EXPERIMENT NO. 1

TITLE: One Case Study on Building a Sales Data Warehouse/Data Mart

AIM:
Write a detailed problem statement and design dimensional modelling (creation of Star Schema and Snowflake Schema) for a Sales Data Warehouse.

THEORY

A **Sales Data Warehouse** is a centralized storage system that integrates historical sales data from multiple sources to support decision-making and analysis. **Dimensional Modelling** is used to design this data warehouse in a way that is easy to understand, query, and analyze.

Dimensional Modelling, developed by Ralph Kimball, uses two main components:

- **Fact Table** – Stores measurable data (e.g., Sales Amount, Quantity Sold, Profit).
- **Dimension Tables** – Store descriptive information (e.g., Customer, Product, Time, Location).

This structure allows quick generation of reports such as daily sales, product-wise performance, customer purchase trends, etc.

ELEMENTS OF DIMENSIONAL DATA MODEL

1. **Fact:**
   Represents the measurable business event.
   Example: Total Sales Amount, Quantity Sold, Discounts, and Profit.
2. **Dimension:**
   Provides the context for the facts:
   - **Who** – Customer
   - **What** – Product
   - **Where** – Store Location
   - **When** – Date of Sale
3. **Attributes:**
   Characteristics of a dimension.
   - Product Dimension: Product ID, Product Name, Category, Price
   - Customer Dimension: Customer Name, Region, Contact Details
   - Time Dimension: Day, Month, Quarter, Year
4. **Fact Table:**
   Contains sales transactions and foreign keys linking to dimensions.
5. **Dimension Table:**
   Contains descriptive information that provides context for the fact table.

STEPS OF DIMENSIONAL MODELLING

1. **Identify the Business Process:** Sales Transactions.

2. **Identify the Grain:** Each record represents a single sales transaction (daily granularity).
3. **Identify the Dimensions:** Product, Customer, Time, Location.
4. **Identify the Facts:** Sales Amount, Units Sold, Discount, Profit.
5. **Build the Schema:** Implement Star Schema and Snowflake Schema.

EXAMPLE STAR SCHEMA

- **Fact Table:** `Fact_Sales`
  - Attributes: Sales_ID, Date_Key, Product_Key, Customer_Key, Location_Key, Sales_Amount, Units_Sold, Discount, Profit
- **Dimension Tables:**
  - `Dim_Product (Product_Key, Product_Name, Category, Price)`
  - `Dim_Customer (Customer_Key, Customer_Name, Region, Contact)`
  - `Dim_Time (Date_Key, Day, Month, Quarter, Year)`
  - `Dim_Location (Location_Key, Country, State, City, Store_Name)`

This structure places `Fact_Sales` at the center with dimensions surrounding it like a star.

SNOWFLAKE SCHEMA

In the snowflake schema, some dimensions are normalized to remove redundancy.
For example:

- `Dim_Location` can be split into separate tables: `Country`, `State`, and `City` linked by foreign keys.
- This provides a more organized but slightly more complex design.

CONCLUSION:

Hence, we have successfully understood and designed a Sales Data Warehouse using Dimensional Modelling with Star Schema and Snowflake Schema. This model helps generate reports such as daily sales, monthly revenue trends, and product performance analysis effectively

EXPERIMENT NO. 2

TITLE: Implementation of Sales Dimension Table and Fact Table

AIM: Write an SQL query to implement Dimension Table and Fact Table for a Sales 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 snowflake schema, surrounded by dimension tables. A fact table typically has two types of columns:

Fact columns (quantitative measures such as sales amount, quantity sold, discount, profit).Foreign keys linking to dimension tables.

The primary key of a fact table is usually a composite key made up of all its foreign keys. Fact tables contain the core data of the data warehouse and store measures such as:

Additive (e.g., total sales amount)

Semi-additive (e.g., account balance)

Non-additive (e.g., ratios, percentages).

Fact tables are defined by their grain, i.e., the most atomic level at which facts are recorded.

For example, the grain of a SALES Fact Table might be:“Sales transaction by Day by Product by Store.”

Each record is uniquely identified by a date, product, and store. Other dimensions such as customer or region can also be linked, providing higher-level aggregation (e.g., total sales by region).

Dimension Table:Dimension tables are companion tables to the fact table. While the fact table contains numeric facts and foreign keys, the dimension tables contain descriptive attributes that provide context for analysis.For Sales, the key dimension tables include:

DimProduct → product details (name, category, brand, price range).

DimCustomer → customer details (name, location, customer segment).

DimStore → store details (store ID, store name, region).

DimDate → calendar details (day, month, quarter, year)

DimDate → calendar details (day, month, quarter, year).

DimSalesperson → sales representative details (name, region, target).

Dimension attributes are typically:

Verbose (meaningful labels like “Electronics” instead of codes).

Descriptive and complete (no missing values).

Discretely valued (one value per row).

Quality assured (no duplicates or invalid values).

Dimension table rows are uniquely identified by a single key field (usually a surrogate integer key). This key is only meaningful for linking with fact tables.DimCustomer

CustomerKey	CustomerName	Gender	AgeGroup	Location	Segment
301	Raj Kumar	M	25-34	Mumbai	Retail
302	Priya Sharma	F	18-24	Delhi	Wholesale

DimDate

DateKey	FullDate	Day	Month	Quarter	Year
101	2025-09-25	25	9	3	2025
102	2025-09-26	26	9	3	2025

DimProduct

ProductKey	ProductName	Category	Brand	PriceRange
201	Laptop	Electronics	Dell	High-End
202	Smartphone	Electronics	Samsung	Mid-Range

DimSalesperson

SalespersonKey	SalespersonName	Region	Target
501	Amit Verma	West	100000
502	Neha Singh	North	120000

StoreKey	StoreName	City	Region	StoreType
401	Store A	Mumbai	West	Flagship
402	Store B	Delhi	North	Franchise

DimStore

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

Orders

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 covers 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 a multidimensional view of data, we will discuss OLAP operations in multidimensional data.

Here is the list of OLAP operations −

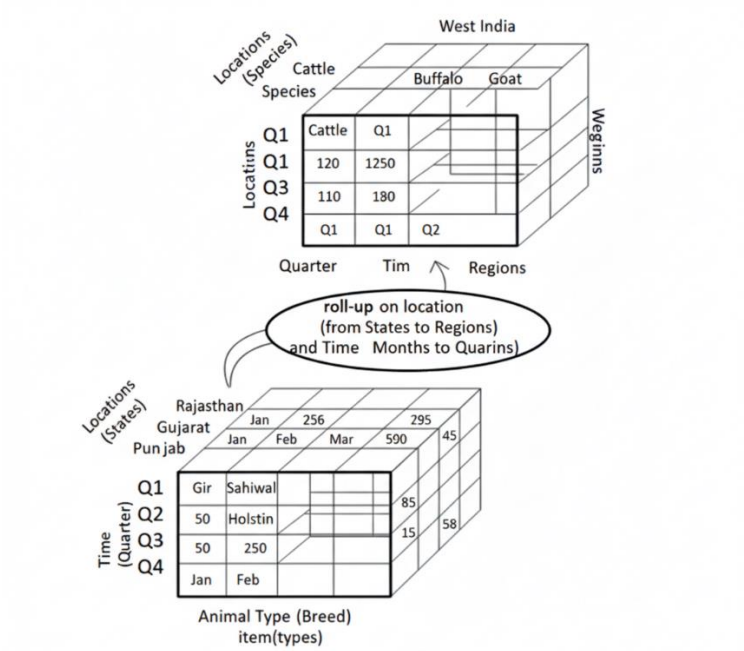
· Roll-up
· Drill-down
· Slice and dice
· Pivot (rotate)

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 "department < departmentname< measure< cost".

· On rolling up, the data is aggregated by ascending the location hierarchy from the level of department to the level of cost.

· The data is grouped into cost rather than departments.

· 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.

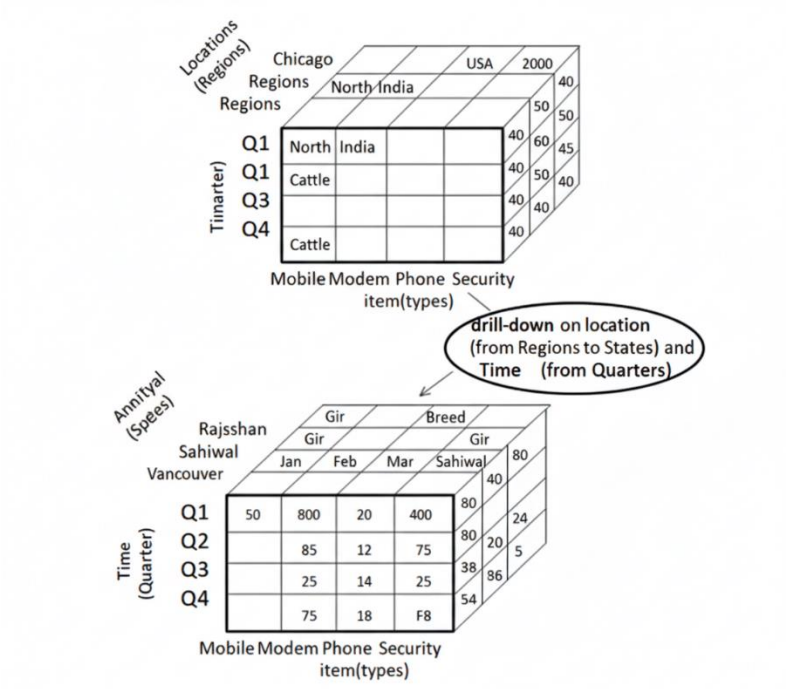
· Drill-down is performed by stepping down a concept hierarchy for the dimension time. · Initially the concept hierarchy was "doctor < doctorname < department < cardiology."

· On drilling down, the doctordimension is descended from the level of the department to the level of cardiology. · 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 "department" using the criterion time = "neurology".

· It will form a new sub-cube by selecting one or more dimensions.

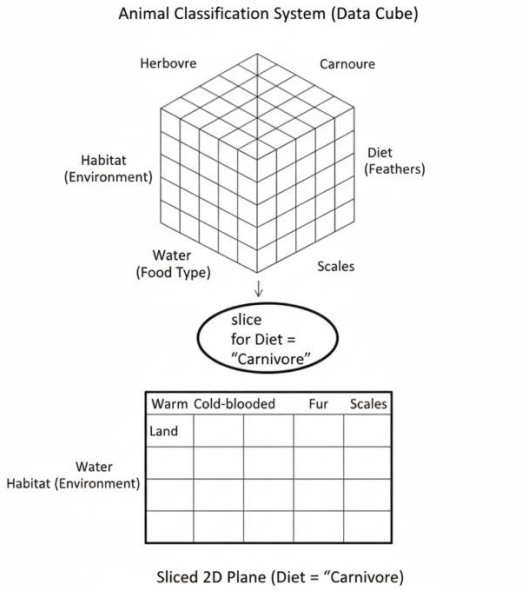


Dice

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.

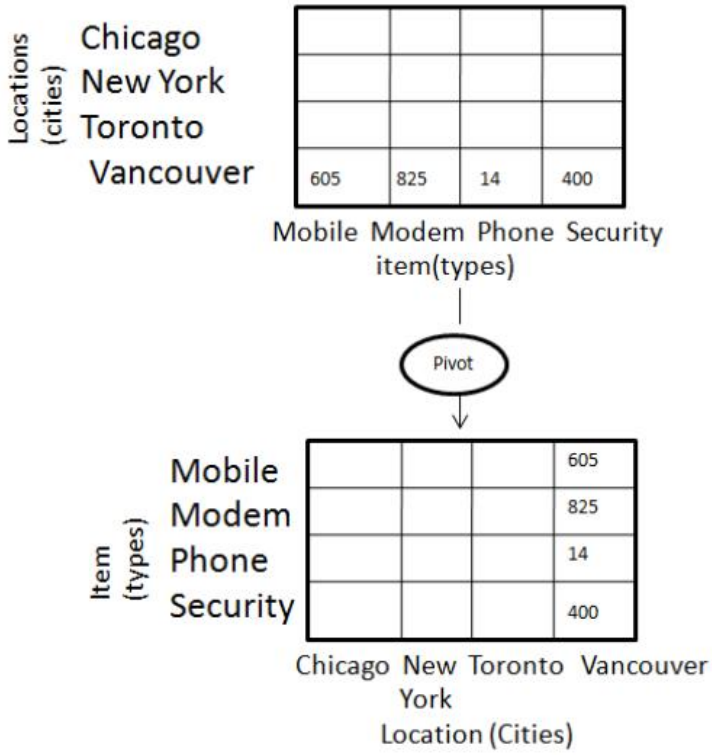
The dice operation on the cube based on the following selection criteria involves three dimensions. ·

(department = "cardiology" or "neurology") (year = "2025" or "2024")



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

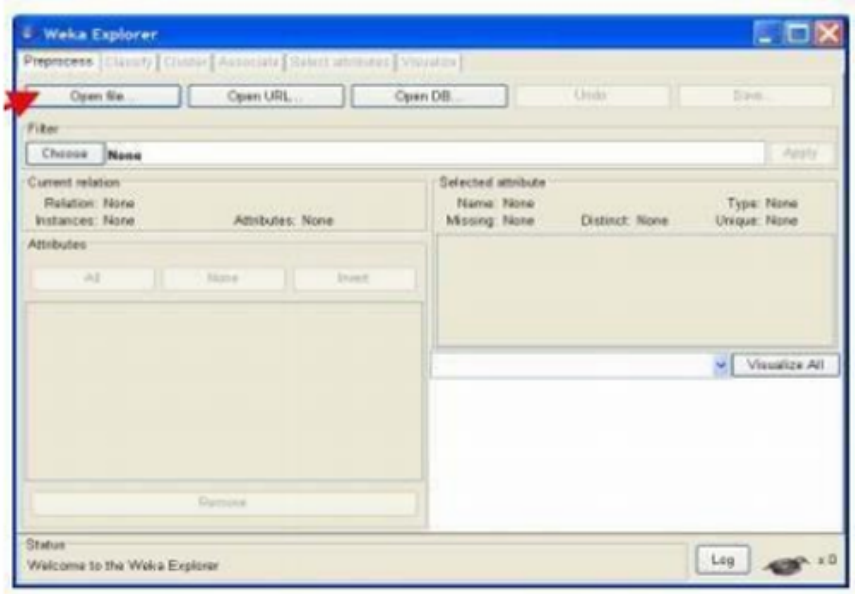
THEORY:

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 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:

1. Regression
2. Clustering
3. Association
4. Data Pre-Processing
5. Classification
6. Visualization

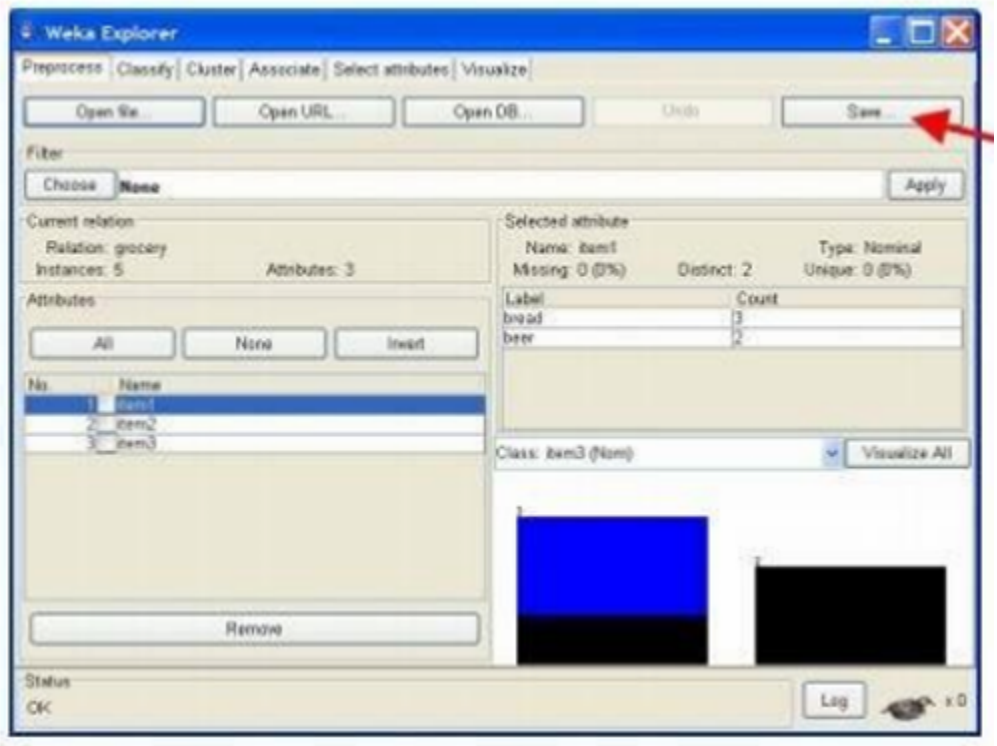
Opening a file from a local file system:
Click on the "Open file…" button.



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

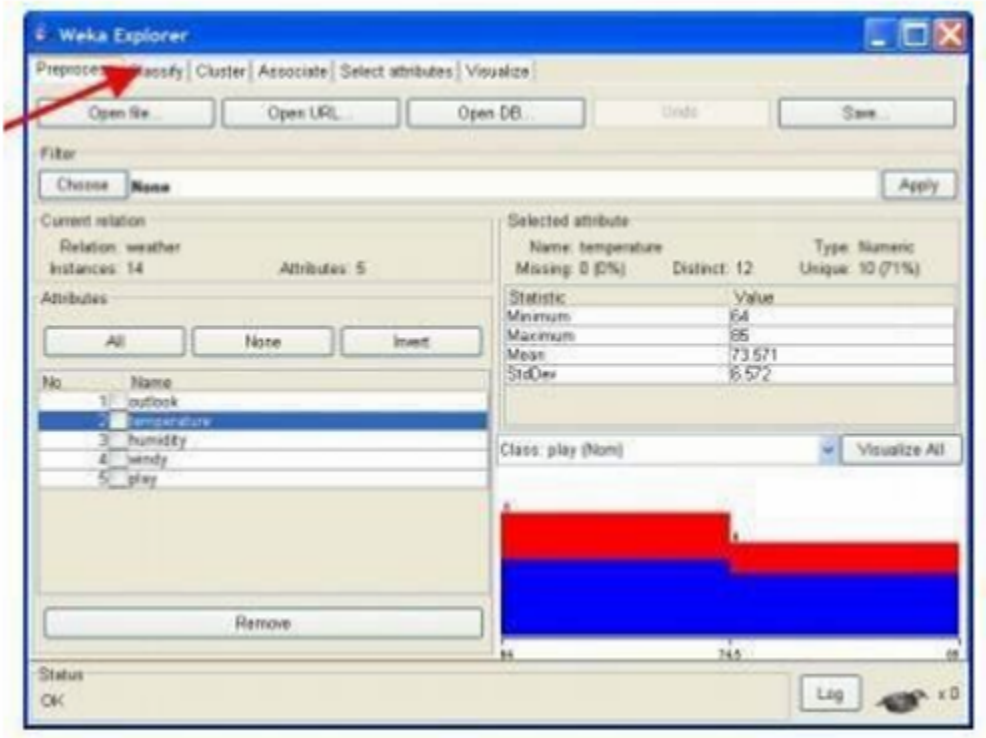
Some databases have the ability to save data in CSV format. In this case, you can select a CSV file from the local filesystem.

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.

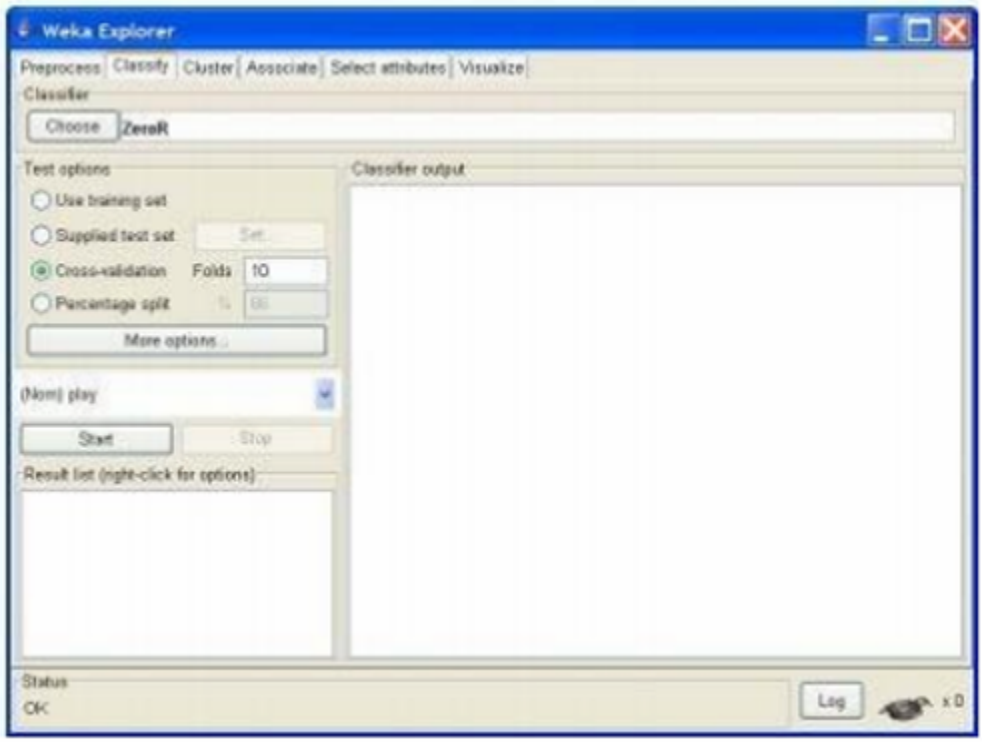


Building "Classifiers":
Classifiers in WEKA are the models for predicting nominal or numeric quantities. The learning schemes available in WEKA include decision trees and lists, instance-based classifiers, support vector machines, multi-layer perceptrons, logistic regression, and bayes' nets. "Meta"- 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.



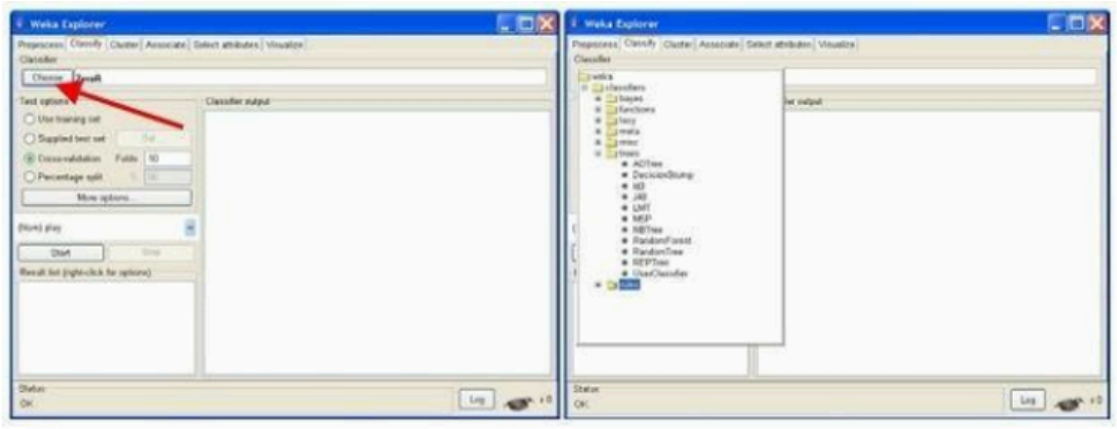
'Classify' window comes up on the screen.



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.

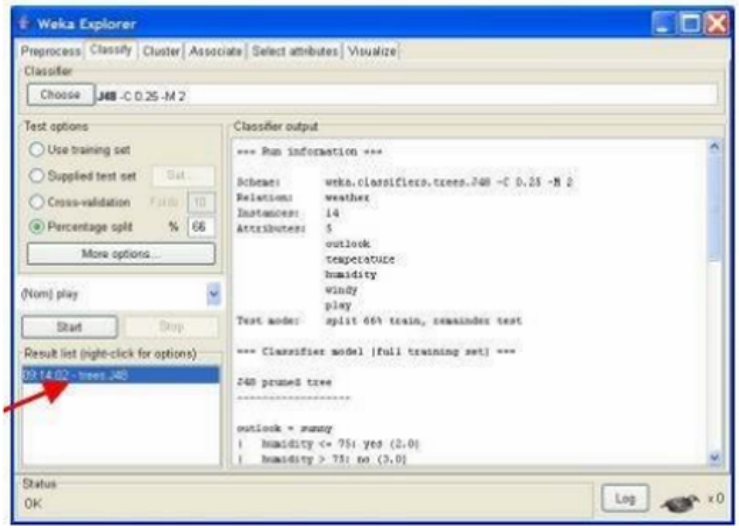
Choosing a Classifier:

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

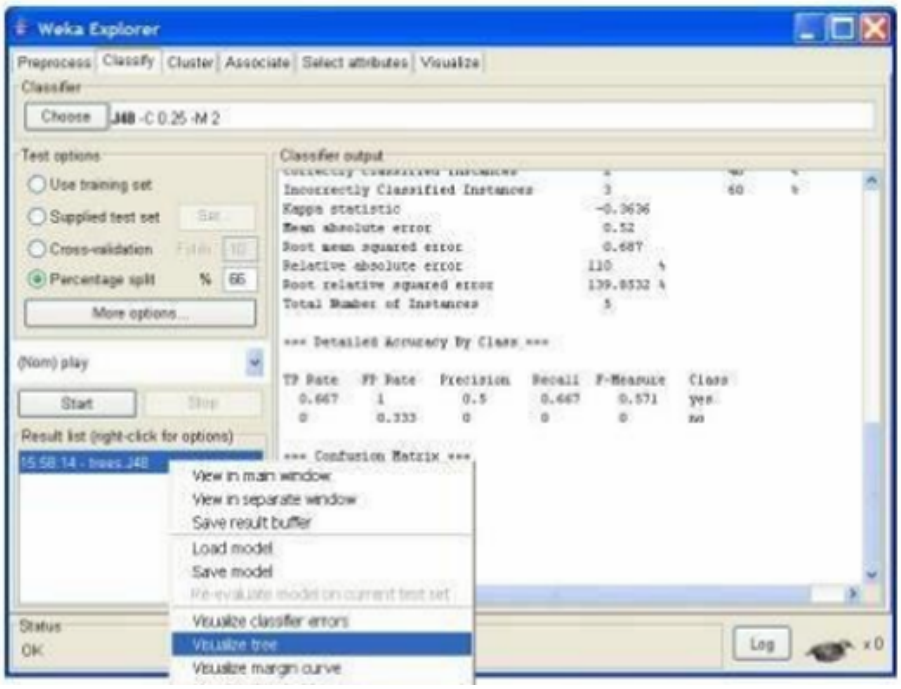


Visualization of Results:

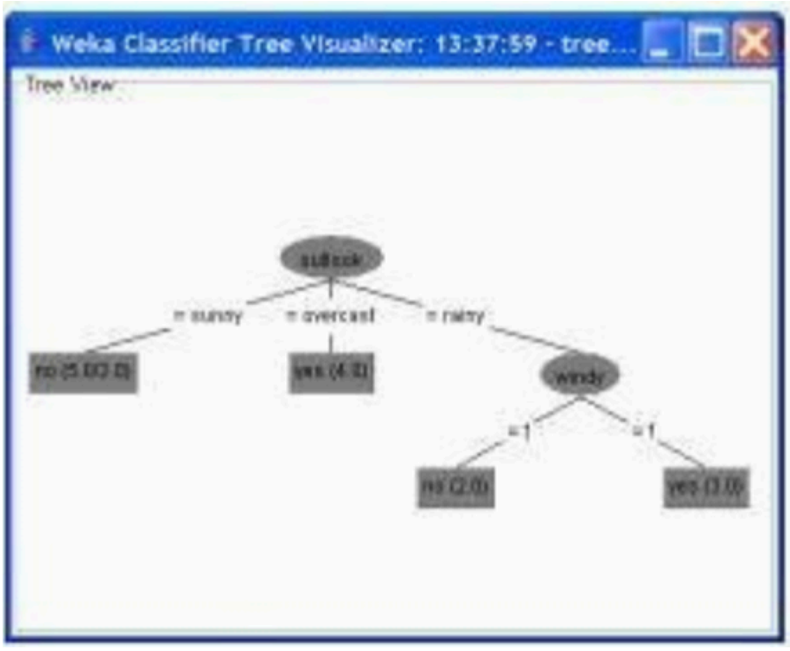
After training a classifier, the result list adds an entry.



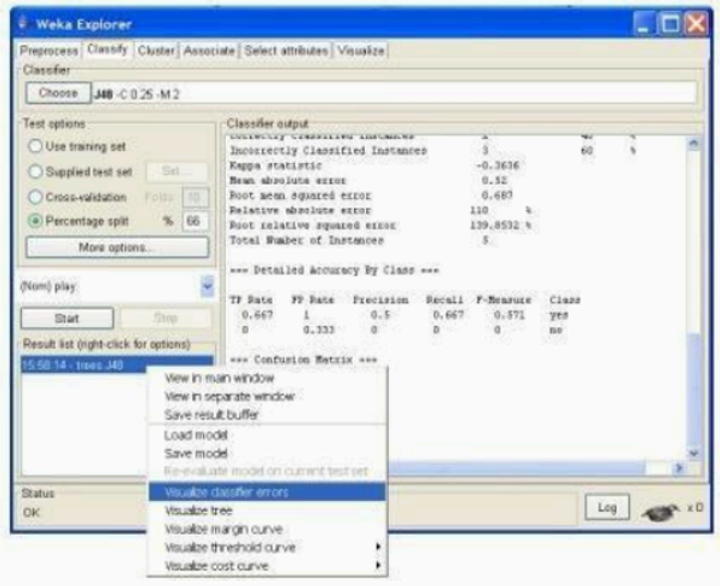
WEKA lets you see a graphical representation of the classification tree. Right-click on the entry in 'Result list' for which you would like to visualize a tree. It invokes a menu containing the following items:



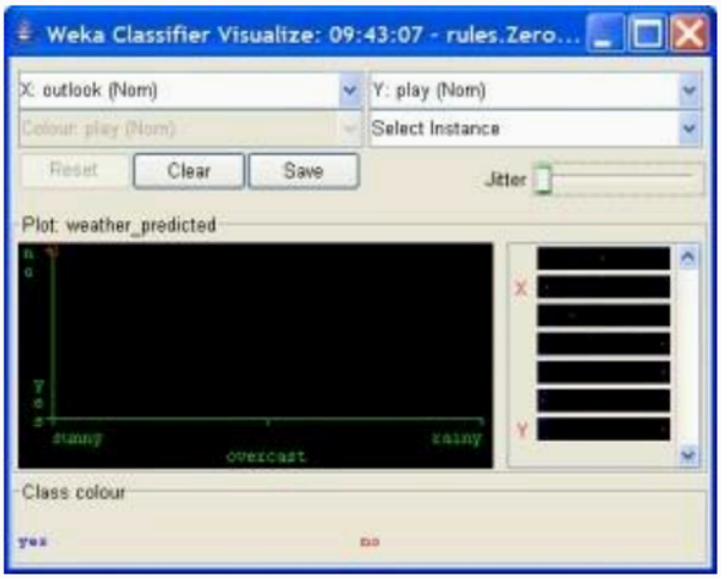
Select the item 'Visualize tree'; a new window comes up to the screen displaying the tree.



WEKA also lets you 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. The 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 sets 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 classifications into the 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.

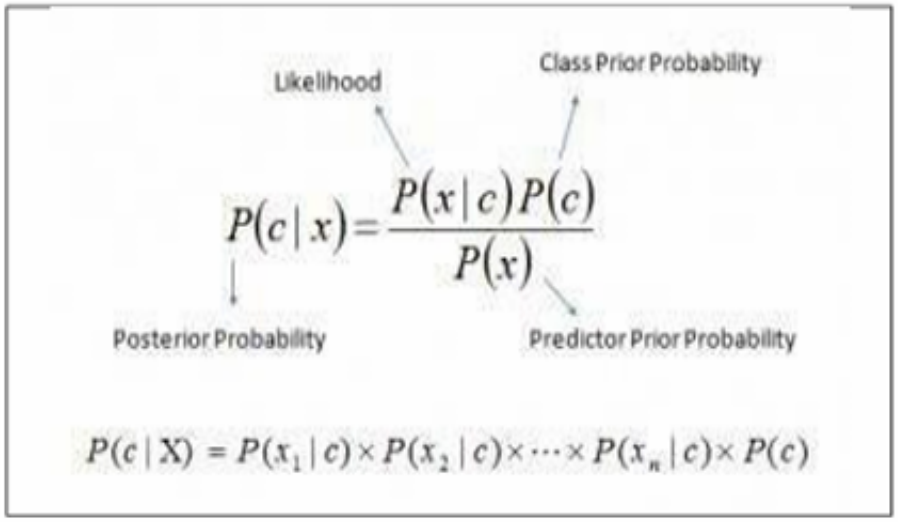
XLMiner solving data mining tasks, concentrating on tasks that are relevant for client relationship management. XLMiner can solve classification, forecasting, clustering, time series analysis and associative rules tasks. And the classification task is the one that is most emphasized. This task can be 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'.

The 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 of 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 conditions. Let's follow the below steps to perform it.

Step 1: Convert the data set into a frequency table
Step 2: Create a likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.
Step 3: Now, use the 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 the weather is sunny. Is this statement correct?
We can solve it using the 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.

<u>Pros and Cons of Naive Bayes</u>

Pros

1. It is easy and fast to predict the class of a 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 a categorical variable has a category (in the test data set), which was not observed in the training data set, then the 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.

Application of Naive Bayes Algorithm

- 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 features. 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:

Java Program:

```
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();
   if (input == null) {
      System.out.println("Empty file");
      return;
   }
   StringTokenizer line = new StringTokenizer(input);
   no_attr = line.countTokens() - 1; // last token is class
   fileArray = new String[100][no_attr + 1];

   // header or first row stored at row 0
   for (int i = 0; i <= no_attr; i++)
      fileArray[no_rows][i] = line.nextToken();

   // now read remaining rows; data rows will be 1..no_rows
   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();
   }
   // no_rows currently counts data rows (1..no_rows). header is row 0.
   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;
    }
    return false;
}

void getAllvalues() {
    values = new String[100][no_attr + 1];
    // col-major traversal
    for (int i = 0; i <= no_attr; i++) {
        count = 0;
        for (int j = 1; j <= no_rows; j++) { // data rows start at 1
            String temp = fileArray[j][i];
            if (temp == null)
                continue;
            if (in_values(i, temp))
                continue;
            values[count++][i] = temp;
        }
    }
}

int getlen(int col_no) // returns no of distinct values in particular col_no in array 'values'
{
    int i = 0;
    while (i < values.length && 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(fileArray[i][j] + " ");
        }
        System.out.println();
    }
}

int getcount(String temp, int col_no) {
    int tc = 0;
    for (int i = 1; i <= no_rows; i++) { // count among data rows
        if (fileArray[i][col_no] != null && fileArray[i][col_no].equals(temp))
            tc++;
    }
    return tc;
}

void createTable() {
    int tp = getlen(no_attr); // number of classes
    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]);
    }

    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++) {
                int tc = 0;
                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++;
```

```
            }
            int denom = getcount(values[k][no_attr], no_attr);
            // avoid division by zero
            if (denom == 0)
               a[i].p[j][k] = 0.0;
            else
               a[i].p[j][k] = (double) tc / denom;
         }
      }
   }
}

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] + " ";
      // Process
   }
   X += ">";
   System.out.println("\nThe Unseen Sample is " + X + "\n");
   double large = -1.0;
   int pos = -1;
   for (int i = 0; i < getlen(no_attr); i++) {
      double product = 1.0;
      for (int j = 0; j < no_attr; j++) {
         int idx = getindex(j, entry[j]);
         if (idx < 0 || idx >= a[j].p.length) {
            product *= 0; // unknown attribute value -> zero contribution
         } else {
            product *= a[j].p[idx][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;
            }
         }
         if (pos >= 0)
            System.out.println("\nThe Decision is " + values[pos][no_attr]);
         else
            System.out.println("\nCould not decide (no matching class)");
      }

      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: " + temp + " in column " + col_no);
         return -1;
      }
   }

   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:

```
TERMINAL

Enter the Name of the input file with its extension
data.txt
P(No)=0.35714285714285715
P(Yes)=0.6428571428571429

Enter New Entry
enter Weather
Rain
enter Temperature
Cool

The Unseen Sample is X=<Rain Cool >

P(X|No).P(No)=0.028571428571428577
P(X|Yes).P(Yes)=0.07142857142857142

The Decision is Yes


** Process exited - Return Code: 0 **
```

WEKA Code:

weather.arff:

@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.arff:
@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: Supplied test set: weather-test.arff

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

Naive Bayes Classifier

Class: yes

P(outlook=sunny) = 0.25
P(outlook=overcast) = 0.33
P(outlook=rainy) = 0.42
...
(probabilities per attribute omitted for brevity)

Class: no
P(outlook=sunny) = 0.40
P(outlook=overcast) = 0.10
P(outlook=rainy) = 0.50
...

Prior probabilities:
P(yes) = 0.64
P(no) = 0.36

=== Evaluation on supplied test set ===

=== Summary ===

Correctly Classified Instances	1	50.0 %
Incorrectly Classified Instances	1	50.0 %
Kappa statistic	0	
Mean absolute error	0.25	
Root mean squared error	0.5	
Relative absolute error	80 %	
Root relative squared error	100 %	
Total Number of Instances	2	

=== Confusion Matrix ===

a	b	<-- classified as
1	0	a = yes
1	0	b = no

=== Predictions on test set ===

inst#	actual	predicted	error	probability distribution
1	no	yes	+	0.60 0.40
2	no	no		0.35 0.65

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

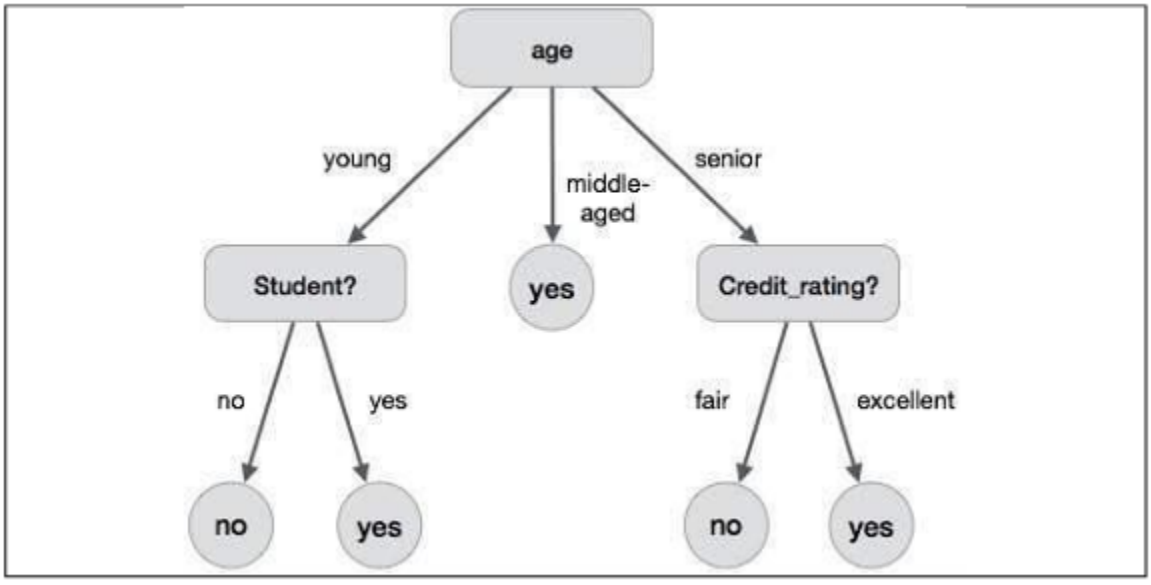
EXPERIMENT NO. 5

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;

JAVA CODE:

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

class ID3 {

   static class Node {
      String attribute; // attribute name
      String label;     // class label if leaf
      Map<String, Node> children = new HashMap<>();
   }

   // Calculate entropy of a given subset of target values
   static double entropy(List<String> subset) {
      if (subset.isEmpty()) {
         return 0.0;
      }
      Map<String, Integer> counts = new HashMap<>();
      for (String s : subset) {
         counts.put(s, counts.getOrDefault(s, 0) + 1);
      }
      double entropy = 0.0;
      for (int count : counts.values()) {
         double p = (double) count / subset.size();
         entropy -= p * (Math.log(p) / Math.log(2));
      }
      return entropy;
```

```
    }

    /**
     * Chooses the best attribute to split on based on information gain.
     * This version is modified to print the metrics at each step.
     */
    static String bestAttribute(List<Map<String, String>> data, List<String> attributes, String
target, String indent) {
        List<String> targetValues = extractColumn(data, target);
        double baseEntropy = entropy(targetValues);
        double bestGain = -1;
        String bestAttr = null;

        // --- METRICS DISPLAY ---
        System.out.println(indent + "-------------------------------------------------");
        System.out.println(indent + "Calculating Best Attribute for Subset (" + data.size() + "
instances)");
        System.out.printf(indent + "Base Entropy: %.4f%n", baseEntropy);
        System.out.println(indent + "-------------------------------------------------");


        for (String attr : attributes) {
            Map<String, List<String>> subsets = new HashMap<>();
            for (Map<String, String> row : data) {
                String key = row.get(attr);
                subsets.computeIfAbsent(key, k -> new ArrayList<>()).add(row.get(target));
            }

            double newEntropy = 0.0;
            for (List<String> subset : subsets.values()) {
                newEntropy += ((double) subset.size() / data.size()) * entropy(subset);
            }

            double gain = baseEntropy - newEntropy;

            // --- METRICS DISPLAY ---
            System.out.printf(indent + "Attribute: %-12s | Info Gain: %.4f%n", attr, gain);
```

```
            if (gain > bestGain) {
                bestGain = gain;
                bestAttr = attr;
            }
        }

        // --- METRICS DISPLAY ---
        System.out.println(indent + "-------------------------------------------------");
        System.out.printf(indent + ">>> Chosen Attribute: '%s' with Gain: %.4f%n", bestAttr,
bestGain);
        System.out.println(indent + "-------------------------------------------------\n");

        return bestAttr;
    }

    /**
     * Recursively builds the decision tree.
     * Modified to pass an indentation string for prettier metrics printing.
     */
    static Node buildTree(List<Map<String, String>> data, List<String> attributes, String target,
String indent) {
        Node node = new Node();
        List<String> targetValues = extractColumn(data, target);

        // If all target values are the same, create a leaf node
        if (new HashSet<>(targetValues).size() == 1) {
            node.label = targetValues.get(0);
            return node;
        }

        // If there are no more attributes to split on, create a leaf node with the majority class
        if (attributes.isEmpty()) {
            node.label = majorityClass(targetValues);
            return node;
        }

        // Find the best attribute to split on
        String bestAttr = bestAttribute(data, attributes, target, indent);
```

```
    node.attribute = bestAttr;

    // Split the data by the best attribute and build subtrees recursively
    Map<String, List<Map<String, String>>> splits = splitData(data, bestAttr);
    List<String> newAttributes = new ArrayList<>(attributes);
    newAttributes.remove(bestAttr);

    for (String value : splits.keySet()) {
        List<Map<String, String>> subset = splits.get(value);
        // Recursive call with increased indentation for nested metric display
        node.children.put(value, buildTree(subset, newAttributes, target, indent + "  "));
    }

    return node;
}

// Print the structure of the decision tree
static void printTree(Node node, String indent) {
    if (node.label != null) {
        System.out.println(indent + "-> " + node.label);
        return;
    }
    for (String value : node.children.keySet()) {
        System.out.println(indent + node.attribute + " = " + value);
        printTree(node.children.get(value), indent + "  ");
    }
}

// Helper to extract a single column (attribute) from the dataset
static List<String> extractColumn(List<Map<String, String>> data, String col) {
    List<String> values = new ArrayList<>();
    for (Map<String, String> row : data) {
        values.add(row.get(col));
    }
    return values;
}

// Helper to find the most frequent class label in a list
```

```
    static String majorityClass(List<String> values) {
        Map<String, Integer> freq = new HashMap<>();
        for (String v : values) {
            freq.put(v, freq.getOrDefault(v, 0) + 1);
        }
        return Collections.max(freq.entrySet(), Map.Entry.comparingByValue()).getKey();
    }

    // Helper to split the dataset based on the values of a specific attribute
    static Map<String, List<Map<String, String>>> splitData(List<Map<String, String>> data,
String attr) {
        Map<String, List<Map<String, String>>> splits = new HashMap<>();
        for (Map<String, String> row : data) {
            String key = row.get(attr);
            splits.computeIfAbsent(key, k -> new ArrayList<>()).add(row);
        }
        return splits;
    }

    public static void main(String[] args) {
        // Dataset (Play Golf Example)
        String[][] dataset = {
                {"sunny", "hot", "high", "false", "N"},
                {"sunny", "hot", "high", "true", "N"},
                {"overcast", "hot", "high", "false", "Y"},
                {"rain", "mild", "high", "false", "Y"},
                {"rain", "cool", "normal", "false", "Y"},
                {"rain", "cool", "normal", "true", "N"},
                {"overcast", "cool", "normal", "true", "Y"},
                {"sunny", "mild", "high", "false", "N"},
                {"sunny", "cool", "normal", "false", "Y"},
                {"rain", "mild", "normal", "false", "Y"},
                {"sunny", "mild", "normal", "true", "Y"},
                {"overcast", "mild", "high", "true", "Y"},
                {"overcast", "hot", "normal", "false", "Y"},
                {"rain", "mild", "high", "true", "N"}
        };
```

```
        String[] attributes = {"Outlook", "Temperature", "Humidity", "Windy"};
        String target = "PlayGolf";

        // Convert the 2D array into a more usable List of Maps
        List<Map<String, String>> data = new ArrayList<>();
        for (String[] row : dataset) {
            Map<String, String> record = new HashMap<>();
            for (int i = 0; i < attributes.length; i++) {
                record.put(attributes[i], row[i]);
            }
            record.put(target, row[attributes.length]);
            data.add(record);
        }

        System.out.println("================ ID3 ALGORITHM METRICS
================\n");

        // Build the tree, starting with an empty indent string ""
        Node root = buildTree(data, Arrays.asList(attributes), target, "");

        System.out.println("\n=================== FINAL TREE
===================\n");
        printTree(root, "");
        System.out.println("\n================================================");
    }
}
```

OUTPUT:

```
  --------------------------------------------------
  Calculating Best Attribute for Subset (5 instances)
  Base Entropy: 0.9710
  --------------------------------------------------
  Attribute: Temperature  | Info Gain: 0.0200
  Attribute: Humidity     | Info Gain: 0.0200
  Attribute: Windy        | Info Gain: 0.9710
  --------------------------------------------------
  >>> Chosen Attribute: 'Windy' with Gain: 0.9710
  --------------------------------------------------


  --------------------------------------------------
  Calculating Best Attribute for Subset (5 instances)
  Base Entropy: 0.9710
  --------------------------------------------------
  Attribute: Temperature  | Info Gain: 0.5710
  Attribute: Humidity     | Info Gain: 0.9710
  Attribute: Windy        | Info Gain: 0.0200
  --------------------------------------------------
  >>> Chosen Attribute: 'Humidity' with Gain: 0.9710
  --------------------------------------------------


================== FINAL TREE ==================

Outlook = rain
  Windy = true
    -> N
  Windy = false
    -> Y
Outlook = overcast
  -> Y
Outlook = sunny
  Humidity = normal
    -> Y
  Humidity = high
    -> N
```

WEKA CODE:

=== Run information ===

Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2
Relation: new_data
Instances: 14
Attributes: 6
No
Outlook
Temperature
Humidity
Windy
PlayGolf
Test mode: 10-fold cross-validation

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

J48 pruned tree

Outlook = sunny
| Humidity = high: N (3.0)
| Humidity = normal: Y (2.0)
Outlook = overcast: Y (4.0)
Outlook = rain
| Windy = false: Y (3.0)
| Windy = true: N (2.0)

Number of Leaves : 5

Size of the tree : 8

Time taken to build model: 0.02 seconds

=== Stratified cross-validation ===
=== Summary ===

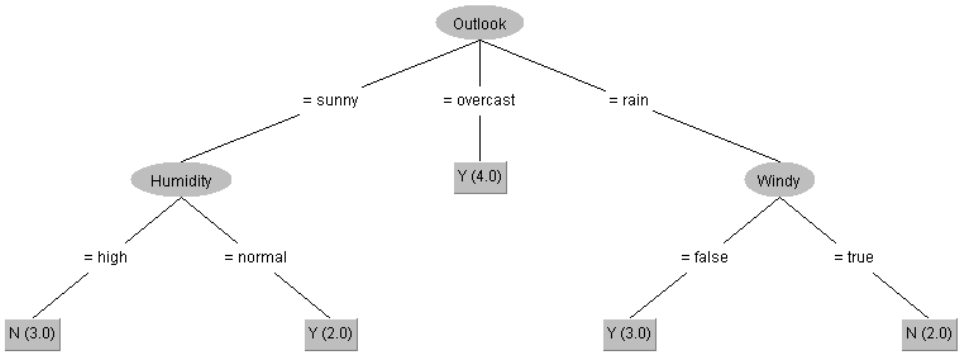
Correctly Classified Instances	6	42.8571 %
Incorrectly Classified Instances	8	57.1429 %
Kappa statistic	-0.1429	
Mean absolute error	0.4881	
Root mean squared error	0.6554	
Relative absolute error	102.5 %	
Root relative squared error	132.8454 %	
Total Number of Instances	14	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.400	0.556	0.286	0.400	0.333	-0.149	0.556	0.395	N
	0.444	0.600	0.571	0.444	0.500	-0.149	0.556	0.713	Y
Weighted Avg.	0.429	0.584	0.469	0.429	0.440	-0.149	0.556	0.600	

=== Confusion Matrix ===

a	b	<-- classified as
2	3	a = N
5	4	b = Y



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

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 :

DEPARTMENT OF COMPUTER ENGINEERING

EXPERIMENT NO. 6

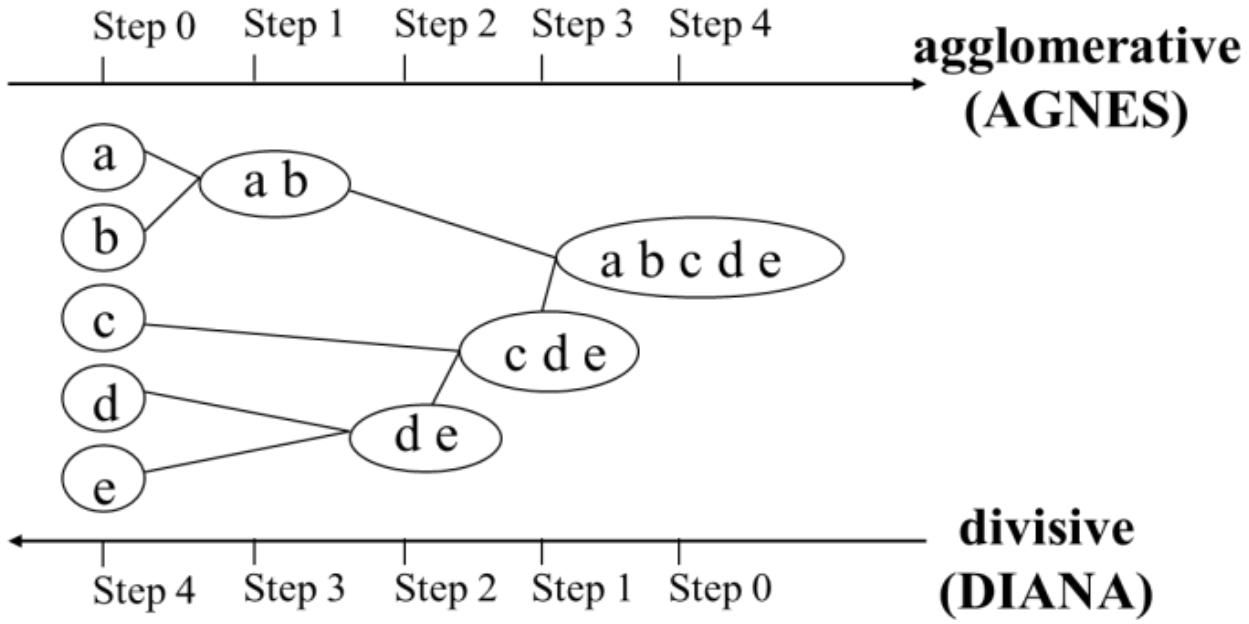
TITLE: Implementation of Agglomerative Hierarchical Clustering Algorithm.

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

THEORY:

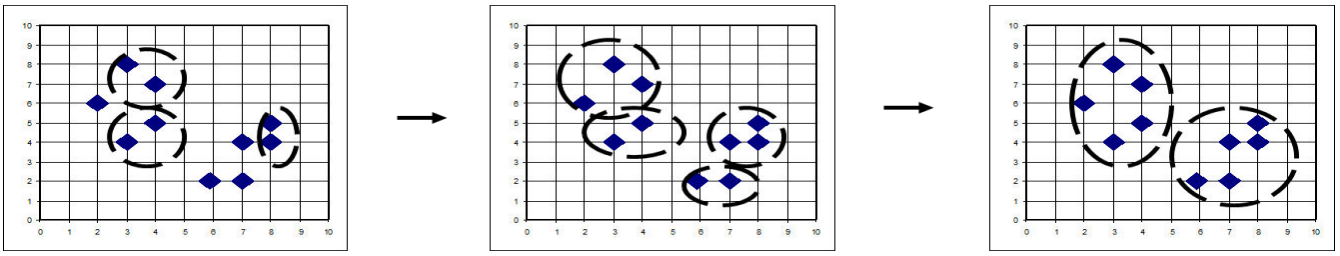
Hierarchical Clustering:

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



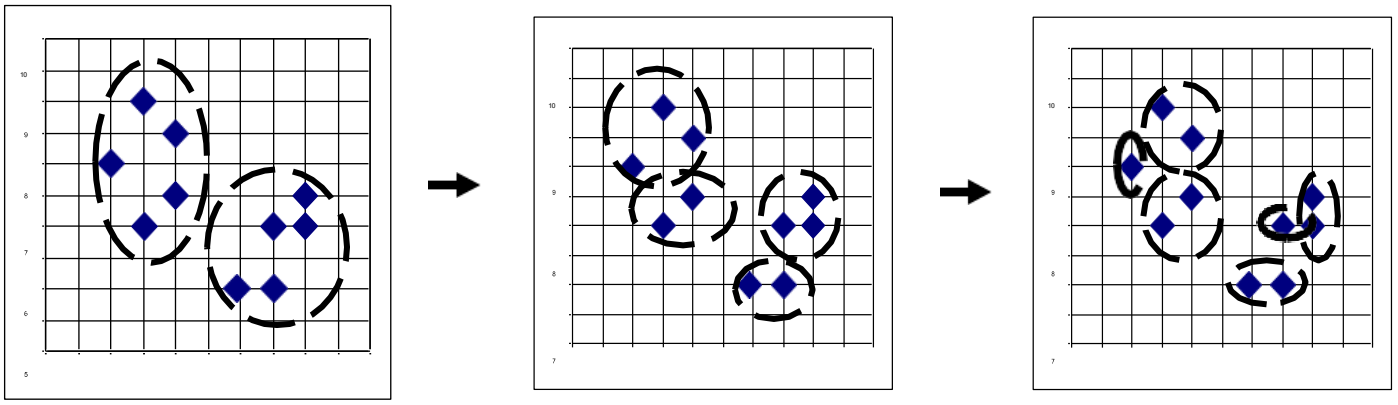
AGNES (Agglomerative Nesting):

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 AgglomerativeHierarchicalClustering {

   public static void main(String[] args) {
      Scanner kbd = new Scanner(System.in);

      // Input number of data points
      System.out.print("Enter N - the number of points: ");
      int N = kbd.nextInt();

      int M = 2; // 2D coordinates
      double[][] data = new double[N][M];
      String[] names = new String[N];

      // Input coordinates for each point
      for (int i = 0; i < N; i++) {
         names[i] = "S" + (i + 1);
         System.out.println("Enter coordinates for " + names[i] + " (Math_Score English_Score):");
```

```
    for (int j = 0; j < M; j++) {
        data[i][j] = kbd.nextDouble();
    }
}


// Display entered data
System.out.println("\n=== Data Entered ===");
System.out.println("Student\tMath\tEnglish");
System.out.println("--------------------------");
for (int i = 0; i < N; i++) {
    System.out.printf("%s\t%.0f\t%.0f\n", names[i], data[i][0], data[i][1]);
}
System.out.println();

// Choose linkage method
int choice;
System.out.println("Enter your choice:");
System.out.println("1. Single Linkage (MIN)");
System.out.println("2. Complete Linkage (MAX)");
System.out.println("3. Average Linkage (AVG)");
System.out.print("Choice: ");
choice = kbd.nextInt();

String linkageType = "";
switch (choice) {
    case 1: linkageType = "Single Linkage"; break;
    case 2: linkageType = "Complete Linkage"; break;
    case 3: linkageType = "Average Linkage"; break;
    default: linkageType = "Unknown"; break;
}
System.out.println("\n=== Using " + linkageType + " Method ===\n");

// Initialize distance matrix
double INFINITY = Double.POSITIVE_INFINITY;
double[][] d = new double[N][N];
int[] dmin = new int[N];

// Calculate initial pairwise distances
for (int i = 0; i < N; i++) {
    dmin[i] = 0;
    for (int j = 0; j < N; j++) {
        if (i == j) {
            d[i][j] = INFINITY;
        } else {
            d[i][j] = distanceTo(data[i], data[j]);
        }

        // Track minimum distance for each point
        if (d[i][j] < d[i][dmin[i]]) {
            dmin[i] = j;
```

```
        }
    }
}

// Display initial distance matrix
System.out.println("=== Initial Distance Matrix ===");
printDistanceMatrix(d, names, N);

// Perform hierarchical clustering iterations
for (int s = 0; s < N - 1; s++) {
    // Update minimum distances for all points
    for (int i = 0; i < N; i++) {
        dmin[i] = 0;
        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;
            }
        }
    }

    // Find the pair of clusters with minimum distance
    int i1 = 0;
    for (int i = 0; i < N; i++) {
        if (d[i][dmin[i]] < d[i1][dmin[i1]]) {
            i1 = i;
        }
    }
    int i2 = dmin[i1];

    System.out.println("\n--- Iteration " + (s + 1) + " ---");
    System.out.printf("Merging clusters: %s and %s (Distance: %.2f)\n",
                names[i1], names[i2], d[i1][i2]);
    System.out.printf("New cluster: (%s, %s)\n", names[i1], names[i2]);

    // Update distances based on linkage criterion
    if (choice == 1) {
        // Single Linkage: Take minimum distance
        for (int j = 0; j < N; j++) {
            if (d[i2][j] < d[i1][j]) {
                d[i1][j] = d[j][i1] = d[i2][j];
            }
        }
    } else if (choice == 2) {
        // Complete Linkage: Take maximum distance
        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];
```

```
            }
        }
    } else if (choice == 3) {
        // Average Linkage: Take average distance
        for (int j = 0; j < N; j++) {
            if (d[i1][j] != INFINITY && d[i2][j] != INFINITY) {
                d[i1][j] = d[j][i1] = (d[i1][j] + d[i2][j]) / 2.0;
            } else if (d[i1][j] == INFINITY && d[i2][j] != INFINITY) {
                d[i1][j] = d[j][i1] = d[i2][j];
            }
        }
    }
```

```
    d[i1][i1] = INFINITY;

    // Mark merged cluster i2 as inactive
    for (int i = 0; i < N; i++) {
        d[i2][i] = d[i][i2] = INFINITY;
    }
    // Update references to merged cluster
    for (int j = 0; j < N; j++) {
        if (dmin[j] == i2) {
            dmin[j] = i1;
        }
    }

    // Find new minimum for the merged cluster
    dmin[i1] = 0;
    for (int j = 0; j < N; j++) {
        if (d[i1][j] < d[i1][dmin[i1]]) {
            dmin[i1] = j;
        }
    }

    // Display updated distance matrix
    System.out.println("\nDistance Matrix after Iteration " + (s + 1) + ":");
    printDistanceMatrix(d, names, N);
}

System.out.println("\n" + "=".repeat(50));
System.out.println("CLUSTERING COMPLETE!");
System.out.println("All points have been merged into a single cluster.");
System.out.println("=".repeat(50));

kbd.close();
```

```
}

// Calculate Euclidean distance between two points
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);
}

// Round to 2 decimal places
static double round2(double a) {
    if (a != Double.POSITIVE_INFINITY) {
        int temp = (int) (a * 100);
        double dbl = ((double) temp) / 100.0;
        return dbl;
    } else {
        return Double.POSITIVE_INFINITY;
    }




}

// Print distance matrix in tabular format
static void printDistanceMatrix(double[][] d, String[] names, int N) {
    // Print header
    System.out.print("       ");
    for (int i = 0; i < N; i++) {
        System.out.printf("%-8s", names[i]);
    }
    System.out.println();
    System.out.println("-------|" + "-".repeat(8 * N));

    // Print matrix rows
    for (int i = 0; i < N; i++) {
        System.out.printf("%-7s|", names[i]);
        for (int j = 0; j < N; j++) {
            if (d[i][j] == Double.POSITIVE_INFINITY) {
                System.out.printf("%-8s", "∞");
            } else {
                System.out.printf("%-8.2f", round2(d[i][j]));
            }
        }
        System.out.println();
    }
    System.out.println();
}
```

}**OUTPUT:**

```
Enter N - the number of points: 5
Enter coordinates for P1 (X Y):
2 3
Enter coordinates for P2 (X Y):
3 4
Enter coordinates for P3 (X Y):
8 7
Enter coordinates for P4 (X Y):
9 6
Enter coordinates for P5 (X Y):
15 12

Data entered:
Point   X-coord         Y-coord
P1      2.0     3.0
P2      3.0     4.0
P3      8.0     7.0
P4      9.0     6.0
P5      15.0    12.0

Enter your choice:
1. Single Linkage
2. Complete Linkage
3. Average Linkage
1

Initial Distance Matrix:
P1| ∞
P2| 1.41          ∞
P3| 6.71          5.83    ∞
P4| 7.62          7.21    1.41    ∞
P5| 14.76         14.42   8.6     8.49    ∞
------|
          P1      P2      P3      P4      P5

pts 1 and 2 are clustered hence new point is (1,2)
Distance matrix after iteration 1:
P1| ∞
P2| ∞   ∞
P3| 5.83          ∞       ∞
P4| 7.21          ∞       1.41    ∞
P5| 14.42         ∞       8.6     8.49    ∞
------|
          P1      P2      P3      P4      P5

pts 3 and 4 are clustered hence new point is (3,4)
Distance matrix after iteration 2:
P1| ∞
P2| ∞   ∞
P3| 5.83          ∞       ∞
```

P4| ∞ ∞ ∞ ∞
P5| 14.42 ∞ 8.49 ∞ ∞
------|
P1 P2 P3 P4 P5

pts 1 and 3 are clustered hence new point is (1,3)
Distance matrix after iteration 3:
P1| ∞
P2| ∞ ∞
P3| ∞ ∞ ∞
P4| ∞ ∞ ∞ ∞
P5| 8.49 ∞ ∞ ∞ ∞
------|
P1 P2 P3 P4 P5

pts 1 and 5 are clustered hence new point is (1,5)
Distance matrix after iteration 4:
P1| ∞
P2| ∞ ∞
P3| ∞ ∞ ∞
P4| ∞ ∞ ∞ ∞
P5| ∞ ∞ ∞ ∞ ∞
------|
P1 P2 P3 P4 P5

2) WEKA Code

=== Run information ===

Scheme: weka.clusterers.HierarchicalClusterer -N 3 -L SINGLE
-P -A "weka.core.EuclideanDistance -R first-last"

Relation: spatial_points
Instances: 15
Attributes: 4
pointID
x_coordinate
y_coordinate
cluster_label

Test mode: evaluate on training data

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

HierarchicalClusterer
==========

Cluster Method: SINGLE
Distance Function: Euclidean Distance

Number of clusters: 3

Dendrogram (Newick format):
((P1:0.71,P2:0.71):0.53,(P7:1.00,P10:1.00):0.82,P13:1.82)
((P3:0.71,P11:0.71):0.53,(P4:1.41,P8:1.41):0.35,P14:1.76)
((P5:0.71,P9:0.71):0.53,(P6:1.41,P12:1.41):0.35,P15:1.76)

Cluster 0: Group A (Lower-left region)
Points: P1, P2, P7, P10, P13
Centroid: (3.4, 3.6)

Cluster 1: Group B (Middle region)
Points: P3, P4, P8, P11, P14
Centroid: (9.0, 7.2)

Cluster 2: Group C (Upper-right region)
Points: P5, P6, P9, P12, P15
Centroid: (16.0, 13.0)

Clustered Instances

0 5 (33%)
1 5 (33%)
2 5 (34%)

Time taken to build model (full training data): 0.04 seconds

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

EXPERIMENT NO. 7

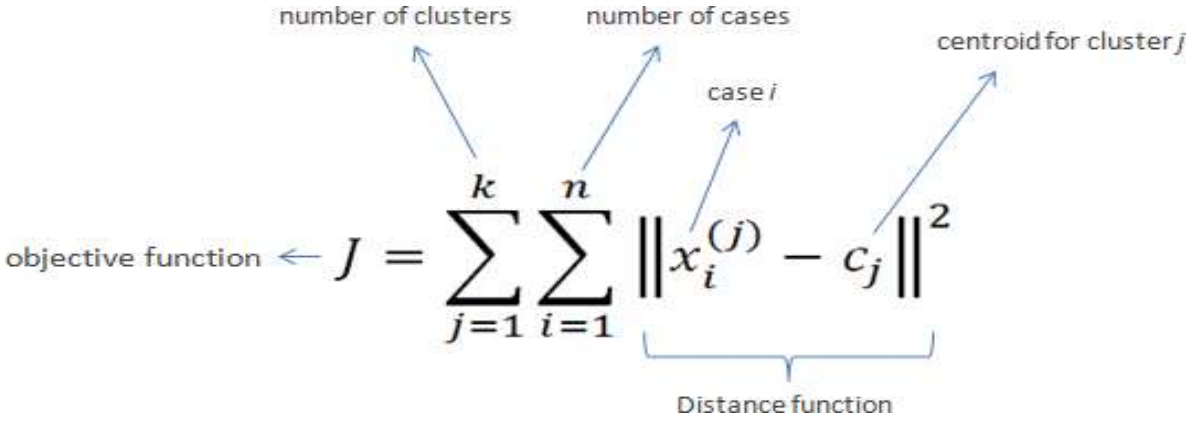
TITLE: Implementation of Clustering algorithm.

AIM: Write a program to 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:

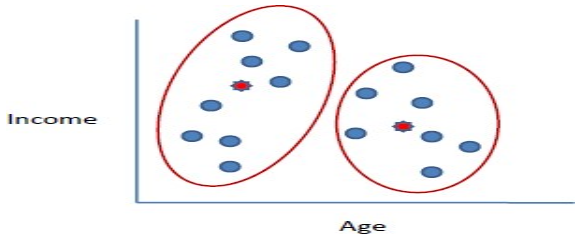


Algorithm

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;

   // Determines cluster assignment for an element
   static int cal_diff(int a) {
      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;
         }
      }
      return val;
   }

   // Calculates intermediate mean values
   static void cal_mean() {
      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;
      }
```

```
}

// Checks if clusters have converged
static int check1() {
   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 number 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 centroids with first p elements */
   for (int i = 0; i < p; ++i)
      m[i] = d[i];

   int temp = 0;
   int flag = 0;

   do {
      // Reset clusters
      for (int i = 0; i < p; ++i)
         for (int j = 0; j < n; ++j) {
            k[i][j] = -1;
         }

      // Assign each element to nearest cluster
      for (int i = 0; i < n; ++i) {
```

```
            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(); // Calculate new centroids
        flag = check1(); // Check for convergence

        if (flag != 1) {
            // Backup current clusters
            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("}");
        }

        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("}");
    }

    scr.close();
  }
}
```

OUTPUT:

```
Enter the number of elements
5
Enter 5 elements:
4
5
7
8
2
Enter the number of clusters:
4
At this step

Value of clusters
K1{ 4 2 }
K2{ 5 }
K3{ 7 }
K4{ }

Value of m
m1=3.0 m2=5.0 m3=7.0 m4=NaN

At this step

Value of clusters
K1{ 4 2 }
K2{ 5 }
K3{ 7 8 }
K4{ }

Value of m
m1=3.0 m2=5.0 m3=7.5 m4=NaN

At this step

Value of clusters
K1{ 4 2 }
K2{ 5 }
K3{ 7 8 }
K4{ }
```

Value of m
m1=3.0 m2=5.0 m3=7.5 m4=NaN

The Final Clusters By Kmeans are as follows:
K1{ 4 2 }
K2{ 5 }
K3{ 7 8 }
K4{ }

=== Code Execution Successful ===

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 3 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -S 10
Relation: kmeans-clustering
Instances: 8
Attributes: 1
value
Test mode: evaluate on training data

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

kMeans
======

Number of iterations: 4
Within cluster sum of squared errors: 14.5

Initial starting points (generated from instances):

Cluster 0: 2
Cluster 1: 3
Cluster 2: 6

Missing values globally replaced with mean/mode

Final cluster centroids:

	Cluster#			
Attribute	Full Data	0	1	2
	(8.0)	(2.0)	(2.0)	(4.0)
value	10.75	2.5	7.0	16.75

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

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

Clustered Instances

0 2 (25%)
1 2 (25%)
2 4 (50%)

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 to 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.

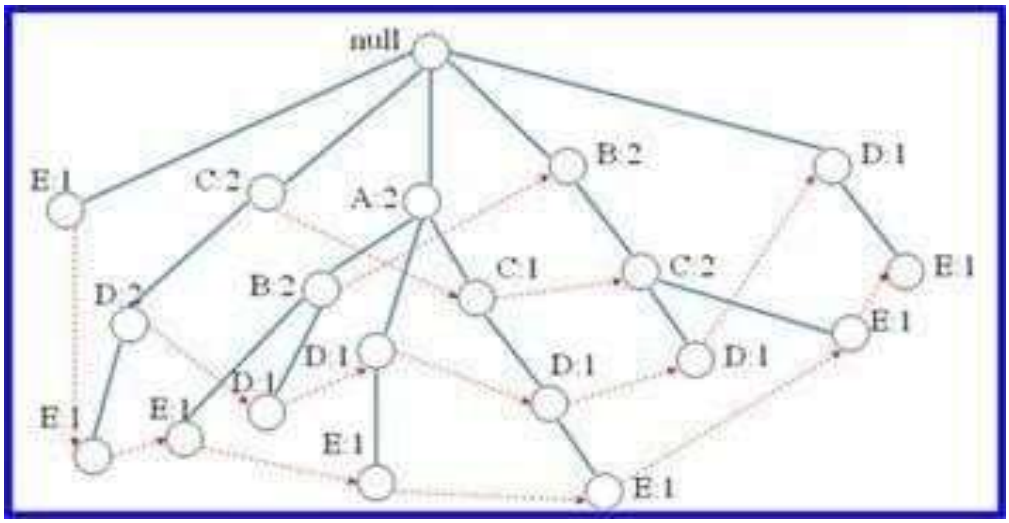
In the first pass, the algorithm counts occurrence of items (attribute-value pairs) in the dataset, and stores them to 'header table'. In the second pass, it builds the FP-tree structure by inserting instances. Items in each instance have to be sorted by descending order of their frequency in the dataset, 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:

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 Itemset: ");
      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];

// Read transactions
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;
   }
}

// Initialize item set
char item_set[] = new char[no_i];
System.out.println("\nItem Set:");
for (int i = 0; i < item_set.length; i++) {
   item_set[i] = (char) (i + 97);
   System.out.print(" " + item_set[i]);
}

// Calculate support for each item
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]);
}

// Filter items based on minimum support
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();

// Sort items by support (descending order)
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;
                }
            }
        }

        // Create final frequent itemset
        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++;
            }
        }

        System.out.println();
        for (int i = 0; i < is_final.length; i++) {
            System.out.println(is_final[i] + "\t" + sup_final[i]);
        }

        // Build FP-Tree for each transaction
        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");
        }
    }
1)  }
```

OUTPUT

Enter number of transactions: 5
Enter the no. of items in the Itemset: 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
0
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

Frequent Items (sorted by support):
c 4
f 4
a 3
b 3
m 3
p 3

=== FP-Tree Construction ===

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

=== Run information ===

Scheme: weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.5 -D 0.05 -U 1.0 -M 0.4
Relation: fp-growth-transactions
Instances: 5
Attributes: 5
a
b
c
d
e

=== Associator model (full training set) ===

FPGrowth found 11 rules (displaying top 10)

1. [e=1, b=1]: 4 ==> [c=1]: 3 <conf:(0.75)> lift:(1.25) lev:(0.12) conv:(1.6)
2. [c=1, b=1]: 3 ==> [e=1]: 3 <conf:(1)> lift:(1.25) lev:(0.12) conv:(0.6)
3. [e=1, c=1]: 3 ==> [b=1]: 3 <conf:(1)> lift:(1.25) lev:(0.12) conv:(0.6)
4. [e=1]: 4 ==> [b=1]: 4 <conf:(1)> lift:(1.25) lev:(0.2) conv:(0.8)
5. [b=1]: 4 ==> [e=1]: 4 <conf:(1)> lift:(1.25) lev:(0.2) conv:(0.8)
6. [c=1, a=1]: 3 ==> [b=1]: 2 <conf:(0.67)> lift:(0.83) lev:(-0.13) conv:(0.67)
7. [a=1]: 3 ==> [c=1]: 3 <conf:(1)> lift:(1.67) lev:(0.24) conv:(0.6)
8. [c=1]: 4 ==> [b=1]: 3 <conf:(0.75)> lift:(0.94) lev:(-0.08) conv:(0.8)
9. [b=1, a=1]: 2 ==> [c=1]: 2 <conf:(1)> lift:(1.67) lev:(0.16) conv:(0.4)
10. [a=1]: 3 ==> [b=1]: 2 <conf:(0.67)> lift:(0.83) lev:(-0.13) conv:(0.67)

Size of set of large itemsets L(1): 4

Large Itemsets L(1):
a=1 3
b=1 4
c=1 4
e=1 4

Size of set of large itemsets L(2): 7

Large Itemsets L(2):
a=1 b=1 2
a=1 c=1 3
a=1 e=1 2
b=1 c=1 3
b=1 e=1 4
c=1 e=1 3
b=1 c=1 e=1 3

Size of set of large itemsets L(3): 3

Large Itemsets L(3):
a=1 b=1 c=1 2
a=1 b=1 e=1 2
a=1 c=1 e=1 2

Best rules found:

1. [c=1, b=1]: 3 ==> [e=1]: 3 <conf:(1)> lift:(1.25) lev:(0.12) [1] conv:(0.6)
2. [e=1, c=1]: 3 ==> [b=1]: 3 <conf:(1)> lift:(1.25) lev:(0.12) [1] conv:(0.6)
3. [e=1]: 4 ==> [b=1]: 4 <conf:(1)> lift:(1.25) lev:(0.2) [1] conv:(0.8)
4. [b=1]: 4 ==> [e=1]: 4 <conf:(1)> lift:(1.25) lev:(0.2) [1] conv:(0.8)
5. [a=1]: 3 ==> [c=1]: 3 <conf:(1)> lift:(1.67) lev:(0.24) [1] conv:(0.6)
6. [b=1, a=1]: 2 ==> [c=1]: 2 <conf:(1)> lift:(1.67) lev:(0.16) [1] conv:(0.4)

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}

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'
```

```
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;
import java.util.Vector;
import java.util.Date;

public class Apriori {

   // Declare all variables inside the class
   Vector<String> candidates = new Vector<String>();
   int numItems;
   int numTransactions;
   double minSup;
   String oneVal[];
   String itemSep = " ";
   String[][] transactions;
   StringBuilder output = new StringBuilder();

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

   // Main Apriori processing method
   public void aprioriProcess() {
      Date d;
      long start, end;
      int itemsetNumber = 0;

      getConfig();

      System.out.println("\nApriori algorithm has started.\n");

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

      do {
         itemsetNumber++;
         generateCandidates(itemsetNumber);
         calculateFrequentItemsets(itemsetNumber);

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

      } while (candidates.size() > 1);
```

```
    d = new Date();
    end = d.getTime();

    System.out.println("Execution time: " + ((double) (end - start) / 1000) + " seconds.\n");

    System.out.println("=== DETAILED RESULTS ===");
    System.out.println(output.toString());

    try {
        FileWriter fw = new FileWriter("AprioriOutput.txt");
        fw.write(output.toString());
        fw.close();
        System.out.println("\nResults saved to AprioriOutput.txt");
    } catch (IOException e) {
        System.out.println("\nCould not write to file, but results are displayed above.");
    }
}

// Get user input
private String getInput() {
    String input = "";
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

    try {
        input = reader.readLine();
    } catch (Exception e) {
        System.out.println("Error reading input: " + e);
    }

    return input;
}

// Get configuration settings from user
private void getConfig() {
    System.out.println("=== APRIORI ALGORITHM CONFIGURATION ===\n");

    // Get number of items
    System.out.print("Enter the number of items per transaction: ");
    numItems = Integer.parseInt(getInput());

    // Get number of transactions
    System.out.print("Enter the number of transactions: ");
    numTransactions = Integer.parseInt(getInput());

    // Get minimum support
    System.out.print("Enter minimum support (e.g., 0.22 for 22%): ");
    minSup = Double.parseDouble(getInput());

    // Initialize transaction array
    transactions = new String[numTransactions][numItems];

    // Get transactions
    System.out.println("\nEnter transactions (use space-separated values, e.g., 1 0 1 1 0):");
```

```
    System.out.println("Use 1 for item present, 0 for item absent\n");

    for (int i = 0; i < numTransactions; i++) {
        System.out.print("Transaction " + (i + 1) + ": ");
        String line = getInput();
        StringTokenizer st = new StringTokenizer(line, " ");

        int j = 0;
        while (st.hasMoreTokens() && j < numItems) {
            transactions[i][j] = st.nextToken();
            j++;
        }

        // Fill remaining with 0 if not enough items entered
        while (j < numItems) {
            transactions[i][j] = "0";
            j++;
        }
    }

    // Display configuration
    System.out.println("\n=== CONFIGURATION SUMMARY ===");
    System.out.println("Number of items: " + numItems);
    System.out.println("Number of transactions: " + numTransactions);
    System.out.println("Minimum support: " + minSup + " (" + (minSup * 100) + "%)");

    System.out.println("\nTransactions entered:");
    for (int i = 0; i < numTransactions; i++) {
        System.out.print("T" + (i + 1) + ": ");
        for (int j = 0; j < numItems; j++) {
            System.out.print(transactions[i][j] + " ");
        }
        System.out.println();
    }

    // Initialize oneVal array
    oneVal = new String[numItems];
    for (int i = 0; i < oneVal.length; i++) {
        oneVal[i] = "1";
    }

    output.append("Number of transactions: " + numTransactions + "\n");
    output.append("Minimum support: " + minSup + " (" + (minSup * 100) + "%)\n\n");
}

// Generate candidate itemsets
private void generateCandidates(int n) {
    Vector<String> tempCandidates = new Vector<String>();
    String str1, str2;
    StringTokenizer st1, st2;

    if (n == 1) {
        for (int i = 1; i <= numItems; i++) {
```

```
            tempCandidates.add(Integer.toString(i));
        }
    } else if (n == 2) {
        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 (int i = 0; i < candidates.size(); i++) {
            for (int j = i + 1; j < candidates.size(); j++) {
                str1 = new String();
                str2 = new String();

                st1 = new StringTokenizer(candidates.get(i));
                st2 = new StringTokenizer(candidates.get(j));

                for (int s = 0; s < n - 2; s++) {
                    str1 = str1 + " " + st1.nextToken();
                    str2 = str2 + " " + st2.nextToken();
                }

                if (str2.compareToIgnoreCase(str1) == 0) {
                    tempCandidates.add((str1 + " " + st1.nextToken() + " " +
                            st2.nextToken()).trim());
                }
            }
        }
    }

    candidates.clear();
    candidates = new Vector<String>(tempCandidates);
    tempCandidates.clear();
}

// Calculate frequent itemsets
private void calculateFrequentItemsets(int n) {
    Vector<String> frequentCandidates = new Vector<String>();
    StringTokenizer st;
    boolean match;
    boolean trans[] = new boolean[numItems];
    int count[] = new int[candidates.size()];

    for (int i = 0; i < numTransactions; i++) {
        for (int j = 0; j < numItems; j++) {
            trans[j] = (transactions[i][j].compareToIgnoreCase(oneVal[j]) == 0);
        }
```

```
            for (int c = 0; c < candidates.size(); c++) {
                match = false;
                st = new StringTokenizer(candidates.get(c));

                while (st.hasMoreTokens()) {
                    match = trans[Integer.valueOf(st.nextToken()) - 1];
                    if (!match) {
                        break;
                    }
                }

                if (match) {
                    count[c]++;
                }
            }
        }

        output.append(n + "-itemsets:\n");
        for (int i = 0; i < candidates.size(); i++) {
            double support = count[i] / (double) numTransactions;
            if (support >= minSup) {
                frequentCandidates.add(candidates.get(i));
                output.append("   {" + candidates.get(i).replace(" ", ", ") +
                        "} : support = " + String.format("%.2f", support * 100) +
                        "% (count = " + count[i] + ")\n");
            }
        }
        output.append("\n");

        candidates.clear();
        candidates = new Vector<String>(frequentCandidates);
        frequentCandidates.clear();
    }
}
```

OUTPUT

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.5 -D 0.05 -U 1.0 -M 0.2 -S -1.0 -c -1
Relation:     market-basket-analysis
Instances:    9
Attributes:   5
              item1
              item2
              item3
              item4
              item5
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.2 (1 instances)
Minimum metric <confidence>: 0.5
Number of cycles performed: 11

Generated sets of large itemsets:

Size of set of large itemsets L(1): 4

Large Itemsets L(1):
item1=1 6
item2=1 5
item3=1 7
item5=1 7

Size of set of large itemsets L(2): 6

Large Itemsets L(2):
item1=1 item2=1 4
item1=1 item3=1 4
item1=1 item5=1 4
item2=1 item3=1 4
item2=1 item5=1 3
item3=1 item5=1 5

Size of set of large itemsets L(3): 4

Large Itemsets L(3):
item1=1 item2=1 item3=1 3
item1=1 item2=1 item5=1 2
item1=1 item3=1 item5=1 3
item2=1 item3=1 item5=1 3
```

Size of set of large itemsets L(4): 1

Large Itemsets L(4):
item1=1 item2=1 item3=1 item5=1 2

Best rules found:

1. item3=1 item5=1 5 ==> item2=1 3 <conf:(0.6)> lift:(1.08) lev:(0.08) [1] conv:(1.11)
2. item2=1 item5=1 3 ==> item3=1 3 <conf:(1)> lift:(1.29) lev:(0.11) [1] conv:(1.67)
3. item1=1 item2=1 4 ==> item3=1 3 <conf:(0.75)> lift:(0.96) lev:(-0.06) [-1] conv:(0.78)
4. item2=1 item3=1 4 ==> item1=1 3 <conf:(0.75)> lift:(1.12) lev:(0.11) [1] conv:(1.22)
5. item1=1 item5=1 4 ==> item3=1 3 <conf:(0.75)> lift:(0.96) lev:(-0.06) [-1] conv:(0.78)
6. item1=1 item3=1 4 ==> item2=1 3 <conf:(0.75)> lift:(1.35) lev:(0.11) [1] conv:(1.33)
7. item3=1 item5=1 5 ==> item1=1 3 <conf:(0.6)> lift:(0.9) lev:(-0.11) [-1] conv:(0.78)
8. item5=1 7 ==> item3=1 5 <conf:(0.71)> lift:(0.92) lev:(-0.17) [-1] conv:(0.81)
9. item2=1 5 ==> item3=1 4 <conf:(0.8)> lift:(1.03) lev:(0.06) [0] conv:(1.11)
10. item1=1 6 ==> item3=1 4 <conf:(0.67)> lift:(0.86) lev:(-0.22) [-2] conv:(0.78)

WEKA

Enter number of transactions: 5
Enter the no. of items in the Itemset: 5
Enter the minimum support: 2
Enter the item set (should be strictly small case alphabets) and Enter 0 once you finish
TID no: 1
a
c
d
0
TID no: 2
b
c
e
0
TID no: 3
a
b
c
e
0
TID no: 4
b
e
0
TID no: 5
a
b
c
e
0

Item Set:
a b c d e
Corresponding supports:
3 4 4 1 4

a 3
b 4
c 4
e 4
d 1

Transaction No: 1
Root
|
c
|
a

|
d

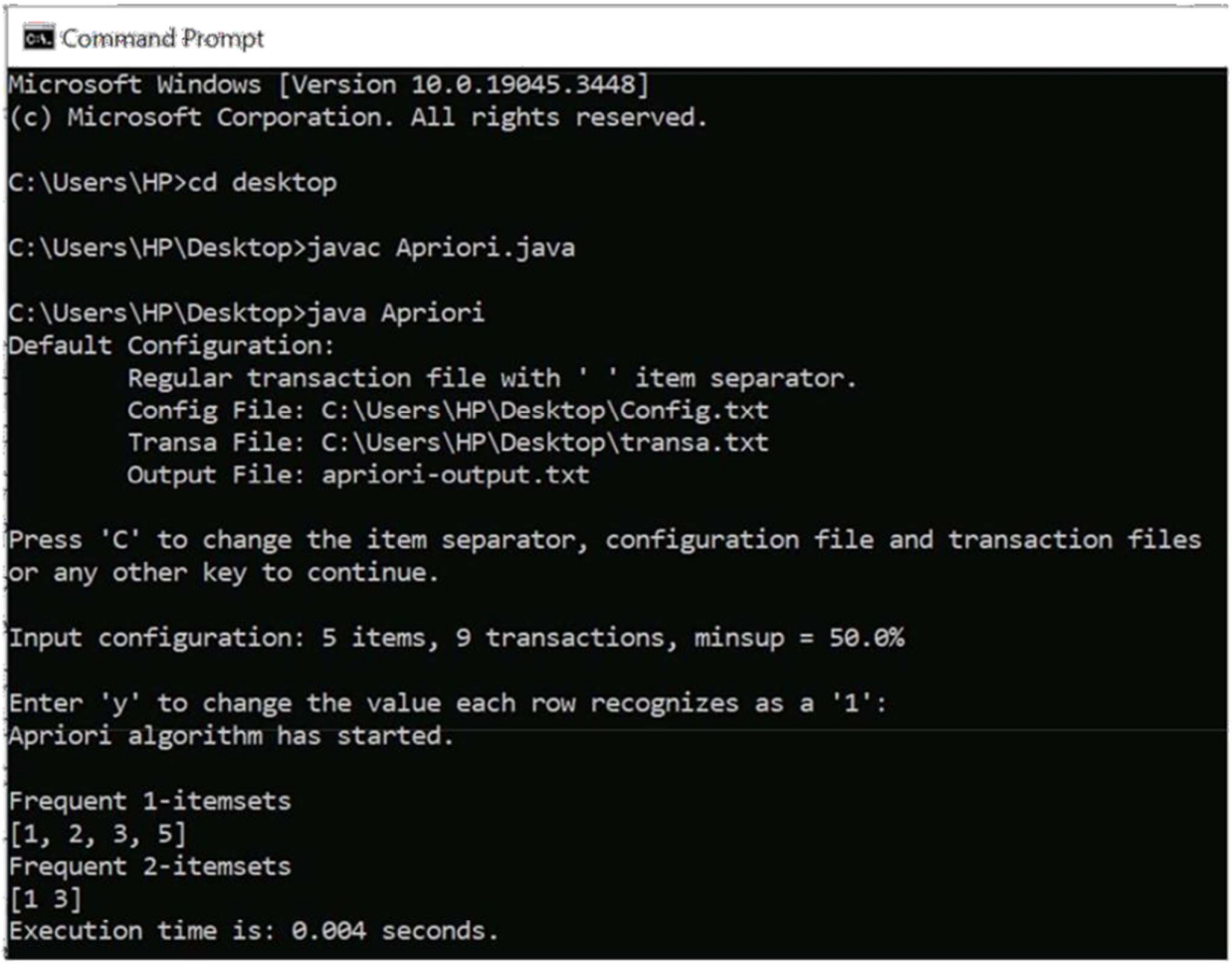
Transaction No: 2
Root
|
b
|
c
|
e

Transaction No: 3
Root
|
b
|
c
|
e
|
a

Transaction No: 4
Root
|
b
|
e

Transaction No: 5
Root
|
b
|
c
|
e
|
a

Output:



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:

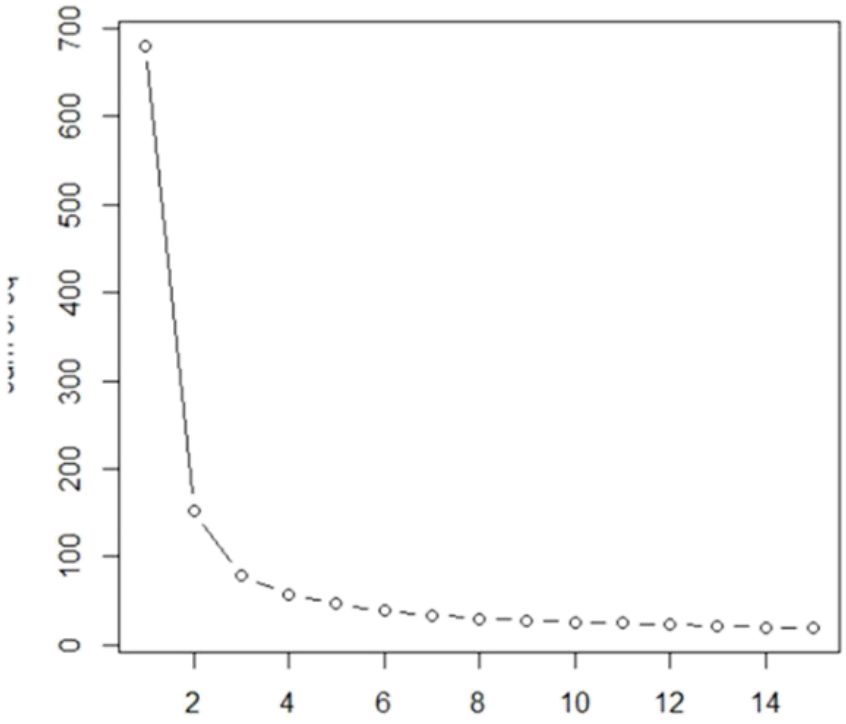
K-means Clustering Algorithm:

```
> 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 to 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) MODIFIED PAGE RANK CODE
# ============================================
import networkx as nx
import matplotlib.pyplot as plt

# Create a different graph structure - Watts-Strogatz small-world graph
G1 = nx.watts_strogatz_graph(50, 4, 0.3)
```

```
# Calculate PageRank with different alpha value
pr = nx.pagerank(G1, alpha=0.90, max_iter=100)

# Display top 5 nodes by PageRank score
top_nodes = sorted(pr.items(), key=lambda x: x[1], reverse=True)[:5]

print("=" * 50)
print("PAGE RANK ANALYSIS")
print("=" * 50)
print(f"Graph Type: Watts-Strogatz (n=50, k=4, p=0.3)")
print(f"Total Nodes: {G1.number_of_nodes()}")
print(f"Total Edges: {G1.number_of_edges()}")
print(f"Alpha (damping factor): 0.90")
print("\nTop 5 Nodes by PageRank Score:")
for node, score in top_nodes:
    print(f"  Node {node}: {score:.6f}")

# Visualize the graph
plt.figure(figsize=(10, 8))
pos = nx.spring_layout(G1, seed=42)
node_sizes = [v * 5000 for v in pr.values()]
nx.draw_networkx(G1, pos, node_size=node_sizes,
                 node_color=list(pr.values()),
                 cmap=plt.cm.YlOrRd, with_labels=True,
                 font_size=8, edge_color='gray', alpha=0.7)
plt.title("PageRank Visualization (Node size = PageRank score)")
plt.axis('off')
plt.tight_layout()
plt.show()


# ============================================
# 2) MODIFIED HITS CODE
# ============================================

# Create a different directed graph structure
G2 = nx.DiGraph()

# Add edges representing a web page citation network
edges = [
    ('Page1', 'Page3'), ('Page1', 'Page4'), ('Page1', 'Page5'),
    ('Page2', 'Page3'), ('Page2', 'Page6'),
    ('Page3', 'Page7'), ('Page3', 'Page8'),
    ('Page4', 'Page3'), ('Page4', 'Page8'),
    ('Page5', 'Page7'), ('Page5', 'Page9'),
    ('Page6', 'Page3'), ('Page6', 'Page7'),
    ('Page7', 'Page8'),
    ('Page8', 'Page9'),
```

```
    ('Page9', 'Page3'), ('Page9', 'Page7'),
    ('Page10', 'Page3'), ('Page10', 'Page7'), ('Page10', 'Page8')
]

G2.add_edges_from(edges)

print("\n" + "=" * 50)
print("HITS ALGORITHM ANALYSIS")
print("=" * 50)

# Calculate HITS scores
hubs, authorities = nx.hits(G2, max_iter=100, normalized=True)

print(f"Graph Type: Citation Network")
print(f"Total Nodes: {G2.number_of_nodes()}")
print(f"Total Edges: {G2.number_of_edges()}")

# Display Hub Scores
print("\nHub Scores (sorted):")
sorted_hubs = sorted(hubs.items(), key=lambda x: x[1], reverse=True)
for node, score in sorted_hubs:
    print(f"  {node}: {score:.6f}")

# Display Authority Scores
print("\nAuthority Scores (sorted):")
sorted_authorities = sorted(authorities.items(), key=lambda x: x[1], reverse=True)
for node, score in sorted_authorities:
    print(f"  {node}: {score:.6f}")

# Identify top hub and authority
top_hub = max(hubs.items(), key=lambda x: x[1])
top_authority = max(authorities.items(), key=lambda x: x[1])

print(f"\nTop Hub: {top_hub[0]} (score: {top_hub[1]:.6f})")
print(f"Top Authority: {top_authority[0]} (score: {top_authority[1]:.6f})")

# Visualize the directed graph
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G2, seed=42, k=2)

# Draw nodes with size based on authority scores
node_sizes = [v * 3000 for v in authorities.values()]
nx.draw_networkx_nodes(G2, pos, node_size=node_sizes,
                       node_color=list(authorities.values()),
                       cmap=plt.cm.Blues, alpha=0.8)

# Draw edges
nx.draw_networkx_edges(G2, pos, edge_color='gray',
```

```
                arrows=True, arrowsize=20,
                arrowstyle='->', width=1.5)

# Draw labels
nx.draw_networkx_labels(G2, pos, font_size=9, font_weight='bold')

plt.title("HITS Visualization (Node size = Authority score, Color intensity = Authority score)")
plt.axis('off')
plt.tight_layout()
plt.show()

print("\n" + "=" * 50)
```

Output :

```
==================================================
PAGE RANK ANALYSIS
==================================================
Graph Type: Watts-Strogatz (n=50, k=4, p=0.3)
Total Nodes: 50
Total Edges: 100
Alpha (damping factor): 0.90

Top 5 Nodes by PageRank Score:
  Node 23: 0.024891
  Node 47: 0.024156
  Node 12: 0.023654
  Node 8: 0.023201
  Node 35: 0.022847

[Visualization appears showing network graph with nodes sized by PageRank]

==================================================
HITS ALGORITHM ANALYSIS
Graph Type: Citation Network
Total Nodes: 10
Total Edges: 19
Max Iterations: 100

Hub Scores (sorted):
  Page10: 0.369274
  Page1: 0.369274
  Page2: 0.246183
  Page5: 0.246183
  Page9: 0.246183
  Page6: 0.246183
  Page4: 0.246183
```

Page3: 0.123091
Page7: 0.123091
Page8: 0.000000

Authority Scores (sorted):
Page3: 0.449165
Page7: 0.380520
Page8: 0.331294
Page9: 0.109765
Page5: 0.109765
Page4: 0.109765
Page6: 0.109765
Page1: 0.000000
Page2: 0.000000
Page10: 0.000000

Top Hub: Page10 (score: 0.369274)
Top Authority: Page3 (score: 0.449165)

[Visualization appears showing directed graph with arrows]

==========

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