candidates
        candidates.clear();
        // new candidates are the old frequent candidates
        candidates = new Vector<String>(frequentCandidates);
        frequentCandidates.clear();
    }
}
```

Input:

Config.txt

```
5
9
50
```

transa.txt

```
1 0 1 1 0
0 1 1 0 1
1 1 1 0 1
0 1 0 0 1
1 1 1 0 0
0 0 1 0 1
1 1 1 0 1
1 0 0 0 1
1 0 1 0 0
```

Output:

Command Prompt

```
Microsoft Windows [Version 10.0.19045.3448]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>cd desktop

C:\Users\HP\Desktop>javac Apriori.java

C:\Users\HP\Desktop>java Apriori
Default Configuration:
        Regular transaction file with ' ' item separator.
        Config File: C:\Users\HP\Desktop\Config.txt
        Transa File: C:\Users\HP\Desktop\transa.txt
        Output File: apriori-output.txt

Press 'C' to change the item separator, configuration file and transaction files
or any other key to continue.

Input configuration: 5 items, 9 transactions, minsup = 50.0%

Enter 'y' to change the value each row recognizes as a '1':
Apriori algorithm has started.

Frequent 1-itemsets
[1, 2, 3, 5]
Frequent 2-itemsets
[1 3]
Execution time is: 0.004 seconds.
```

CONCLUSION: Hence we have implemented association mining apriori using Java and WEKA.

DEPARTMENT OF COMPUTER ENGINEERING

EXPERIMENT NO. 10

AIM: To implement Clustering/Association Rule/ Classification Algorithms using R-tool.

OBJECTIVE: To understand the Clustering/Association Rule/ Classification Algorithms using R-tool.

THEORY:

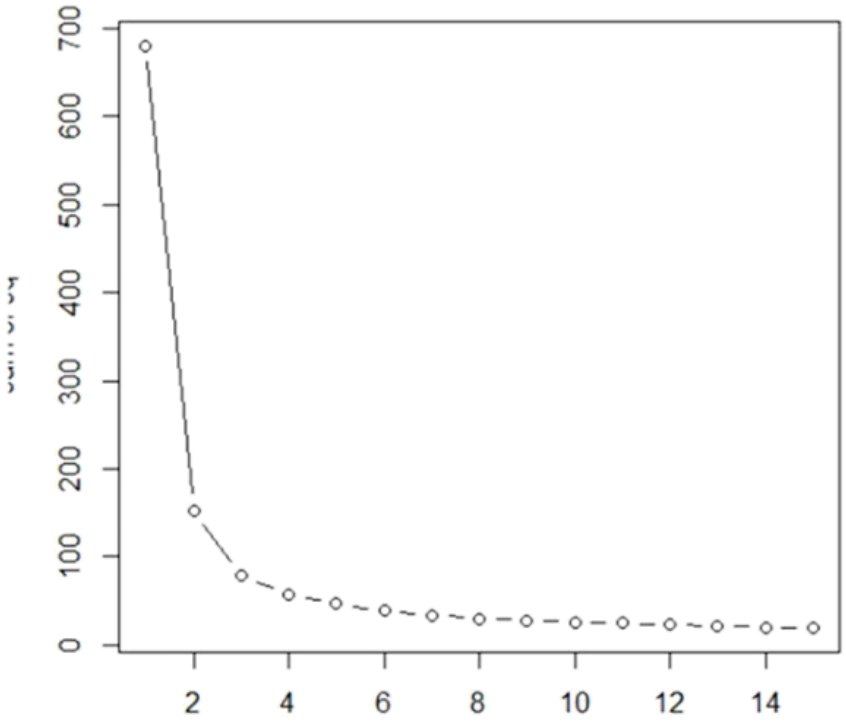
K-means Clustering Algorithm:

```
> library(plyr)
> library(ggplot2)
> library(cluster)
> library(lattice)
> library(graphics)
> library(grid)
> library(gridExtra)
```

```
> kmdata_orig=as.matrix(iris_input[,c("sepal_length","sepal_width","petal_length","petal_width")])
> kmdata<-kmdata_orig[,1:4]
> kmdata[1:10,]
      sepal_length sepal_width petal_length petal_width
 [1,]          5.1         3.5          1.4         0.2
 [2,]          4.9         3.0          1.4         0.2
 [3,]          4.7         3.2          1.3         0.2
 [4,]          4.6         3.1          1.5         0.2
 [5,]          5.0         3.6          1.4         0.2
 [6,]          5.4         3.9          1.7         0.4
 [7,]          4.6         3.4          1.4         0.3
 [8,]          5.0         3.4          1.5         0.2
 [9,]          4.4         2.9          1.4         0.2
[10,]          4.9         3.1          1.5         0.1
> wss<-numeric(15)
> for(k in 1:15)wss[k]<-sum(kmeans(kmdata,centers = k,nstart = 25)$withinss)
> km=kmeans(kmdata,3,nstart = 25)
> km
K-means clustering with 3 clusters of sizes 62, 50, 38

Cluster means:
  sepal_length sepal_width petal_length petal_width

Cluster means:
  sepal_length sepal_width petal_length petal_width
1     5.901613    2.748387     4.393548    1.433871
2     5.006000    3.418000     1.464000    0.244000
3     6.850000    3.073684     5.742105    2.071053

Clustering vector:
  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [46] 2 2 2 2 2 2 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [91] 1 1 1 1 1 1 1 1 1 1 3 1 3 3 3 3 3 1 3 3 3 3 3 3 1 1 3 3 3 3 1 3 1 3 1 3 3 1 1 3 3 3 3 3 1 3
[136] 3 3 3 1 3 3 3 1 3 3 3 3 1 3 3 1

Within cluster sum of squares by cluster:
[1] 39.82097 15.24040 23.87947
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
```

```
> plot(1:15,wss,type = "b",xlab = "no of clusters",ylab = "sum of sq")
```

DEPARTMENT OF COMPUTER ENGINEERING



DEPARTMENT OF COMPUTER ENGINEERING

Naive-Bayes Classification Algorithm:

```
> sample<-read.table("C:/Users/complab/Documents/weather.csv",header = TRUE,sep = ",")
> traindata<-as.data.frame(sample[1:13,])
> testdata<-as.data.frame(sample[14,])
> traindata<-as.data.frame(sample[1:14,])
> testdata<-as.data.frame(sample[15,])
> traindata
    outlook temperature humidity windy play
1     sunny         hot     high FALSE   no
2     sunny         hot     high  TRUE   no
3  overcast         hot     high FALSE  yes
4     rainy        mild     high FALSE  yes
5     rainy        cool   normal FALSE  yes
6     rainy        cool   normal  TRUE   no
7  overcast        cool   normal  TRUE  yes
8     sunny        mild     high FALSE   no
9     sunny        cool   normal FALSE  yes
10    rainy        mild   normal FALSE  yes
11    sunny        mild   normal  TRUE  yes
12 overcast        mild     high  TRUE  yes
13 overcast         hot   normal FALSE  yes
14    rainy        mild     high  TRUE

> traindata<-as.data.frame(sample[1:13,])
> testdata<-as.data.frame(sample[14,])

> install.packages("e1071")
trying URL 'https://cran.rstudio.com/bin/windows

> library("e1071")

> model1<-naiveBayes(play ~ outlook+temperature+humidity+windy,traindata)
```

DEPARTMENT OF COMPUTER ENGINEERING

```
> model1

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
                 no       yes
0.0000000 0.3076923 0.6923077

Conditional probabilities:
     outlook
Y      overcast     rainy     sunny

  no  0.0000000 0.2500000 0.7500000
  yes 0.4444444 0.3333333 0.2222222

     temperature
Y          cool       hot      mild

  no  0.2500000 0.5000000 0.2500000
  yes 0.3333333 0.2222222 0.4444444

     humidity
Y          high    normal

  no  0.7500000 0.2500000
  yes 0.3333333 0.6666667

     windy
Y         FALSE      TRUE

  no  0.5000000 0.5000000
  yes 0.6666667 0.3333333

> results<-predict(model1,testdata)
> results
[1] yes
Levels:  no yes
```

DEPARTMENT OF COMPUTER ENGINEERING

Association Rule Mining Apriori Algorithm:

```
> library('arulesViz')

> summary(Groceries)
transactions as itemMatrix in sparse format with
 9835 rows (elements/itemsets/transactions) and
 169 columns (items) and a density of 0.02609146

most frequent items:
      whole milk other vegetables       rolls/buns             soda           yogurt
            2513             1903             1809             1715             1372
         (other)
           34055

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19
2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55   46   29   14   14
  20   21   22   23   24   26   27   28   29   32
   9   11    4    6    1    1    1    1    3    1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   3.000   4.409   6.000  32.000

includes extended item information - examples:
       labels  level2           level1
1 frankfurter sausage meat and sausage
2     sausage sausage meat and sausage
3  liver loaf sausage meat and sausage
>

> itemsets<- apriori(Groceries,parameter = list(minlen=1,maxlen=1,support=0.02,target="frequent itemsets"))

> summary(itemsets)
set of 59 itemsets

most frequent items:
frankfurter     sausage         ham        meat     chicken     (Other)
          1           1           1           1           1          54

element (itemset/transaction) length distribution:sizes
 1
59

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1       1       1       1       1       1

summary of quality measures:
    support           count
 Min.   :0.02105   Min.   : 207.0
 1st Qu.:0.03015   1st Qu.: 296.5
 Median :0.04809   Median : 473.0
 Mean   :0.06200   Mean   : 609.8
 3rd Qu.:0.07666   3rd Qu.: 754.0
 Max.   :0.25552   Max.   :2513.0

includes transaction ID lists: FALSE

mining info:
      data ntransactions support confidence
 Groceries          9835    0.02          1
```

DEPARTMENT OF COMPUTER ENGINEERING

```
> itemsets<- apriori(Groceries,parameter = list(minlen=2,maxlen=2,support=0.02,target="frequent itemsets"))

> summary(itemsets)
set of 61 itemsets

most frequent items:
      whole milk other vegetables           yogurt       rolls/buns             soda          (other)
              25                17                9                9                9               53

element (itemset/transaction) length distribution:sizes
 2
61

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      2       2       2       2       2       2

summary of quality measures:
    support           count
 Min.   :0.02003   Min.   :197.0
 1st Qu.:0.02227   1st Qu.:219.0
 Median :0.02613   Median :257.0
 Mean   :0.02951   Mean   :290.3
 3rd Qu.:0.03223   3rd Qu.:317.0
 Max.   :0.07483   Max.   :736.0

includes transaction ID lists: FALSE

mining info:
      data ntransactions support confidence
 Groceries          9835    0.02          1

> itemsets<- apriori(Groceries,parameter = list(minlen=3,maxlen=3,support=0.02,target="frequent itemsets"))

> summary(itemsets)
set of 2 itemsets

most frequent items:
other vegetables       whole milk  root vegetables           yogurt      frankfurter          (Other)
               2                2                1                1                0                0

element (itemset/transaction) length distribution:sizes
3
2

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      3       3       3       3       3       3

summary of quality measures:
    support           count
 Min.   :0.02227   Min.   :219.0
 1st Qu.:0.02250   1st Qu.:221.2
 Median :0.02272   Median :223.5
 Mean   :0.02272   Mean   :223.5
 3rd Qu.:0.02295   3rd Qu.:225.8
 Max.   :0.02318   Max.   :228.0

includes transaction ID lists: FALSE

mining info:
      data ntransactions support confidence
 Groceries          9835    0.02          1

> rules<-apriori(Groceries,parameter = list(support=0.001,confidence=0.6,target="rules"))
```

DEPARTMENT OF COMPUTER ENGINEERING

```
> summary(rules)
set of 2918 rules

rule length distribution (lhs + rhs):sizes
   2    3    4    5    6
   3  490 1765  626   34

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   4.000   4.000   4.068   4.000   6.000

summary of quality measures:
    support           confidence          lift            count
 Min.   :0.001017   Min.   :0.6000   Min.   : 2.348   Min.   :10.00
 1st Qu.:0.001118   1st Qu.:0.6316   1st Qu.: 2.668   1st Qu.:11.00
 Median :0.001220   Median :0.6818   Median : 3.168   Median :12.00
 Mean   :0.001480   Mean   :0.7028   Mean   : 3.450   Mean   :14.55
 3rd Qu.:0.001525   3rd Qu.:0.7500   3rd Qu.: 3.692   3rd Qu.:15.00
 Max.   :0.009354   Max.   :1.0000   Max.   :18.996   Max.   :92.00

mining info:
      data ntransactions support confidence
 Groceries          9835   0.001        0.6
```

CONCLUSION: Hence, we have implemented Clustering/Association Rule/ Classification Algorithms using R-tool.

EXPERIMENT NO. 11

TITLE: Implementation of Page Rank/Hits Algorithm.

AIM: Write a program **t**o implement Page Rank/Hits Algorithm using Java and WEKA tool

THEORY:

Page Rank:

The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collections of documents of any size. It is assumed in several research papers that the distribution is evenly divided among all documents in the collection at the beginning of the computational process. The PageRank computations require several passes, called “iterations”, through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value.

Simplified algorithm:

Assume a small universe of four web pages: A, B, C, and D. Links from a page to itself, or multiple outbound links from one single page to another single page, are ignored. PageRank is initialized to the same value for all pages. In the original form of PageRank, the sum of PageRank over all pages was the total number of pages on the web at that time, so each page in this example would have an initial value of 1. However, later versions of PageRank, and the remainder of this section, assume a probability distribution between 0 and 1. Hence the initial value for each page in this example is 0.25.

The PageRank transferred from a given page to the targets of its outbound links upon the next iteration is divided equally among all outbound links.

If the only links in the system were from pages B, C, and D to A, each link would transfer 0.25 PageRank to A upon the next iteration, for a total of 0.75.

PR(A) = PR(B) + PR(C) + PR(D).\,

Suppose instead that page B had a link to pages C and A, page C had a link to page A, and page D had links to all three pages. Thus, upon the first iteration, page B would transfer half of its existing value, or 0.125, to page A and the other half, or 0.125, to page C. Page C would transfer all of its existing value, 0.25, to the only page it links to, A. Since D had three outbound links, it would transfer one-third of its existing value, or approximately 0.083, to A. At the completion of this iteration, page A will have a PageRank of approximately 0.458.

PR(A)={\frac {PR(B)}{2}}+{\frac {PR(C)}{1}}+{\frac {PR(D)}{3}}.\,

In other words, the PageRank conferred by an outbound link is equal to the document’s own PageRank score divided by the number of outbound links L( ).

PR(A)={\frac {PR(B)}{L(B)}}+{\frac {PR(C)}{L(C)}}+{\frac {PR(D)}{L(D)}}.\, In the general case, the PageRank value for any page u can be expressed as:

PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}, i.e. the PageRank value for a page u is dependent on the

PageRank values for each page v contained in the set Bu (the set containing all pages linking to page u), divided by the number L(v) of links from page v. The algorithm involves a damping factor for the calculation of the PageRank. It is like the income tax which the govt extracts from one despite paying him itself.

HITS Algorithm:

Hyperlink Induced Topic Search (HITS) Algorithm is a Link Analysis Algorithm that rates webpages, developed by Jon Kleinberg. This algorithm is used to the web link-structures to discover and rank the webpages relevant for a particular search. HITS uses hubs and authorities to define a recursive relationship between webpages.

- Given a query to a Search Engine, the set of highly relevant web pages are called Roots. They are potential Authorities.

- Pages which are not very relevant but point to pages in the Root are called Hubs. Thus, an Authority is a page that many hubs link to whereas a Hub is a page that links to many authorities.

Algorithm:

→ Let number of iterations be k.

→ Each node is assigned a Hub score = 1 and an Authority score = 1.

→ Repeat k times:

- Hub update : Each node's Hub score = (Authority score of each node it points to).

- Authority update : Each node's Authority score = (Hub score of each node pointing to it).

- Normalize the scores by dividing each Hub score by square root of the sum of the squares of all Hub scores, and dividing each Authority score by square root of the sum of the squares of all Authority scores. (optional)

LAB EXERCISE:

1) **Page Rank Code**

```
import networkx as nx

G=nx.barabasi_albert_graph(60,41)

pr=nx.pagerank(G,0.4)

pr

def pagerank(G, alpha=0.85, personalization=None,
```

```
                max_iter=100, tol=1.0e-6, nstart=None, weight='weight',
                dangling=None):
    if len(G) == 0:
        return {}
    if not G.is_directed():
        D = G.to_directed()
    else:
        D = G
    # Create a copy in (right) stochastic form
    W = nx.stochastic_graph(D, weight=weight)
    N = W.number_of_nodes()
    # Choose fixed starting vector if not given
    if nstart is None:
        x = dict.fromkeys(W, 1.0 / N)
    else:
        # Normalized nstart vector
        s = float(sum(nstart.values()))
        x = dict((k, v / s) for k, v in nstart.items())
    if personalization is None:
        # Assign uniform personalization vector if not given
        p = dict.fromkeys(W, 1.0 / N)
    else:
        missing = set(G) - set(personalization)
        if missing:
            raise NetworkXError('Personalization dictionary must have a value for every node
Missing nodes %s' % missing)
        s = float(sum(personalization.values()))
        p = dict((k, v / s) for k, v in personalization.items())
    if dangling is None:
```

```
        # Use personalization vector if dangling vector not specified
        dangling_weights = p
    else:
        missing = set(G) - set(dangling)
        if missing:
            raise NetworkXError('Dangling node dictionary must have a value for every node
Missing nodes %s' % missing)
        s = float(sum(dangling.values()))
        dangling_weights = dict((k, v/s) for k, v in dangling.items())
    dangling_nodes = [n for n in W if W.out_degree(n, weight=weight) == 0.0]
    # power iteration: make up to max_iter iterations
    for _ in range(max_iter):
        xlast = x
        x = dict.fromkeys(xlast.keys(), 0)
        danglesum = alpha * sum(xlast[n] for n in dangling_nodes)
        for n in x:


            # this matrix multiply looks odd because it is
            # doing a left multiply x^T=xlast^T*W
            for nbr in W[n]:
                x[nbr] += alpha * xlast[n] * W[n][nbr][weight]
            x[n] += danglesum * dangling_weights[n] + (1.0 - alpha) * p[n]


        # check convergence, l1 norm
        err = sum([abs(x[n] - xlast[n]) for n in x])
        if err < N*tol:
            return x
    raise NetworkXError('pagerank: power iteration failed to converge in %d iterations.' %
max_iter)
```

OUTPUT:

```
{0: 0.012774147598875784, 1: 0.013359655345577266, 2: 0.013157355731377924,
3: 0.012142198569313045, 4: 0.013160014506830858, 5: 0.012973342862730735,
 6: 0.012166706783753325, 7: 0.011985935451513014, 8: 0.012973502696061718,
9: 0.013374146193499381, 10: 0.01296354505412387, 11: 0.013163220326063332,
 12: 0.013368514624403237, 13: 0.013169335617283102, 14: 0.012752071800520563,
15: 0.012951601882210992, 16: 0.013776032065400283, 17: 0.012356820581336275,
18: 0.013151652554311779, 19: 0.012551059531065245, 20: 0.012583415756427995,
 21: 0.013574117265891684, 22: 0.013167552803671937, 23: 0.013165528583400423,
 24: 0.012584981049854336, 25: 0.013372989228254582, 26: 0.012569416076848989,
 27: 0.013165322299539031, 28: 0.012954300960607157, 29: 0.012776091973397076,
 30: 0.012771016515779594, 31: 0.012953404860268598, 32: 0.013364947854005844,
33: 0.012370004022947507, 34: 0.012977539153099526, 35: 0.013170376268827118,
 36: 0.012959579020039328, 37: 0.013155319659777197, 38: 0.013567147133137161,
 39: 0.012171548109779459, 40: 0.01296692767996657, 41: 0.028089802328702826,
 42: 0.027646981396639115, 43: 0.027300188191869485, 44: 0.02689771667021551,
 45: 0.02650459107960327, 46: 0.025971186884778535, 47: 0.02585262571331937,
48: 0.02565482923824489, 49: 0.024939722913691394, 50: 0.02458271197701402,
 51: 0.024263128557312528, 52: 0.023505217517258568, 53: 0.023724311872578157,
 54: 0.02312908947188023, 55: 0.02298716954828392, 56: 0.02270220663300396,
 57: 0.022060403216132875, 58: 0.021932442105075004, 59: 0.021643288632623502}
```

2) HITS Code

```
# importing modules
import networkx as nx
import matplotlib.pyplot as plt

G = nx.DiGarph()

G.add_edges_from([('A', 'D'), ('B', 'C'), ('B', 'E'), ('C', 'A'),
                  ('D', 'C'), ('E', 'D'), ('E', 'B'), ('E', 'F'),
                  ('E', 'C'), ('F', 'C'), ('F', 'H'), ('G', 'A'),
                  ('G', 'C'), ('H', 'A')])

plt.figure(figsize =(10, 10))
```

```
nx.draw_networkx(G, with_labels = True)

hubs, authorities = nx.hits(G, max_iter = 50, normalized = True)
# The in-built hits function returns two dictionaries keyed by nodes
# containing hub scores and authority scores respectively.

print("Hub Scores: ", hubs)
    print("Authority Scores: ", authorities)
```

OUTPUT:

```
Hub Scores:  {'A': 0.04642540386472174, 'D': 0.133660375232863,
              'B': 0.15763599440595596, 'C': 0.037389132480584515,
              'E': 0.2588144594158868, 'F': 0.15763599440595596,
              'H': 0.037389132480584515, 'G': 0.17104950771344754}

Authority Scores:  {'A': 0.10864044085687284, 'D': 0.13489685393050574,
                    'B': 0.11437974045401585, 'C': 0.3883728005172019,
                    'E': 0.06966521189369385, 'F': 0.11437974045401585,
                    'H': 0.06966521189369385, 'G': 0.0}
```

CONCLUSION: Hence, we have implemented Page Rank/Hits Algorithm.