**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S no.** | **Name of the experiment** | **Page no.** | **Date** | **Signature** |
| **1** | Multidimensional Data Models | **2** |  |  |
| **2** | Data Warehousing | **14** |  |  |
| **3** | OLAP (Online Analytical Processing Cube) | **15** |  |  |
| **4** | Basics of WEKA tool  a. Investigate the Application interfaces.  b. Explore the default datasets | **20** |  |  |
| **5** | Creating new ARFF file | **26** |  |  |
| **6** | Data Preprocessing  a. Attribute Selection  b. Handling Missing Values  c. Discretization  d. Eliminating Outliers | **28** |  |  |
| **7** | Generate Association Rules using the Apriori algorithm. | **37** |  |  |
| **8** | Generate Association Rules using the FP-Growth algorithm. | **46** |  |  |
| **9** | Decision Tree by using J48 algorithm. | **52** |  |  |
| **10** | Classify the given data item using Naïve Bayes theorem using the given data set. | **60** |  |  |
| **11** | Apply K-means clustering | **65** |  |  |
| **12** | R Program for Regression | **73** |  |  |

**1. Multidimensional Data Models**

**a. Star Schema**

**Aim:**

Understand and write a program to implement Star Schema

**Theory:**

The star schema (sometimes referenced as star join schema) is the simplest data warehouse schema, consisting of a single "*fact table*" with a compound primary key, with one segment for each "*dimension*" and with additional columns of additive, numeric facts. The star schema makes multi-dimensional database (MDDB) functionality possible using a traditional relational database. Because relational databases are the most common data management system in organizations today, implementing multi-dimensional views of data using a relational database is very appealing. Even if you are using a specific MDDB solution, its sources likely are relational databases. Another reason for using star schema is its ease of understanding. Fact tables in star schema are mostly in third normal form (3NF), but dimensional tables are in de-normalized second normal form (2NF). If we want to normalize dimensional tables, they look like snowflakes (see snowflake schema) and the same problems of relational databases arise - you need complex queries and business users cannot easily understand the meaning of data. Although query performance may be improved by advanced DBMS technology and hardware, highly normalized tables make reporting difficult and applications complex

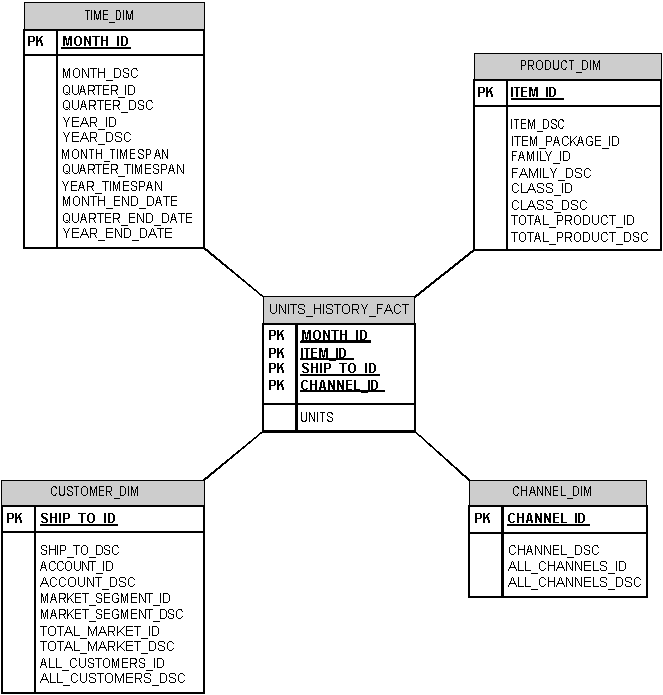


Figure 1: Star Schema

**Program:**

//Star Schema

#include<stdio.h>

#include<conio.h>

struct location

{

int l\_key;

char street[30];

char city[30];

char p\_st[100];

char country[30];

}l[10];

void main()

{

int n,flag=0,i,j;

clrscr();

printf("\*\*\*\*\*location table\*\*\*\*\*");

printf("\n enter the no of entries\n");

scanf("%d",&n);

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

{

printf("enter the values for l[%d]\n",i+1);

printf("enter the location key");

scanf("%d",&l[i].l\_key);

printf("Enter the street:");

scanf("%s",l[i].street);

printf("enter the city");

scanf("%s",l[i].city);

fflush(stdin);

printf("Enter the country");

scanf("%s",l[i].country);

}

printf("\n Entered Tabels are \n");

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

{

printf("%4d%4s%4s%4s %4s\n", l[i].l\_key,l[i].street,l[i].city,l[i].p\_st,l[i].country);

}

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

{

for(j=i+1;j<n;j++)

{

if(strcmp(l[i].country,l[j].country)==0)

{

if(strcmp(l[i].p\_st,l[j].p\_st)==0)

{

flag=1;

printf("\n Redundancy occurs for entry %d and %d",i,j);

}

}

}

}

if(flag==0)

printf("Redundancy not occurs");

getch();

}

**Output:**

\*\*\*\*\*location table\*\*\*\*\*

enter the no of entries:

2

enter the values for l[1]

enter the location key: 1

Enter the street: Mehdipatnam

enter the city: Hyderabad

Enter the country: India

enter the values for l[2]

enter the location key: 2

Enter the street: Somajiguda

enter the city: Hyderabad

Enter the country: India

Entered Tables are

1 Mehdipatnam Hyderabad India

2 Somajiguda Hyderabad India

Redundancy occurs for entry 1 and 2

**b. Snow Flake Schema**

**Aim:**

Understand and write a program to implement Snow Flake

**Theory:**

A snowflake schema is a logical arrangement of tables in a relational database such that the entity relationship diagram resembles a snowflake in shape. Closely related to the star schema, the snowflake schema is represented by centralized fact tables which are connected to multiple dimensions. In the snowflake schema, however, dimensions are normalized into multiple related tables whereas the star schema's dimensions are de-normalized with each dimension being represented by a single table. When the dimensions of a snowflake schema are elaborate, having multiple levels of relationships, and where child tables have multiple parent tables ("forks in the road"), a complex snowflake shape starts to emerge. The "snowflaking" effect only affects the dimension tables and not the fact tables.

The star and snowflake schema are most commonly found in dimensional data warehouses and data marts where speed of data retrieval is more important than the efficiency of data manipulations. As such, the tables in these schemas are not normalized much, and are frequently designed at a level of normalization short of third normal form.

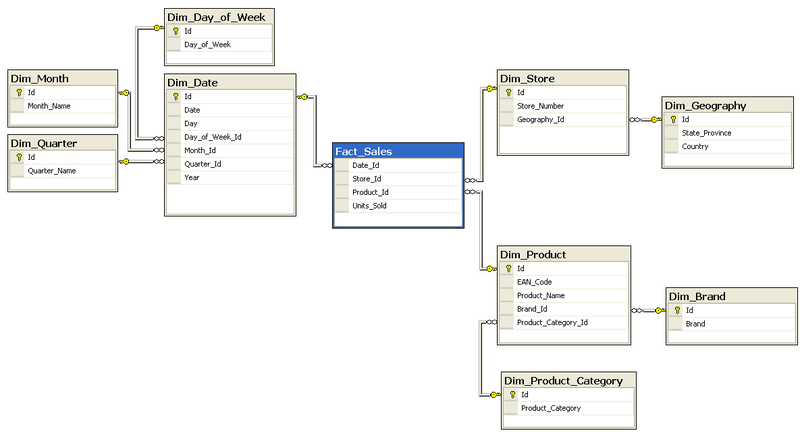


Figure 2: Snow Flake Schema

**Program:**

//Snow Flake schema

#include<stdio.h>

#include<conio.h>

struct location

{

int l\_key;

char street[10];

int city\_key;

struct city

{

int city\_key;

char city\_name[10];

char p\_state[10];

char country[10];

}c;

}l[10];

void main()

{

int n,i,j;

printf("Enter the number of entries:");

scanf("%d",&n);

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

{

printf("Enter the values fr location tuple[%d]",i+1);

printf("\nL\_Key:");

scanf("%d",&l[i].l\_key);

fflush(stdin);

printf("\nSTREET:");

gets(l[i].street);

fflush(stdin);

printf("\nCITY\_KEY:");

scanf("%d",&l[i].city\_key);

printf("Enter the Values for City Tuples[%d]",i+1);

printf("\nCITY\_KEY:");

scanf("%d",&l[i].c.city\_key);

fflush(stdin);

printf("\nCITY:");

gets(l[i].c.city\_name);

fflush(stdin);

printf("\n P\_STATE:");

gets(l[i].c.p\_state);

fflush(stdin);

printf("\n COUNTRY:");

gets(l[i].c.country);

}

printf("Entered location tabel values are:\n");

printf("L\_KEY\tSTREET\tCITY\_KEY\n");

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

{

printf("%d\t%s\t%d\n",l[i].l\_key,l[i].street,l[i].city\_key);

}

printf("Entered City tabel Values are:\n");

printf("C\_KEY\tC\_NAME\tP\_STATE\tCOUNTRY\n");

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

{

printf("%d\t%s\t%s\t%s\n",l[i].c.city\_key,l[i].c.city\_name,l[i].c.p\_state,l[i].c.country);

}

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

{

for(j=i+1;j<n;j++)

{

if(strcmp(l[i].c.country,l[j].c.country)==0)

if(strcmp(l[i].c.p\_state,l[j].c.p\_state)==0)

printf("\n Redundancy for this data\n");

else

printf("\n No redundancy for this data\n");

}

}

getch();

}

**Output:**

Enter the number of entries: 2

Enter the values for location tuple[1]

L\_KEY: 101

STREET: Somajiguda

CITY\_KEY: 1001

Enter the values for city tuple[1]

CITY\_KEY: 1001

CITY: Hyderabad

P\_STATE: AP

COUNTRY: India

Enter the values for location tuple[2]

L\_KEY: 102

STREET: Singapore

CITY\_KEY: 1002

Enter the values for city tuple[2]

CITY\_KEY: 1002

CITY: Singapore

P\_STATE: Singapore

COUNTRY: SGP

Entered Location Table values are:

L\_KEY STREET CITY\_KEY

102 Somajiguda 1001

102 Singapore 1002

Entered City Table values are:

C\_KEY C\_NAME P\_STATE COUNTRY

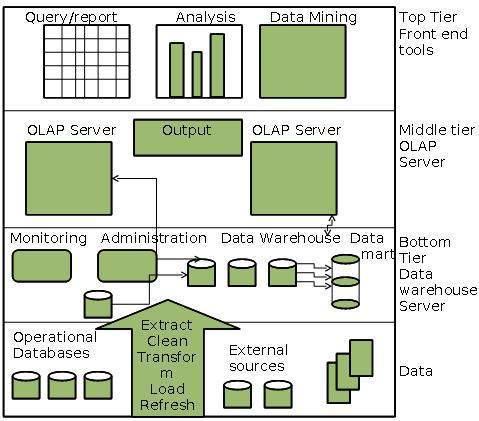
1001 Somajiguda AP India

1002 Singapore Singapore SGP

No Redundancy for this data

**2. Data Warehousing**

## Three-Tier Data Warehouse Architecture



* **Bottom Tier** - The bottom tier of the architecture is the data warehouse database server. It is the relational database system. We use the back-end tools and utilities to feed data into the bottom tier. These back-end tools and utilities perform the Extract, Clean, Load, and refresh functions.
* **Middle Tier** - In the middle tier, we have the OLAP Server that can be implemented in either of the following ways.
  + By Relational OLAP (ROLAP), which is an extended relational database management system. The ROLAP maps the operations on multidimensional data to standard relational operations.
  + By Multidimensional OLAP (MOLAP) model, which directly implements the multidimensional data and operations.
* **Top-Tier** - This tier is the front-end client layer. This layer holds the query tools and reporting tools, analysis tools and data mining tools.

**3. OLAP Cube**

An **OLAP cube** is a term that typically refers to multi-dimensional [array](https://en.wikipedia.org/wiki/Array_data_structure) of data. *OLAP* is an acronym for [online analytical processing](https://en.wikipedia.org/wiki/Online_analytical_processing),[[1]](https://en.wikipedia.org/wiki/OLAP_cube" \l "cite_note-1)which is a computer-based technique of analyzing data to look for insights. The term *cube* here refers to a multi-dimensional dataset, which is also sometimes called a [hypercube](https://en.wikipedia.org/wiki/Hypercube) if the number of dimensions is greater than 3.

**Operations:**

*1.* ***Slice*** is the act of picking a rectangular subset of a cube by choosing a single value for one of its dimensions, creating a new cube with one fewer dimension.[[4]](https://en.wikipedia.org/wiki/OLAP_cube" \l "cite_note-OLAPGlossary1995-4) The picture shows a slicing operation: The sales figures of all sales regions and all product categories of the company in the year 2005 and 2006 are "sliced" out of the data cube.

2.***Dice*:** The dice operation produces a subcube by allowing the analyst to pick specific values of multiple dimensions.[[5]](https://en.wikipedia.org/wiki/OLAP_cube" \l "cite_note-5)The picture shows a dicing operation: The new cube shows the sales figures of a limited number of product categories, the time and region dimensions cover the same range as before.

3.***Drill Down/Up***allows the user to navigate among levels of data ranging from the most summarized (up) to the most detailed (down).[[4]](https://en.wikipedia.org/wiki/OLAP_cube" \l "cite_note-OLAPGlossary1995-4) The picture shows a drill-down operation: The analyst moves from the summary category "Outdoor-Schutzausrüstung" to see the sales figures for the individual products.

4**. *Roll-up*:** A roll-up involves summarizing the data along a dimension. The summarization rule might be computing totals along a hierarchy or applying a set of formulas such as

profit = sales – expenses.

5***. Pivot:***allows an analyst to rotate the cube in space to see its various faces. For example, cities could be arranged vertically and products horizontally while viewing data for a particular quarter. Pivoting could replace products with time periods to see data across time for a single product.

OLAP Server Architectures:

* + Relational OLAP (ROLAP) servers: These are the intermediate servers that stand in between a relational back-end server and client front-end tools. They use relational or extended relational DBMS to store and manage warehouse data.
  + Multidimensional OLAP (MOLAP) servers: These support multidimensional data views through array based search engines. They map multidimensional views directly to data cube array structures.
  + Hybrid OLAP (HOLAP) servers: This approach combines ROLAP and MOLAP technology, benefiting from the greater scalability of ROLAP and faster computation of MOLAP.

The database that needs to analyzed using OLAP cube writer is displayed in Figure 1

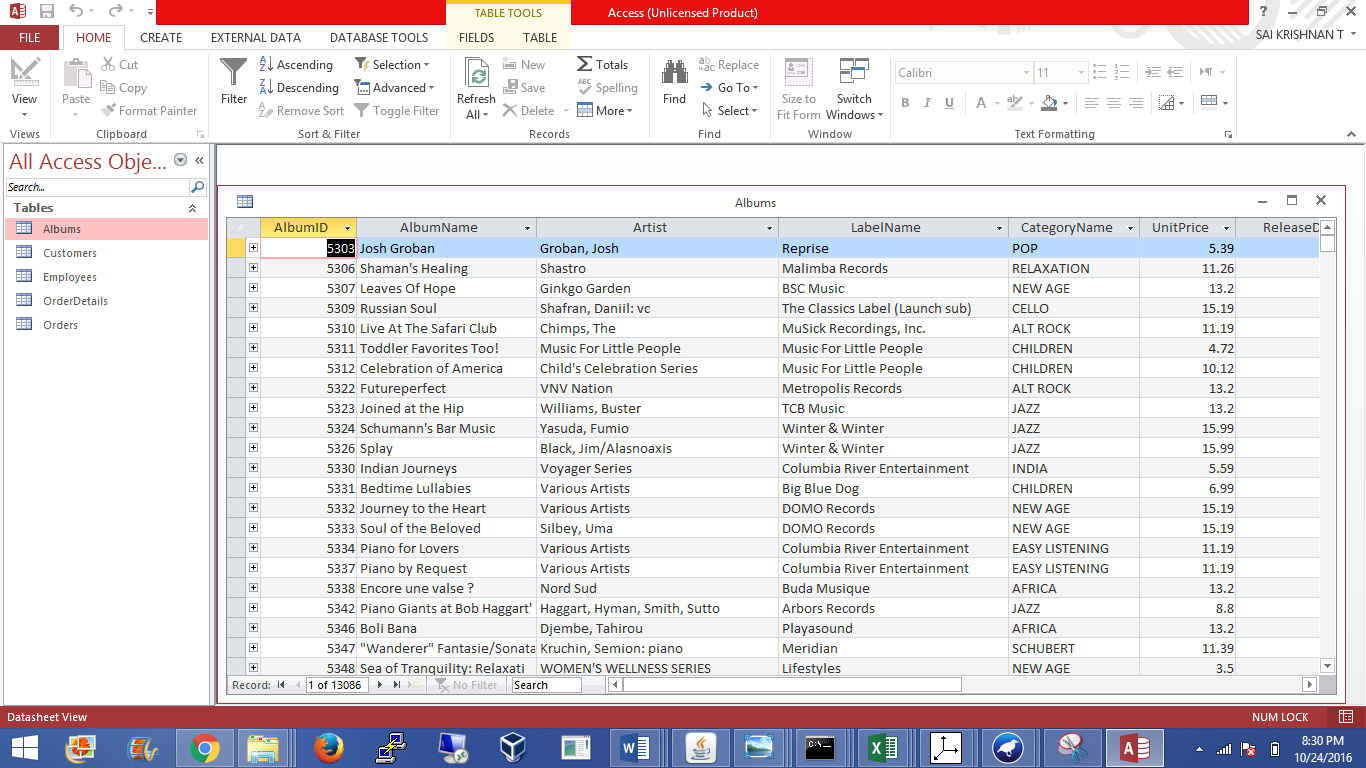


Figure 1 : Music Database

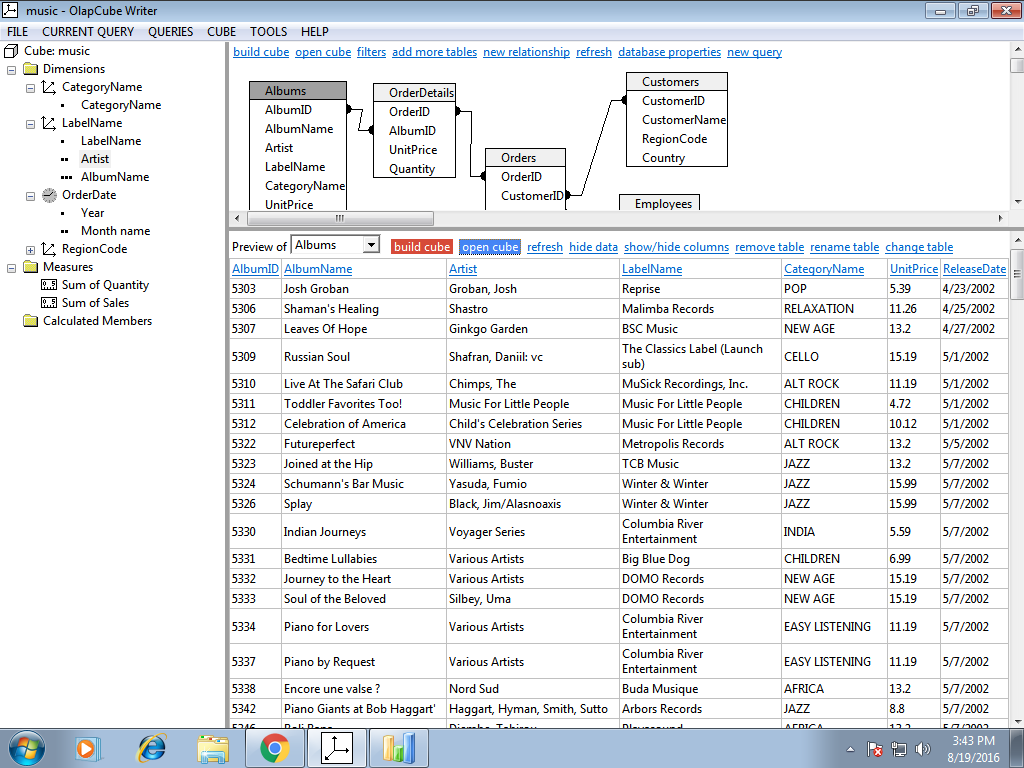


Figure 2 Database in OLAP cube writer

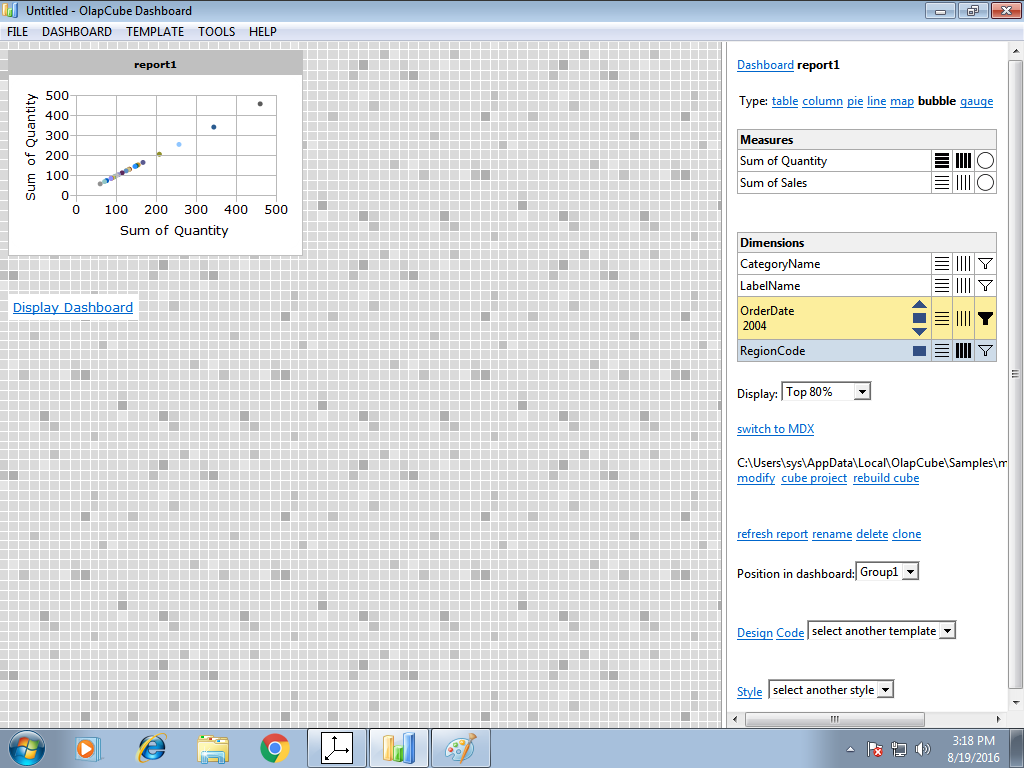


Figure 3: Graph between sum of quantities

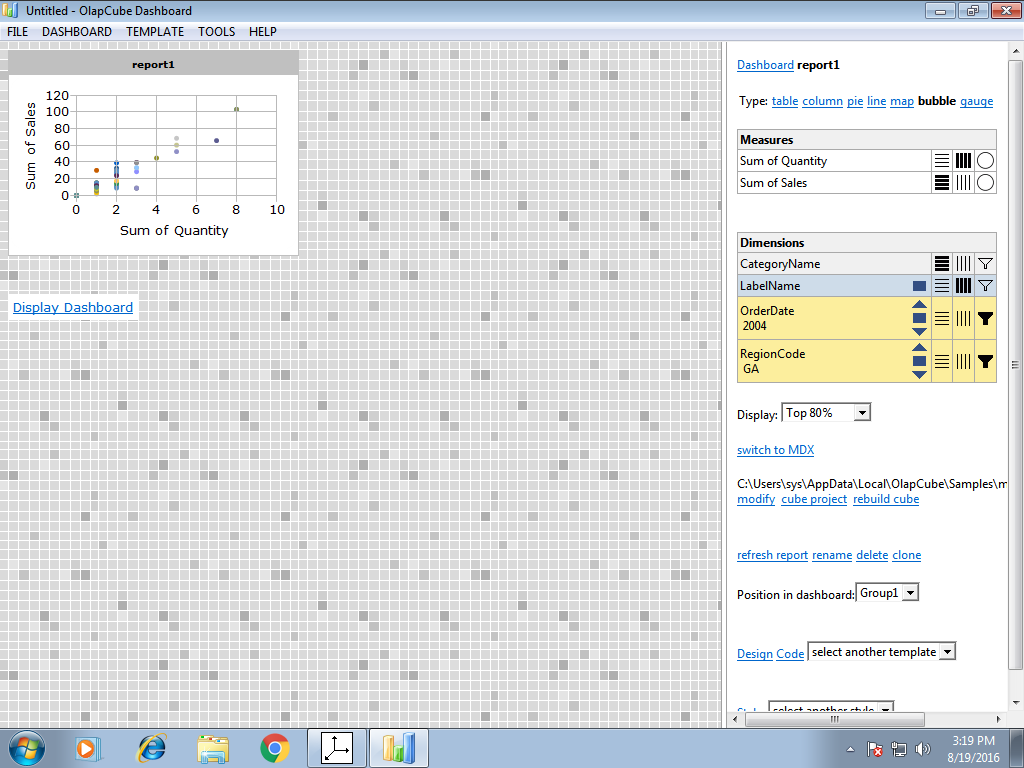


Figure 4: Graph between Sum of sales and quantities

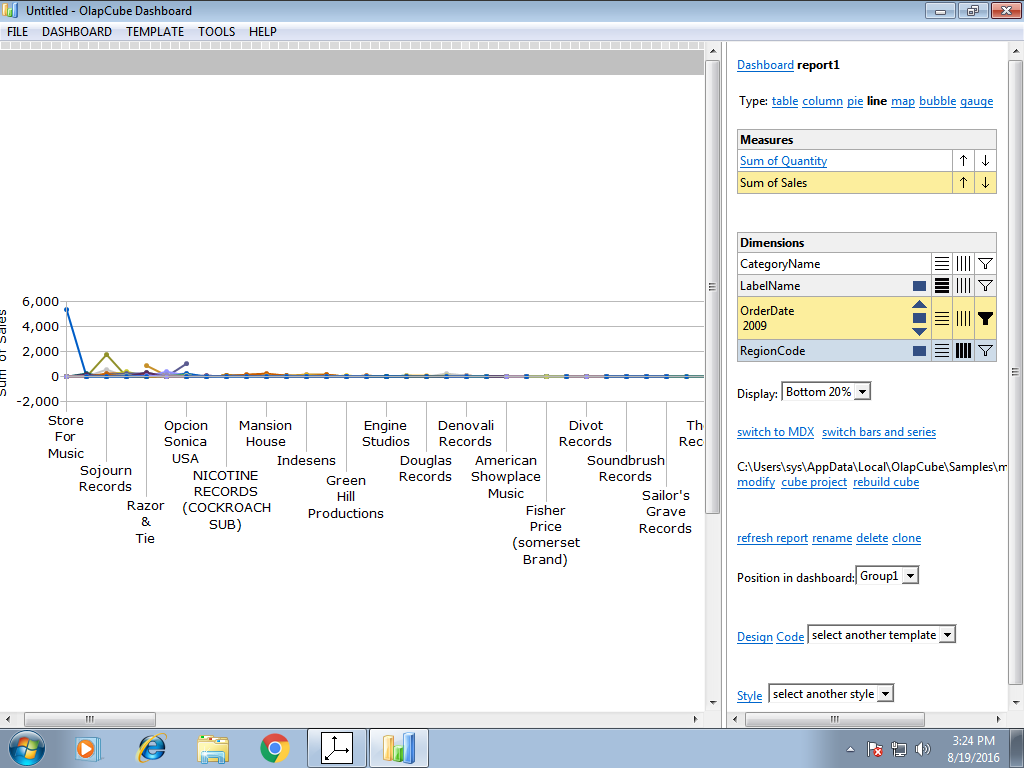


Figure 5: Sales from different stores

**4. Basics of WEKA tool**

**Aim**: A. Investigation the Application interfaces of the Weka tool.

**Introduction**:Weka (pronounced to rhyme with Mecca) is a workbench that contains a collection of visualization tools and algorithms for [data analysis](https://en.wikipedia.org/wiki/Data_analysis) and [predictive modeling](https://en.wikipedia.org/wiki/Predictive_modeling), together with graphical user interfaces for easy access to these functions. The original non-Java version of Weka was a [Tcl](https://en.wikipedia.org/wiki/Tcl)/Tk front-end to (mostly third-party) modeling algorithms implemented in other programming languages, plus data [preprocessing](https://en.wikipedia.org/wiki/Preprocessing) utilities in [C](https://en.wikipedia.org/wiki/C_(programming_language)), and [Make file](https://en.wikipedia.org/wiki/Makefile)-based system for running machine learning experiments. This original version was primarily designed as a tool for analyzing data from agricultural domains, but the more recent fully [Java](https://en.wikipedia.org/wiki/Java_(programming_language))-based version (Weka 3), for which development started in 1997, is now used in many different application areas, in particular for educational purposes and research. Advantages of Weka include:

* Free availability under the [GNU General Public License](https://en.wikipedia.org/wiki/GNU_General_Public_License).
* Portability, since it is fully implemented in the [Java programming language](https://en.wikipedia.org/wiki/Java_programming_language) and thus runs on almost any modern computing platform.
* A comprehensive collection of data [preprocessing](https://en.wikipedia.org/wiki/Preprocessing) and modeling techniques.
* Ease of use due to its graphical user interfaces.

**Description:**

Open the program. Once the program has been loaded on the user’s machine it is opened by navigating to the programs start option and that will depend on the user’s operating system. Figure 1.1 is an example of the initial opening screen on a computer.

There are four options available on this initial screen:

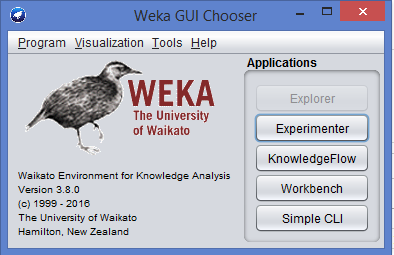


Fig: 4.1 Weka GUI

1. **Explorer** - the graphical interface used to conduct experimentation on raw data

After clicking the Explorer button the weka explorer interface appears.

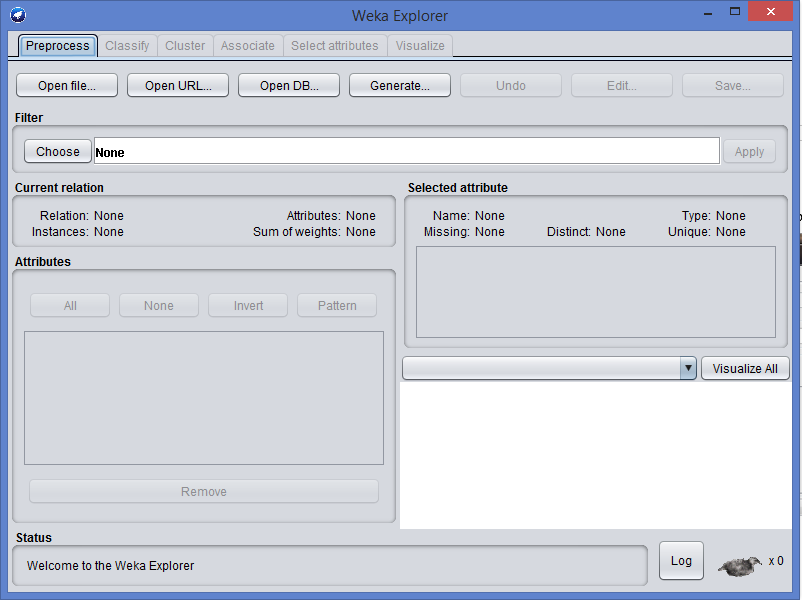


Fig: 4.2 Pre-processor

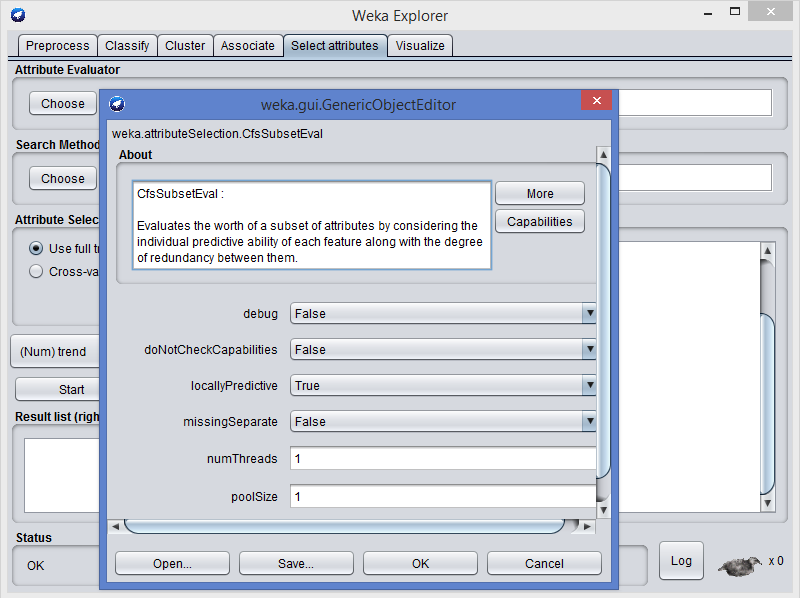


Fig: 4.3 Parameters

Inside the weka explorer window there are six tabs:

1. **Preprocess-** used to choose the data file to be used by the application.

♦**Open File**- allows for the user to select files residing on the local machine or recorded medium

♦**Open URL**- provides a mechanism to locate a file or data source from a different location specified by the user

♦**Open Database**- allows the user to retrieve files or data from a database source provided by user

1. **Classify-** used to test and train different learning schemes on the preprocessed data file under experimentation

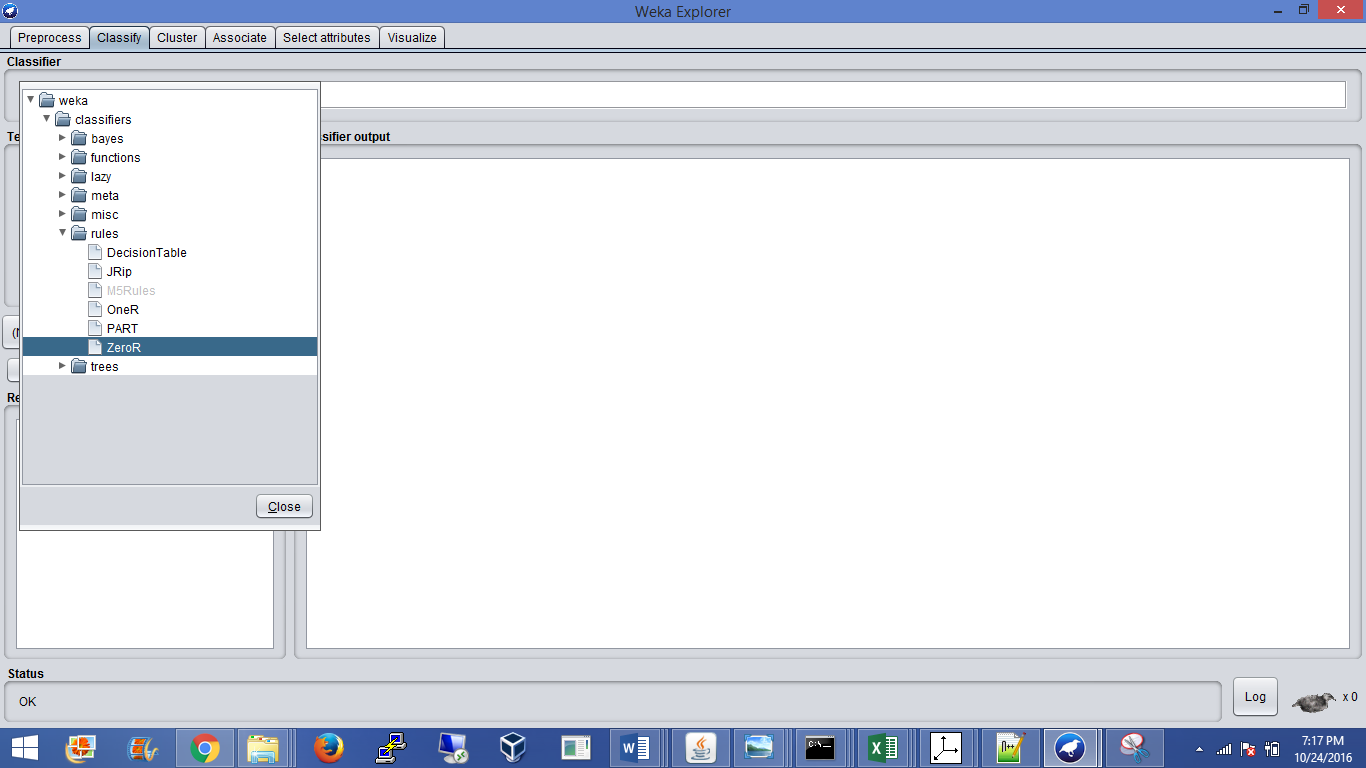


Fig: 4.4 Choosing Zero set from classify

Again there are several options to be selected inside of the classify tab. Test option gives the user the choice of using four different test mode scenarios on the data set.

1. Use training set

2. Supplied training set

3. Cross validation

4. Split percentage

**3. Cluster-** used to apply different tools that identify clusters within the data file.

The Cluster tab opens the process that is used to identify commonalties or clusters of occurrences within the data set and produce information for the user to analyze.

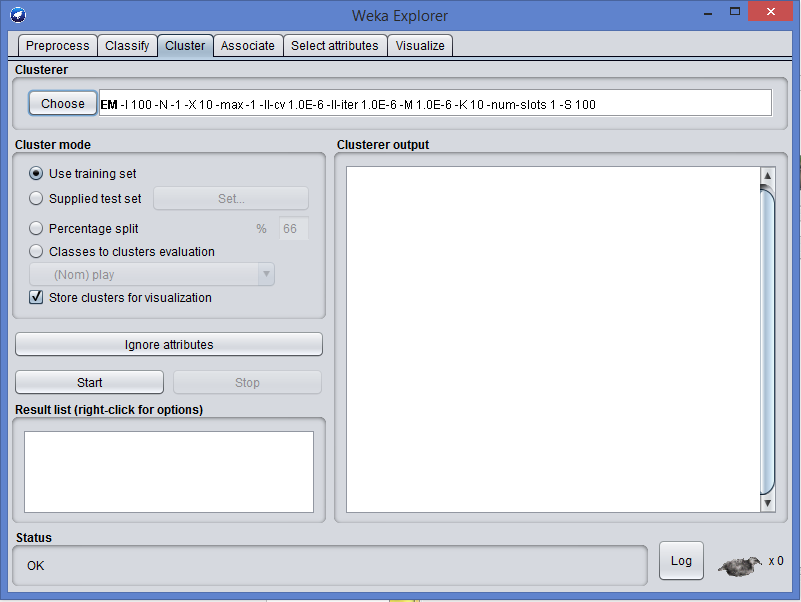


Fig: 4.5 Using training data set

**4. Association-** used to apply different rules to the data file that identify association within the data. The associate tab opens a window to select the options for associations within the data set.

**5. Select attributes-**used to apply different rules to reveal changes based on selected attributes inclusion or exclusion from the experiment

**6. Visualize-** used to see what the various manipulation produced on the data set in a 2D format, in scatter plot and bar graph output.

**2. Experimenter** - this option allows users to conduct different experimental variations on data sets and perform statistical manipulation. The Weka Experiment Environment enables the user to create, run, modify, and analyze experiments in a more convenient manner than is possible when processing the schemes individually. For example, the user can create an experiment that runs several schemes against a series of datasets and then analyze the results to determine if one of the schemes is (statistically) better than the other schemes.

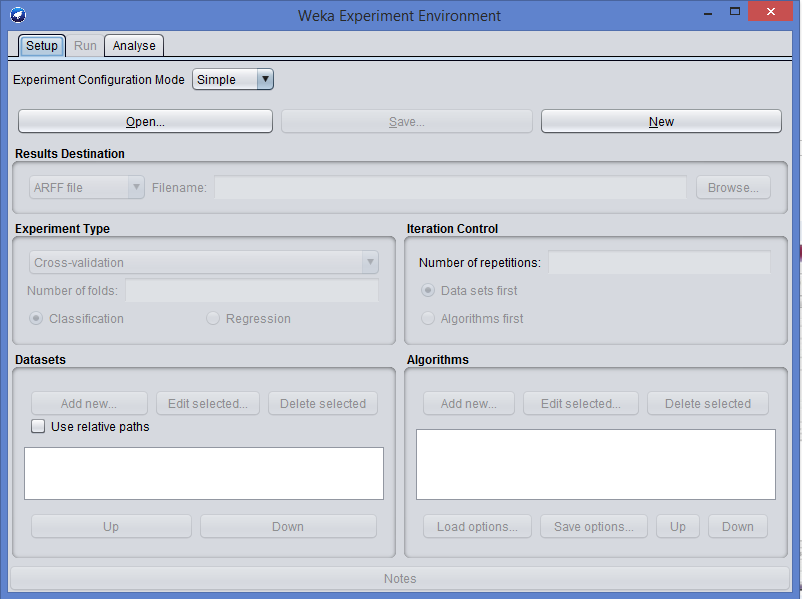


Fig: 4.6 Weka experiment

**Results destination**: ARFF file, CSV file, JDBC database.

**Experiment type**: Cross-validation (default), Train/Test Percentage Split (data randomized).

**Iteration control**: Number of repetitions, Data sets first/Algorithms first.

**Algorithms**: filters

**3. Knowledge Flow** -basically the same functionality as Explorer with drag and drop functionality. The advantage of this option is that it supports incremental learning from previous results

**4. Simple CLI** - provides users without a graphic interface option the ability to execute commands from a terminal window.

**b. Explore the default datasets in weka tool.**

Click the “***Open file…***” button to open a data set and double click on the “***data***” directory. Weka provides a number of small common machine learning datasets that you can use to practice on.

Select the “***iris.arff***” file to load the Iris dataset

**5. Creating new ARFF file**

**Aim:** Creating a new ARFF file

An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software in WEKA, each data entry is an instance of the java class weka.core. Instance, and each instance consists of a For loading datasets in WEKA, WEKA can load ARFF files. Attribute Relation File Format has two sections:

1) The Header section defines relation (dataset) name, attribute name, and type.

2) The Data section lists the data instances.

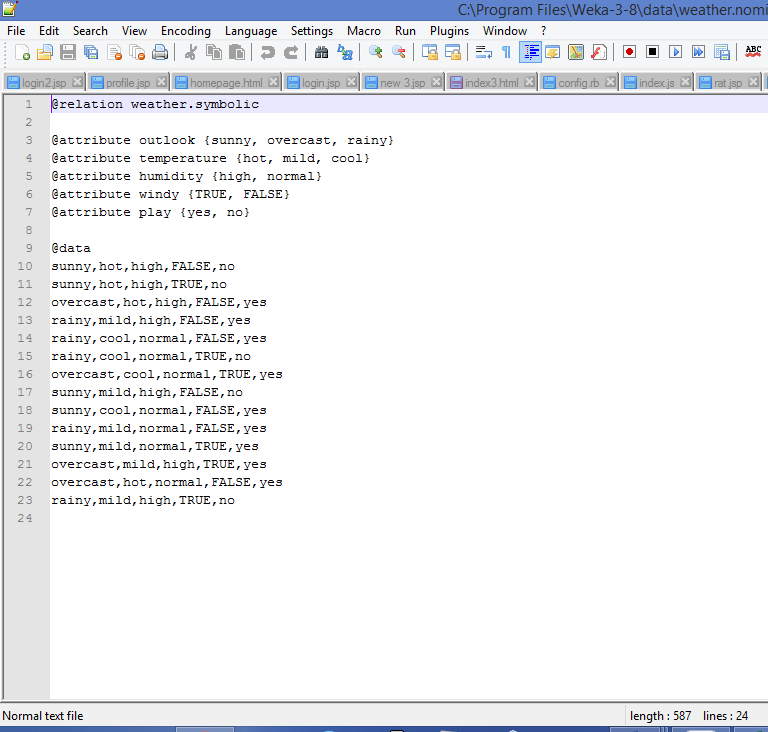


Fig: 5.1 ARFF file

The figure above is from the textbook that shows an ARFF file for the weather data. Lines beginning with a % sign are comments. And there are three basic keywords:

 "@relation" in Header section, followed with relation name.

 "@attribute" in Header section, followed with attributes name and its type (or range).

 "@data" in Data section, followed with the list of data instances.

The external representation of an Instances class Consists of:

**− A header:** Describes the attribute types

**− Data section:** Comma separated list of data

Procedure:

* Create a csv file in any of the editors. (notepad, excel etc)
* Save it as <filename>.csv and open Weks.
* Open the file in Weka and save it in the .ARFF format.
* Open the ARFF file in Wordpad to view the relations between attributes. (see above figure for example)

**6. Data pre-processesing Techniques**

**Aim: 6a)** Pre-process a given dataset based on Attribute selection

To search through all possible combinations of attributes in the data and find which subset of attributes works best for prediction, make sure that you set up attribute evaluator to ‘Cfs SubsetEval’ and a search method to ‘Best First’. The evaluator will determine what method to use to assign a worth to each subset of attributes. The search method will determine what style of search to perform. The options that you can set for selection in the ‘Attribute Selection Mode’ fig no: 3.2

**1. Use full training set**. The worth of the attribute subset is determined using the full set of training data.

**2. Cross-validation**. The worth of the attribute subset is determined by a process of cross-validation. The ‘Fold’ and ‘Seed’ fields set the number of folds to use and the random seed used when shuffling the data.

Specify which attribute to treat as the class in the drop-down box below the test options. Once all the test options are set, you can start the attribute selection process by clicking on ‘Start’ button.

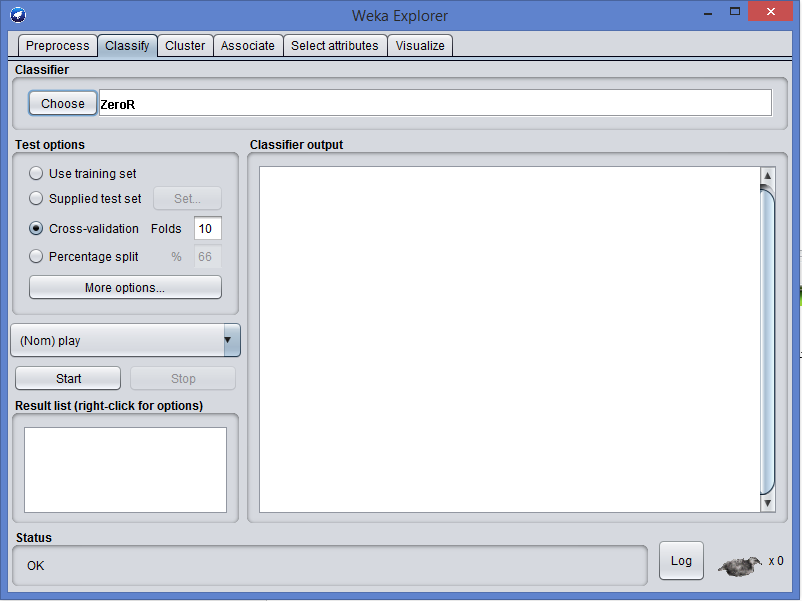


Fig: 6.1 Choosing Cross validation

When it is finished, the results of selection are shown on the right part of the window and entry is added to the ‘Result list’.

**2. Visualizing Results**

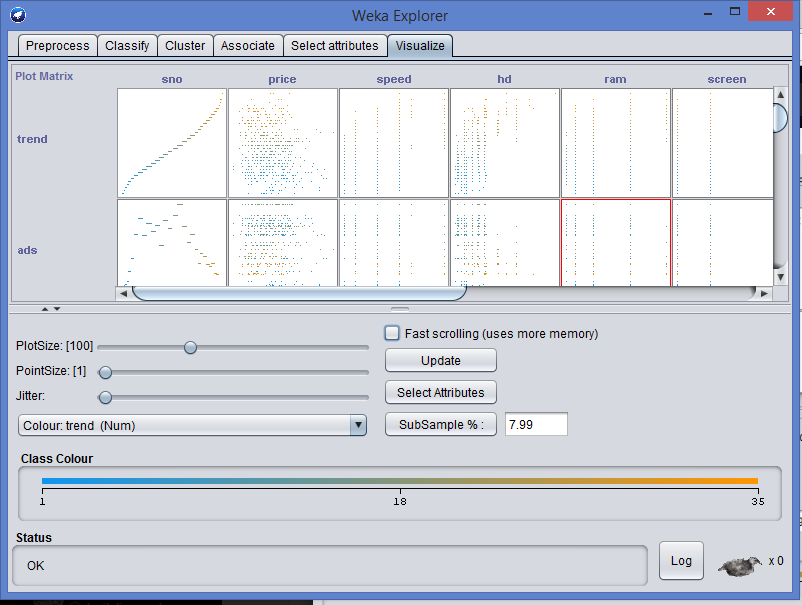


Fig: 6.2 Data Visualization

WEKA’s visualization allows you to visualize a 2-D plot of the current working relation. Visualization is very useful in practice; it helps to determine difficulty of the learning problem. WEKA can visualize single attributes (1-d) and pairs of attributes (2-d), rotate 3-d visualizations (Xgobi-style). WEKA has “Jitter” option to deal with nominal attributes and to detect “hidden” data points.

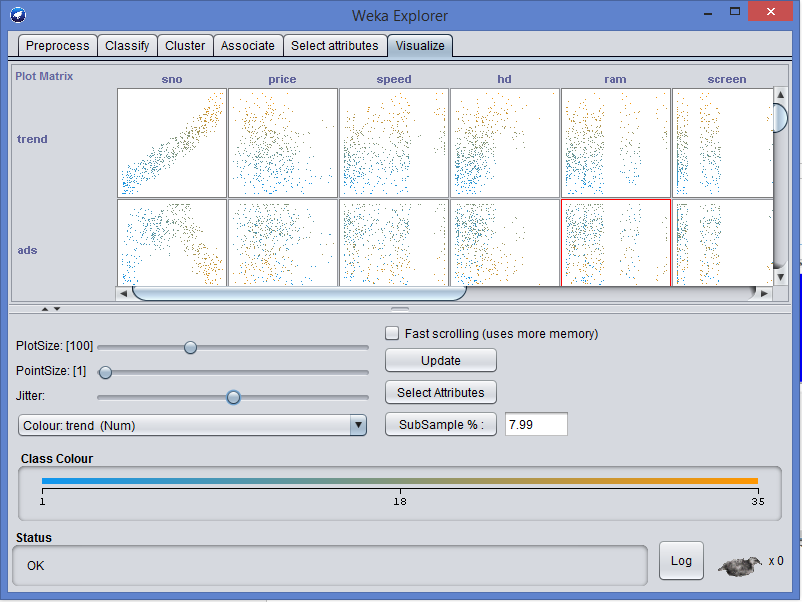


Fig 6.3: Preprocessing with jitter

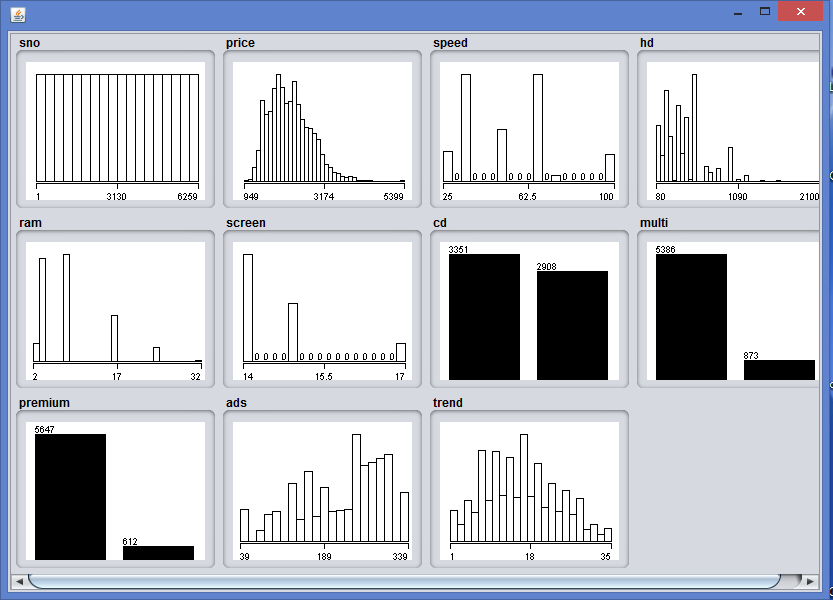


Fig: 6.4 Data visualization

**Aim: 6b).** Pre-process a given dataset based on Handling Missing Values

**Process**: Replacing Missing Attribute Values by the Attribute Mean.

This method is used for data sets with numerical attributes. An example of such a data set is presented in fig no: 3.4

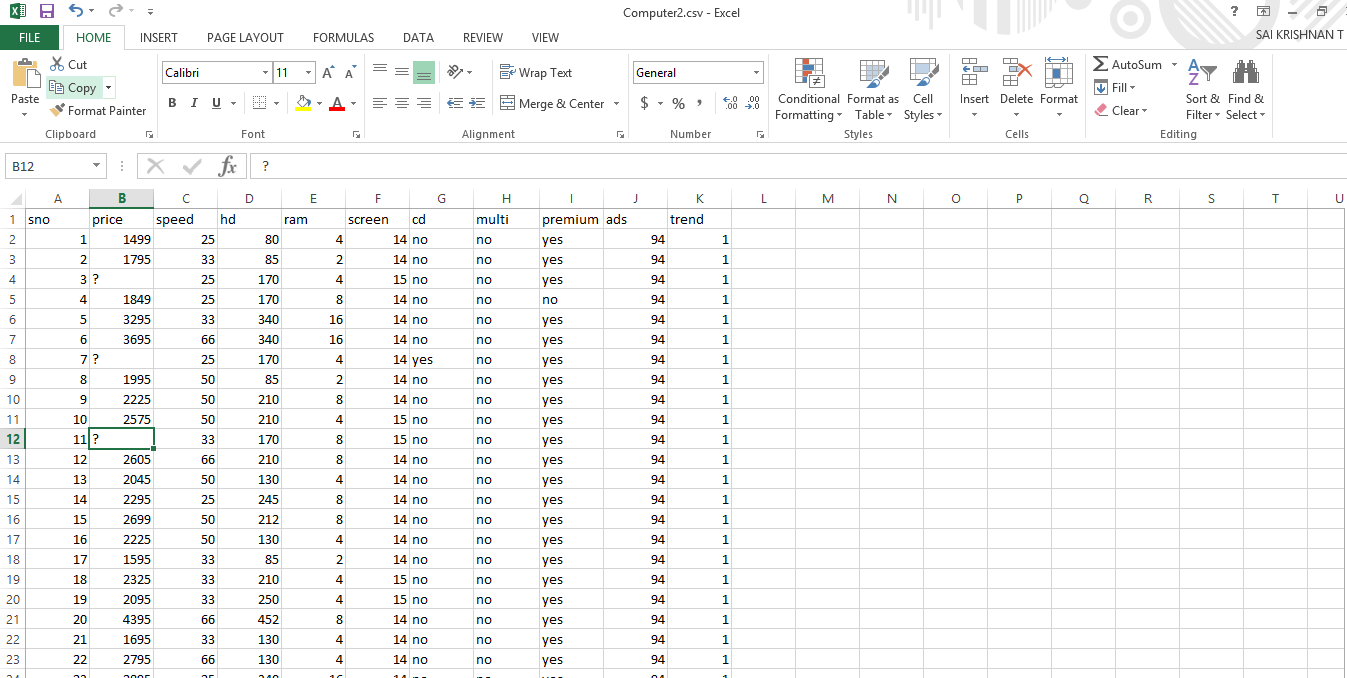


Fig: 6.5 Missing values

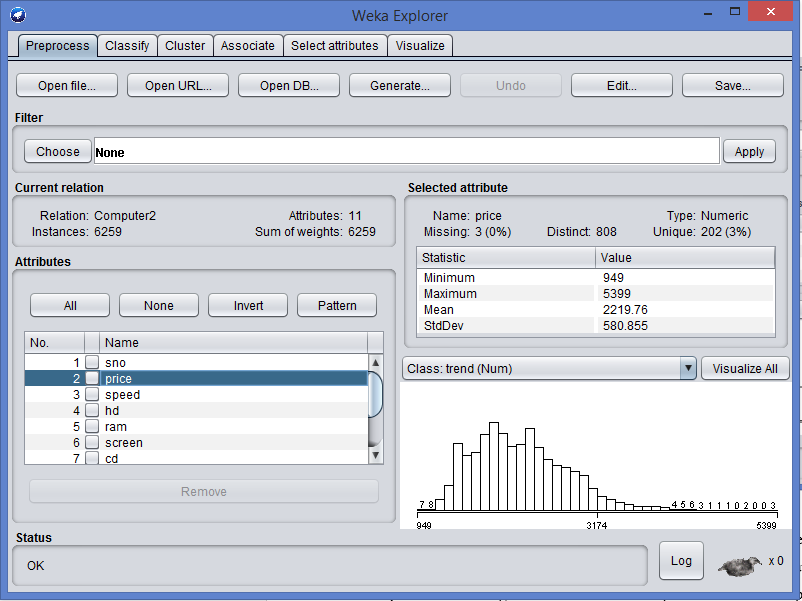


Fig: 6.6 Choosing a dataset

In this method, every missing attribute value for a numerical attribute is replaced by the arithmetic mean of known attribute values. In Fig, the mean of known attribute values for Temperature is 99.2, hence all missing attribute values for Temperature should be replaced by 99.2. The table with missing attribute values replaced by the mean is presented in fig. For symbolic attributes Headache and Nausea, missing attribute values were replaced using the most common value of the Replace Missing Values.

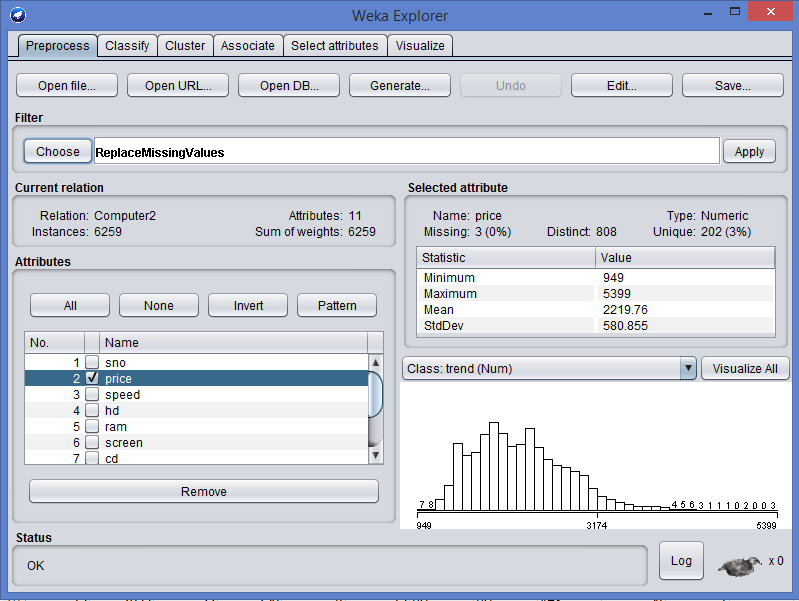


Fig: 6.6 Replace missing values

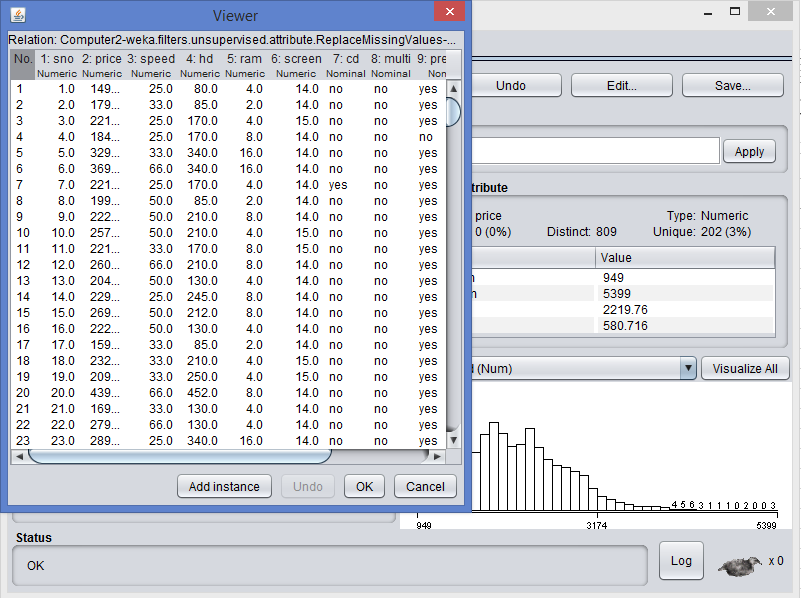


Fig: 6.7 After replacing

**Aim: 6**c). Discretization of data

**Description:**

Discretization is the process of converting continuous attribute to discrete attribute. It is a process of data reduction.

Data discretization transforms numeric data by mapping values to interval or concept labels. Such methods can be used to automatically generate concept hierarchies for the data, which allows for mining at multiple levels of granularity. Discretization techniques include binning, histogram analysis, cluster analysis, decision tree analysis, and correlation analysis. For nominal data, concept hierarchies may be generated based on schema deﬁnitions as well as the number of distinct values per attribute.

Steps:

* Open CSV file in weka.
* Under filter, choose discretize and click apply.
* We can find that attributes are converted to discrete attributes.

Initial Data

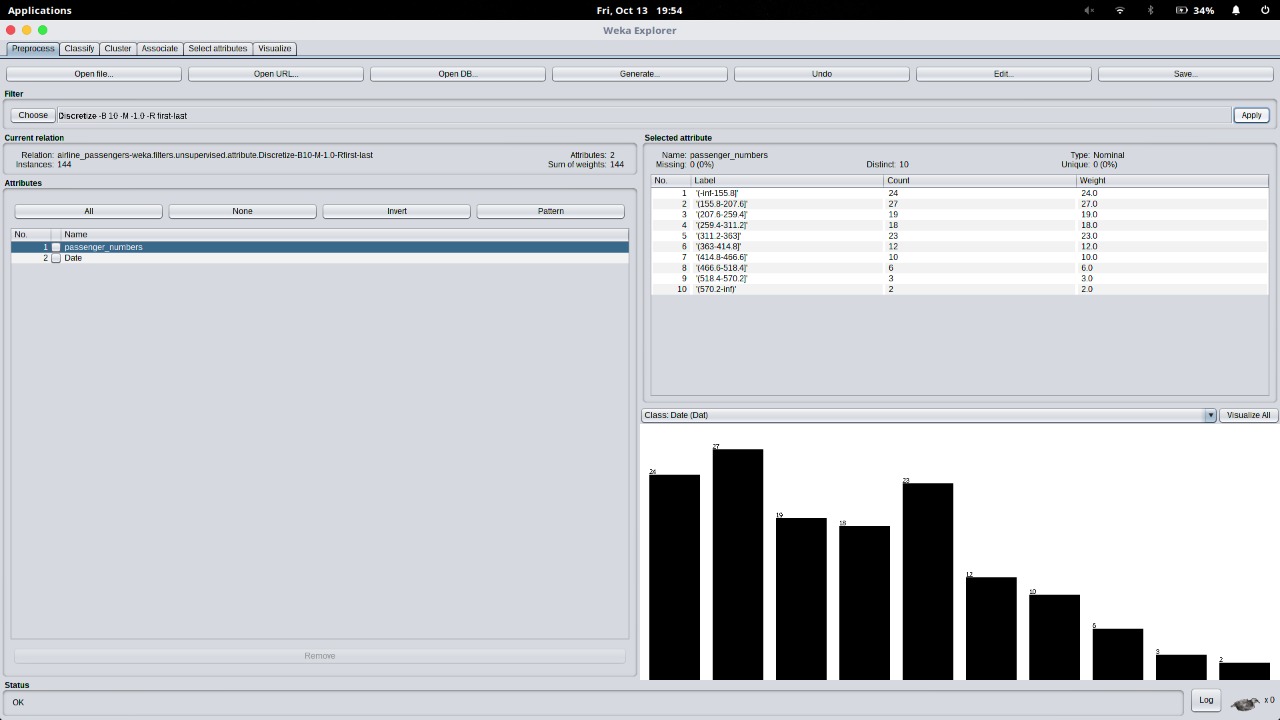


Fig: 6.6 After Discretization

**Aim: 6**d). Normalization of data

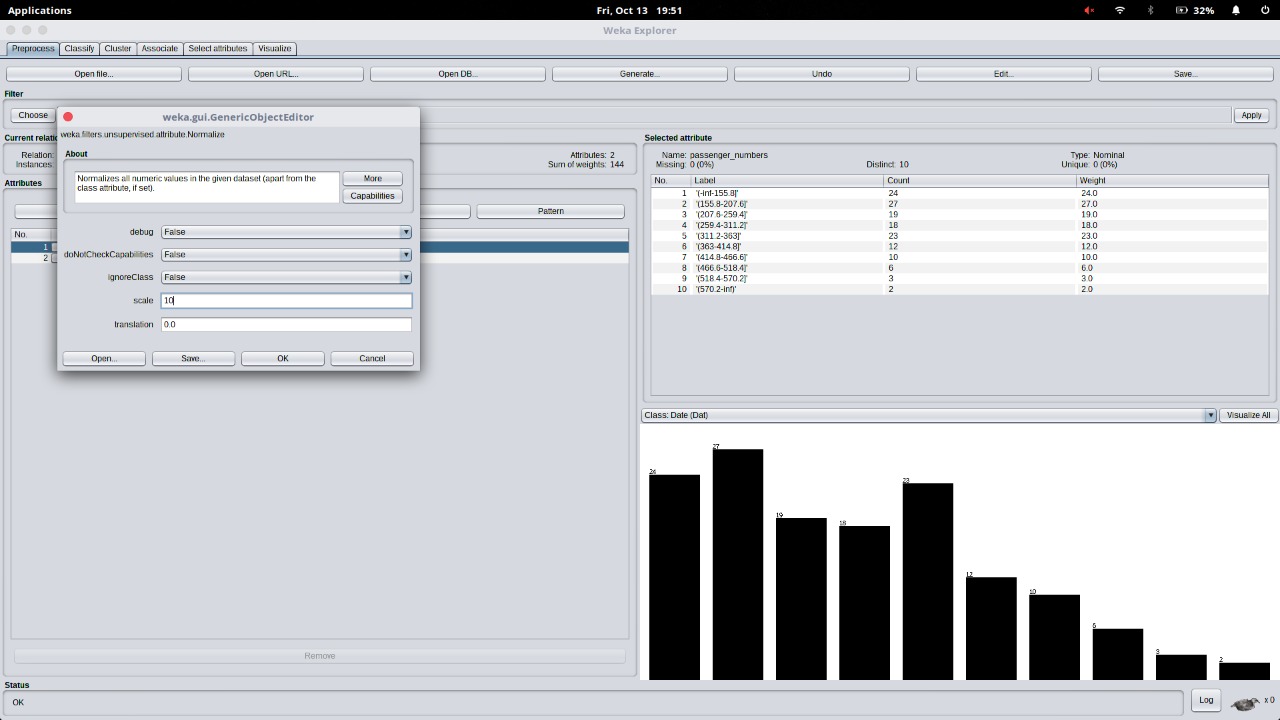
**Description:**

Normalization is the process of converting the scale of an attribute from one form to other form.

Normalization is of 3 types:

* Min-max Normalization: Min max normalization performs a linear transformation on the original data.
* Z score normalization (zero-mean normalization): The values for an attribute, A,are normalized based on the mean(i.e., average) and standard deviation of A.
* Decimal scaling : Normalization by decimal scaling normalizes by moving the decimal point of values of attribute A.

Procedure:

* Select the csv file to be normalized.
* Select normalization from the filters.
* Set scale value to 10.
* Values are displayed in range of 0 to 10.
* 

**7. Generate Association Rules using the Apriori Algorithm**

**Description:**

The Apriori algorithm is an influential algorithm for mining frequent item sets for Boolean association rules. It uses a “bottom-up” approach, where frequent subsets are extended one at a time (a step known as candidate generation, and groups of candidates are tested against the data).

* **Problem:**

|  |  |
| --- | --- |
| **TID** | **ITEMS** |
| 100 | 1,3,4 |
| 200 | 2,3,5 |
| 300 | 1,2,3,5 |
| 400 | 2,5 |

To find frequent item sets for above transaction with a minimum support of 2 having confidence measure of 70% (i.e, 0.7).

**Procedure:**

Step 1:

Count the number of transactions in which each item occurs

|  |  |
| --- | --- |
| **ITEM** | **NO. OF TRANSACTIONS** |
| 1 | 2 |
| 2 | 3 |
| 3 | 3 |
| 4 | 1 |
| 5 | 3 |

Step 2:

Eliminate all those occurrences that have transaction numbers less than the minimum support ( 2 in this case).

|  |  |
| --- | --- |
| **ITEM** | **NO. OF TRANSACTIONS** |
| 1 | 2 |
| 2 | 3 |
| 3 | 3 |
| 5 | 3 |

This is the single items that are bought frequently. Now let’s say we want to find a pair of items that are bought frequently. We continue from the above table (Table in step 2).

Step 3:

We start making pairs from the first item like 1,2;1,3;1,5 and then from second item like 2,3;2,5. We do not perform 2,1 because we already did 1,2 when we were making pairs with 1 and buying 1 and 2 together is same as buying 2 and 1 together. After making all the pairs we get,

|  |
| --- |
| **ITEM PAIRS** |
| 1,2 |
| 1,3 |
| 1,5 |
| 2,3 |
| 2,5 |
| 3,5 |

Step 4:

Now, we count how many times each pair is bought together.

|  |  |
| --- | --- |
| **ITEM PAIRS** | **NO.OF TRANSACTIONS** |
| 1,2 | 1 |
| 1,3 | 2 |
| 1,5 | 1 |
| 2,3 | 2 |
| 2,5 | 3 |
| 3,5 | 2 |

Step 5:

Again remove all item pairs having number of transactions less than 2.

|  |  |
| --- | --- |
| **ITEM PAIRS** | **NO.OF TRANSACTIONS** |
| 1,3 | 2 |
| 2,3 | 2 |
| 2,5 | 3 |
| 3,5 | 2 |

These pair of items is bought frequently together. Now, let’s say we want to find a set of three items that are bought together. We use above table (of step 5) and make a set of three items.

Step 6:

To make the set of three items we need one more rule (It’s termed as self-join), it simply means, from item pairs in above table, we find two pairs with the same first numeric, so, we get (2,3) and (2,5), which gives (2,3,5). Then we find how many times (2, 3, 5) are bought together in the original table and we get the following

|  |  |
| --- | --- |
| **ITEM SET** | **NO. OF TRANSACTIONS** |
| (2,3,5) | 2 |

Thus, the set of three items that are bought together from this data are (2, 3, 5).

Confidence:

We can take our frequent item set knowledge even further, by finding association rules using the frequent item set. In simple words, we know (2, 3, 5) are bought together frequently, but what is the association between them. To do this, we create a list of all subsets of frequently bought items (2, 3, 5) in our case we get following subsets:

* {2}
* {3}
* {5}
* {2,3}
* {3,5}
* {2,5}

Now, we find association among all the subsets.

{2} => {3,5}: ( If ‘2’ is bought , what’s the probability that ‘3’ and ‘5’ would be bought in same transaction)

Confidence = P (3ᴗ5ᴗ2)/ P(2) =2/3 =67%

{3}=>{2,5}= P (3ᴗ5ᴗ2)/ P(3)=2/3=67%

{5}=>{2,3}= P (3ᴗ5ᴗ2)/ P(5)=2/3=67%

{2,3}=>{5}= P (3ᴗ5ᴗ2)/ P(2ᴗ3)=2/2=100%

{3,5}=>{2}= P (3ᴗ5ᴗ2)/ P(3ᴗ5)=2/2=100%

{2,5}=>{3}= P (3ᴗ5ᴗ2)/ P(2ᴗ5)=2/3=67%

Also, considering the remaining 2-items sets, we would get the following associations-

{1}=>{3}=P(1ᴗ3)/P(1)=2/2=100%

{3}=>{1}=P(1ᴗ3)/P(3)=2/3=67%

{2}=>{3}=P(3ᴗ2)/P(2)=2/3=67%

{3}=>{2}=P(3ᴗ2)/P(3)=2/3=67%

{2}=>{5}=P(2ᴗ5)/P(2)=3/3=100%

{5}=>{2}=P(2ᴗ5)/P(5)=3/3=100%

{3}=>{5}=P(3ᴗ5)/P(3)=2/3=67%

{5}=>{3}=P(3ᴗ5)/P(5)=2?3=67%

Eliminate all those having confidence less than 70%.

Hence, the rules would be –

{2,3}=>{5}, {3,5}=>{2}, {1}=>{3},{2}=>{5}, {5}=>{2}.

Now these manual results should be checked with the rules generated in WEKA. So first create a csv file for the above problem, the csv file for the above problem will look like the rows and columns in the above figure. This file is written in excel sheet.

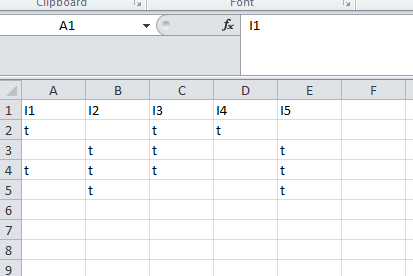


Fig: 7.1 Data for Apriori

**Procedure for running the rules in weka:**

**Step 1:**

Open weka explorer and open the file and then select all the item sets. The figure gives a better understanding of how to do that.

**Step 2:**Now select the association tab and then choose apriori algorithm by setting the minimum support and confidence as shown in the figure.

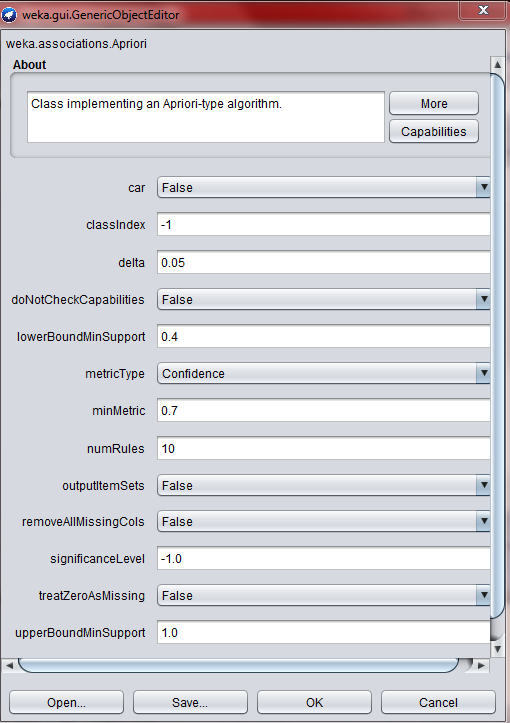


Fig: 7.2 Apriori Parameters

**Step 3:**

Now run the apriori algorithm with the set values of minimum support and the confidence. After running the weka generates the association rules and the respective confidence with minimum support as shown in the figure.

The above csv file has generated 5 rules as shown in the figure:

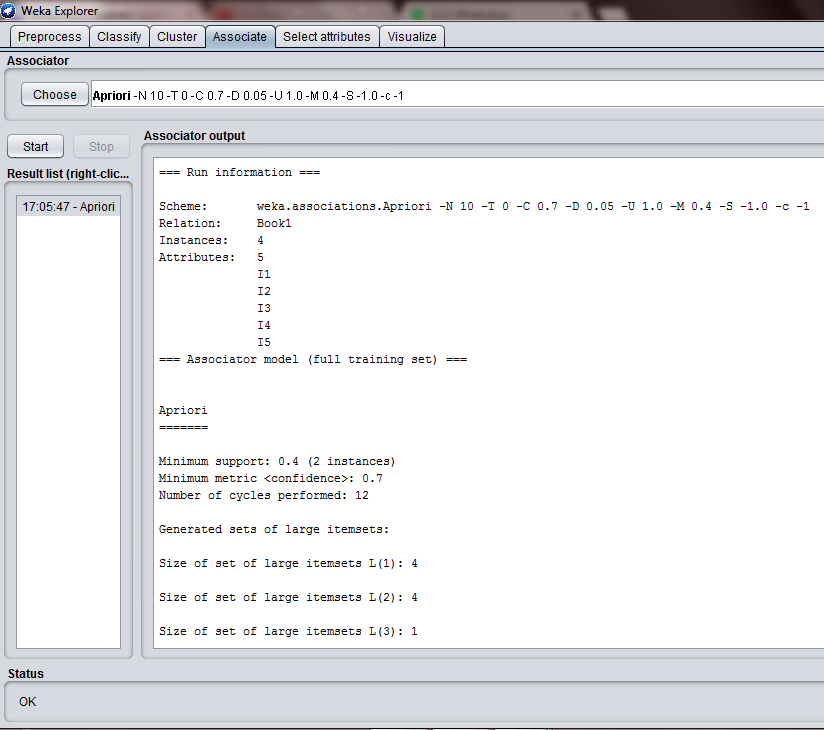


Fig: 7.3 Frequent Item Sets

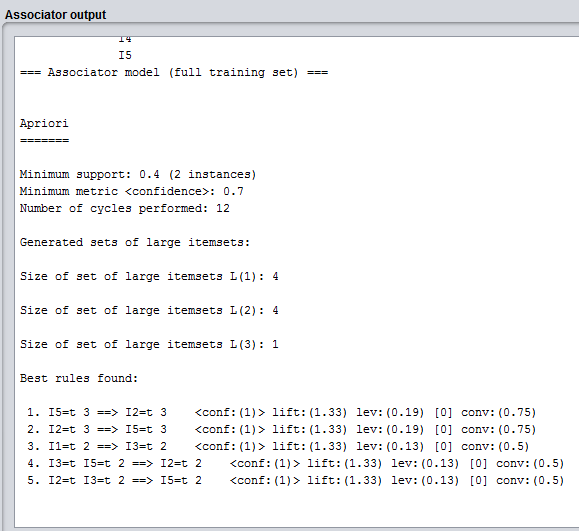


Fig: 7.4 Frequent Item Sets (cont.…)

**Conclusion:**

As we have seen the total rules generated by us manually and by the weka are matching, hence the rules generated are 5.

**Rscript for Association Analysis:**

#includes the libraries

library(arules)

#finding association rules

txn ← read.transactions(file="ItemList.csv", rm.duplicates= TRUE, format="basket",sep=",",cols=1)

txn@itemInfo$labels <- gsub("\"","",txn@itemInfo$labels)

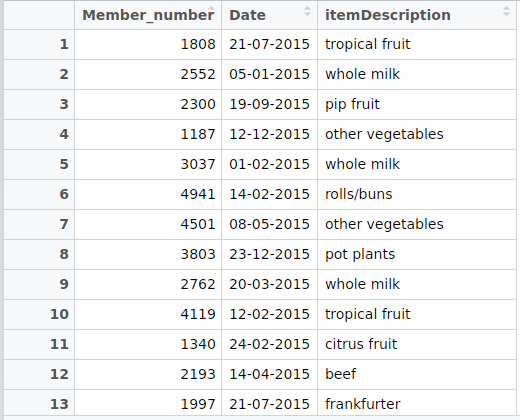
basket\_rules <- apriori(txn,parameter = list(sup = 0.01, conf = 0.5,target="rules"));

inspect(basket\_rules)

#Plot the baskets

itemFrequencyPlot(txn, topN = 5)

**Input DataSet (groceries dataset):**



**Output:**

Apriori

Parameter specification:

confidence minval smax arem aval originalSupport maxtime support minlen maxlen target ext

0.5 0.1 1 none FALSE TRUE 5 0.01 1 10 rules FALSE

Algorithmic control:

filter tree heap memopt load sort verbose

0.1 TRUE TRUE FALSE TRUE 2 TRUE

Absolute minimum support count: 149

items support count

[1] {i1} 0.50 2

[2] {i2} 0.75 3

[3] {i5} 0.75 3

[4] {i3} 0.75 3

[5] {i1,i3} 0.50 2

[6] {i2,i5} 0.75 3

[7] {i2,i3} 0.50 2

[8] {i3,i5} 0.50 2

[9] {i2,i3,i5} 0.50 2

lhs rhs support confidence lift count

[1] {i1} => {i3} 0.50 1.0000000 1.333333 2

[2] {i3} => {i1} 0.50 0.6666667 1.333333 2

[3] {i2} => {i5} 0.75 1.0000000 1.333333 3

[4] {i5} => {i2} 0.75 1.0000000 1.333333 3

[5] {i2,i3} => {i5} 0.50 1.0000000 1.333333 2

[6] {i3,i5} => {i2} 0.50 1.0000000 1.333333 2

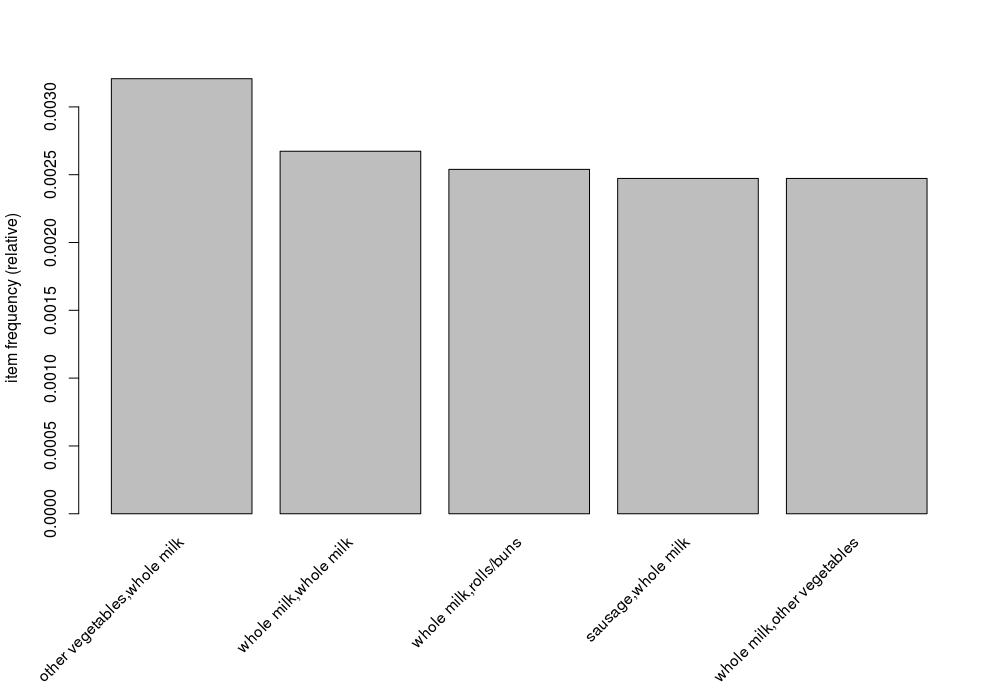
[7] {} => {i1} 0.50 0.5000000 1.000000 2

[8] {} => {i2} 0.75 0.7500000 1.000000 3

[9] {} => {i5} 0.75 0.7500000 1.000000 3

[10] {} => {i3} 0.75 0.7500000 1.000000 3

**Graph:**



**8. Generating Association Rules Using FP Growth Algorithm**

**(8a) Aim:** To generate association rules using FP Growth Algorithm

PROBLEM:

To find all frequent item sets in following dataset using FP-growth algorithm. Minimum support=2 and confidence =70%

|  |  |
| --- | --- |
| **TID** | **ITEMS** |
| 100 | 1,3,4 |
| 200 | 2,3,5 |
| 300 | 1,2,3,5 |
| 400 | 2,5 |

Solution:

Similar to Apriori Algorithm, find the frequency of occurrences of all each item in dataset and then prioritize the items according to its descending order of its frequency of occurrence. Eliminating those occurrences with the value less than minimum support and assigning the priorities, we obtain the following table.

|  |  |  |
| --- | --- | --- |
| **ITEM** | **NO. OF TRANSACTIONS** | **PRIORITY** |
| 1 | 2 | 4 |
| 2 | 3 | 1 |
| 3 | 3 | 2 |
| 5 | 3 | 3 |

Re-arranging the original table, we obtain-

|  |  |
| --- | --- |
| **TID** | **ITEMS** |
| 100 | 1,3 |
| 200 | 2,3,5 |
| 300 | 2,3,5,1 |
| 400 | 2,5 |

**Construction of tree:**

Note that all FP trees have ‘null’ node as the root node. So, draw the root node first and attach the items of the row 1 one by one respectively and write their occurrences in front of it. The tree is further expanded by adding nodes according to the prefixes (count) formed and by further incrementing the occurrences every time they occur and hence the tree is built.

*Prefixes:*

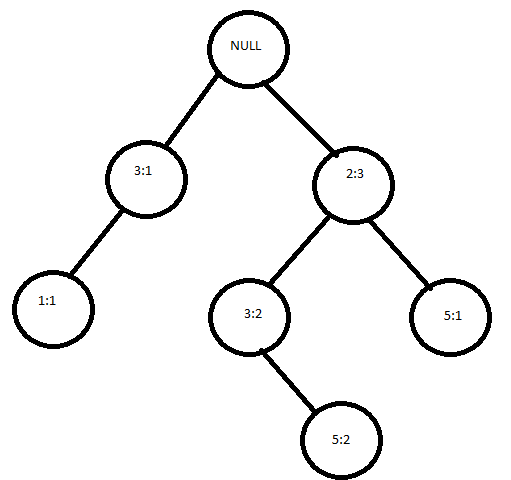
* 1->3:1 2,3,5:1
* 5->2,3:2 2:1
* 3->2:2

*Frequent item sets:*

* 1-> 3:2 /\*2 and 5 are eliminated because they’re less than minimum support, and the occurrence of 3 is obtained by adding the occurrences in both the instances\*/
* Similarly, 5->2,3:2 ; 2:3;3:2
* 3->2 :2

Therefore, the frequent item sets are {3,1}, {2,3,5}, {2,5}, {2,3},{3,5}

The tree is constructed as below:



Generating the association rules for the following tree and calculating the confidence measures we get-

* {3}=>{1}=2/3=67%
* {1}=>{3}=2/2=100%
* {2}=>{3,5}=2/3=67%
* {2,5}=>{3}=2/3=67%
* {3,5}=>{2}=2/2=100%
* {2,3}=>{5}=2/2=100%
* {3}=>{2,5}=2/3=67%
* {5}=>{2,3}=2/3=67%
* {2}=>{5}=3/3=100%
* {5}=>{2}=3/3=100%
* {2}=>{3}=2/3=67%
* {3}=>{2}=2/3=67%

Thus, eliminating all the sets having confidence less than 70%, we obtain the following conclusions:

{1} => {3} , {3,5}=>{2} , {2,3}=>{5} , {2}=>{5}, {5}=>{2}.

As we see there are 5 rules that are being generated manually and these are to be checked against the results in WEKA. Inorder to check the results in the tool we need to follow the similar procedure like Apriori.

So first create a csv file for the above problem, the csv file for the above problem will look like the rows and columns in the above figure. This file is written in excel sheet.

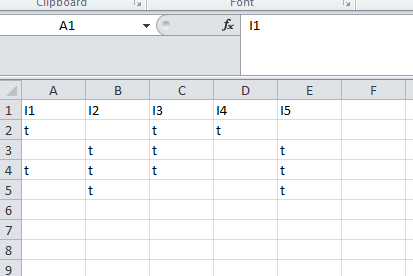


Fig: 8.1 Data for FP Growth

**Procedure for running the rules in weka:**

**Step 1:**

Open weka explorer and open the file and then select all the item sets. The figure gives a better understanding of how to do that.

**Step 2:** Now select the association tab and then choose FPgrowth algorithm by setting the minimum support and confidence as shown in the figure.

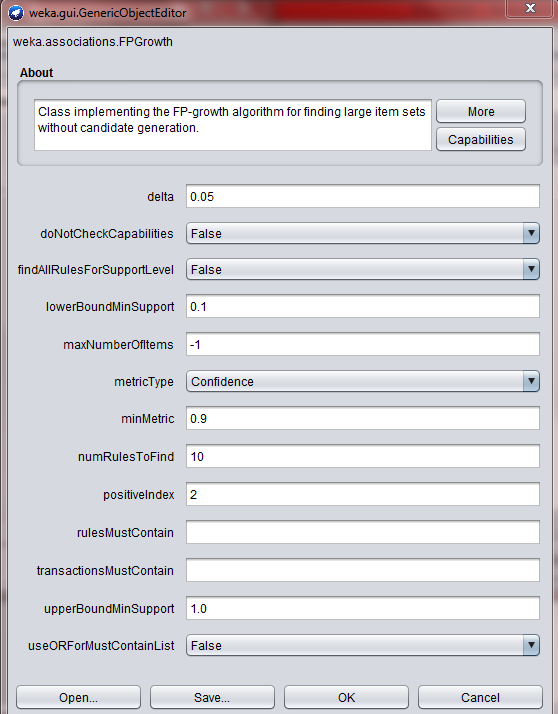


Fig: 8.2 FP growth algorithm

**Step 3:**

Now run the FP Growth algorithm with the set values of minimum support and the confidence. After running the weka generates the association rules and the respective confidence with minimum support as shown in the figure.

The above csv file has generated 5 rules as shown in the figure:

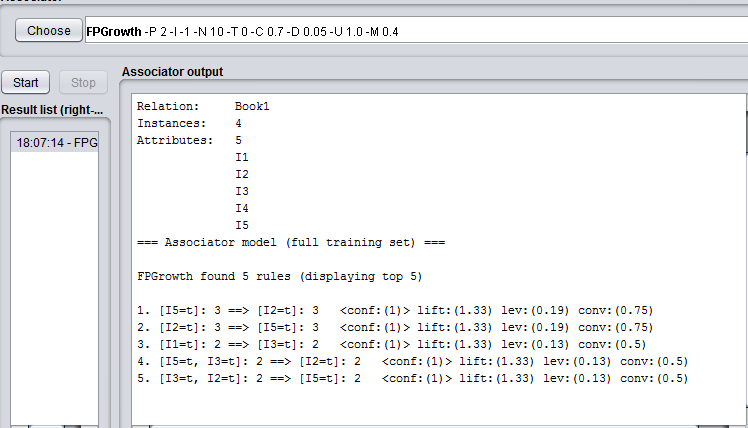


Fig: 8.3 Output for FP Growth

**Conclusion:**

As we have seen the total rules generated by us manually and by the weka are matching, hence the rules generated are 5.

**9. Build a Decision Tree by using J48 algorithm**

**(9a) Aim:** Generate a Decision Tree by using J48 algorithm.

**DESCRIPTION:**

Decision tree learning is one of the most widely used and practical methods for inductive inference over supervised data. It represents a procedure for classifying categorical databased on their attributes. This representation of acquired knowledge in tree form is intuitive and easy to assimilate by humans.

**ILLUSTRATION:**

**Build a decision tree for the following data**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AGE | INCOME | STUDENT | CREDIT\_RATING | BUYS\_COMPUTER |
| Youth | High | No | Fair | No |
| Youth | High | No | Excellent | No |
| Middle aged | High | No | Fair | Yes |
| Senior | Medium | No | Fair | Yes |
| Senior | Low | Yes | Fair | Yes |
| Senior | Low | Yes | Excellent | No |
| Middle aged | Medium | Yes | Excellent | Yes |
| Youth | Low | No | Fair | No |
| Youth | Medium | Yes | Fair | Yes |
| Senior | Medium | Yes | Fair | Yes |
| Youth | Medium | Yes | Excellent | Yes |
| Middle aged | Medium | No | Excellent | Yes |
| Middle aged | High | Yes | Fair | Yes |
| Senior | Medium | No | Excellent | No |

The entropy is a measure of the uncertainty associated with a random variable. As uncertainty increases, so does entropy, values range from [0-1] to present the entropy of information

Entropy (D) =

Information gain is used as an attribute selection measure; pick the attribute having the highest information gain, the gain is calculated by:

Gain (D, A) = Entropy (D) -

Where, D: A given data partition

A: Attribute

V: Suppose we were partition the tuples in D on some attribute A having v distinct values D is split into v partition or subsets, (D1, D2….. Dj) , where Dj contains those tuples in D that have outcome Aj of A.

* Class P: buys\_computer=”yes”
* Class N: buys\_computer=”no”

Entropy (D) = -9/14log (9/14)-5/15log (5/14) =0.940

* Compute the expected information requirement for each attribute start with the attribute age

Gain (age, D) = Entropy (D) -

= Entropy ( D ) – 5/14Entropy(Syouth) - 4/14Entropy(Smiddle-aged) - 5/14Entropy(Ssenior)

= 0.940-0.694

=0.246

Similarly, for other attributes,

Gain (Income, D) =0.029

Gain (Student, D ) = 0.151

Gain (credit\_rating, D) = 0.04

AGE?

Senior

Youth

|  |  |  |  |
| --- | --- | --- | --- |
| Income | Student | Credit\_rating | Class |
| High | No | Fair | No |
| High | No | Excellent | No |
| Medium | No | Fair | No |
| Low | Yes | Fair | Yes |
| medium | Yes | excellent | yes |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Income | student | | Credit\_rating | class | |
| High | | No | Fair | | ***Yes*** |
| Low | | Yes | Excellent | | ***Yes*** |
| Medium | | No | Excellent | | ***Yes*** |
| High | | yes | Fair | | ***yes*** |

|  |  |  |  |
| --- | --- | --- | --- |
| income | Student | Credit\_rating | class |
| Medium | No | Fair | No |
| Low | Yes | Fair | Yes |
| Low | Yes | Excellent | No |
| Medium | Yes | Fair | Yes |
| medium | No | Excellent | No |

The attribute age has the highest information gain and therefore becomes the splitting attribute at the root node of the decision tree. Branches are grown for each outcome of age. These tuples are shown partitioned accordingly.

Now, calculating information gain for subtable (age<=30)

I (2,3) = -(2/5)log(2/5)-(3/5)log(3/5)=0.971

\*INCOME\*

Income=”high” S11=0, S12=2

I=0

Income=”medium” S21=1 S22=1

I (S21, S23) = 1

Income=”low” S31=1 S32=0

I=0

Entropy for income

E( income ) = (2/5)(0) + (2/5)(1) + (1/5)(0) = 0.4

Gain( income ) = 0.971 - 0.4 = 0.571

Similarly,

Gain(student)=0.971

Gain(credit)=0.0208

Gain( student) is highest

no

no

senior

<=30

31..40

|  |  |  |  |
| --- | --- | --- | --- |
| income | student | Credit\_rating | class |
| Medium | No | Fair | No |
| Low | Yes | Fair | Yes |
| Low | Yes | Excellent | No |
| Medium | Yes | Fair | Yes |
| medium | No | Excellent | No |

For age > 40, information gain= -(3/5)log(3/5)-(2/5)log(2/5)=0.971

\*income\* income=”low” I=1, income=”medium” I=0.917 , E(income)=(2/5)(1)+(3/5)(0.917)=0.9502

Gain(income)=0.0208.Similarly, gain( student)=0.0208, gain(credit)=0.971 and this would result in final decision tree

excellent

fair

senior

Middle\_aged

yes

no

youth

A decision tree for the concept buys\_computer, indicating whether a customer at AllElectronics is likely to purchase a computer. Each internal (non-leaf) node represents a test on an attribute. Each leaf node represents a class ( either buys\_computer=”yes” or buys\_computer=”no”.

CHECKING THE RESULT IN WEKA**:**

First create a csv file for the above problem,the csv file for the above problem will look like the rows and columns in the above figure. This file is written in excel sheet.

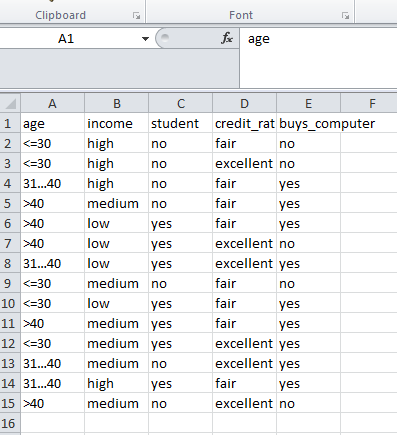


Fig: 9.1 Input Data for Decision tree algorithm

**Procedure for running the rules in weka:**

Step 1:

Open weka explorer and open the file and then select all the item sets.The figure gives a better understanding of how to do that.

Step2:

Now select the classify tab in the tool and click on start button and then we can see the result of the problem as below

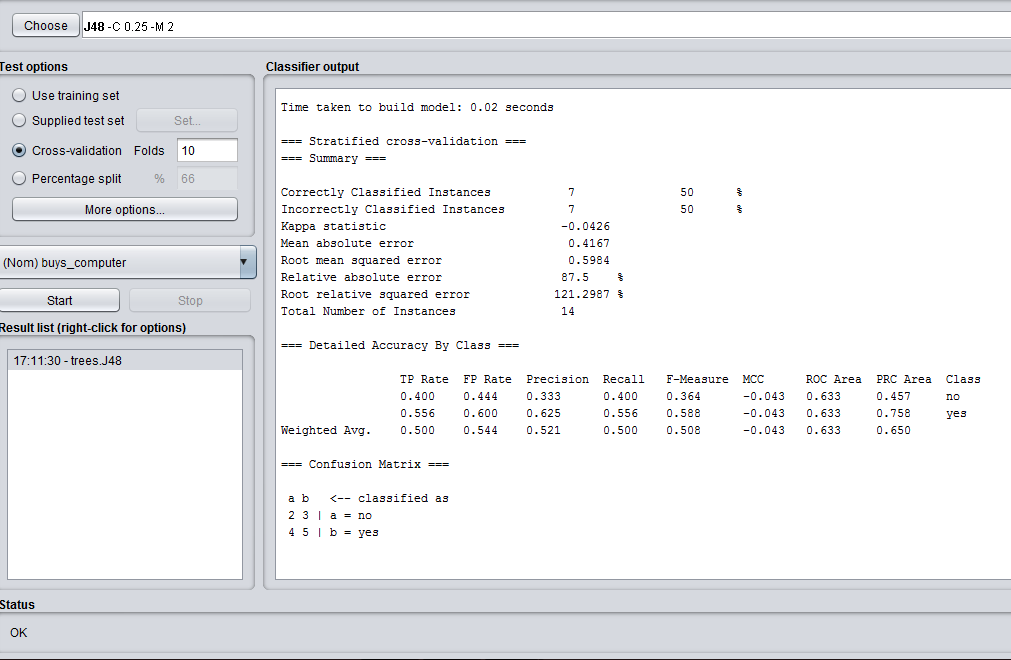


Fig: 9.2 Decision tree output summary

Step3:

Check the main result which we got manually and the result in weka by right clicking on the result and visualizing the tree.

The visualized tree in weka is as shown below:

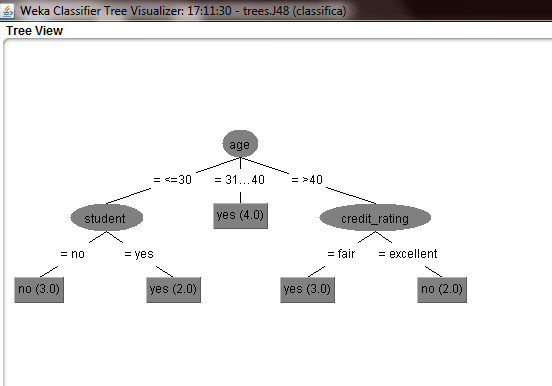


Fig: 9.3 Decision tree

**Conclusion:**

The solution what we got manually and the weka both are same.

**Rscript for Decision Tree**

#load libraries

library('rpart')

library('rpart.plot')

# Classification Tree with rpart

library(rpart)

input.dat = read.csv("data.csv",header=TRUE,sep = "\t")

input.dat

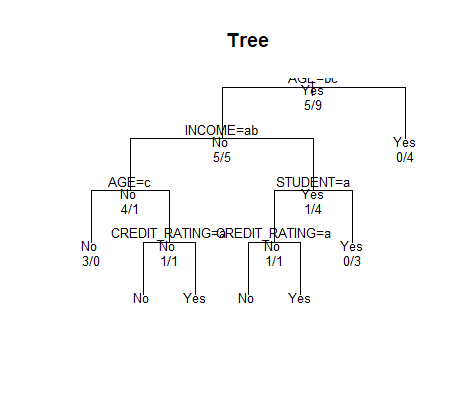
# grow tree

fit <- rpart(BUYS\_COMPUTER ~ AGE + INCOME + STUDENT + CREDIT\_RATING, method = 'class',data = input.dat,control =rpart.control(minsplit = 1,minbucket=1, cp=0))

# plot tree

plot(fit, uniform=TRUE, main="Tree")

text(fit, use.n=TRUE, all=TRUE, cex=.8)



**10. Naïve Bayes classification on a given data set**

**AIM:**

To apply naïve bayes classifier on a given data set.

**Description:**

In machine learning, Naïve Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes’ Theorem with strong (naïve) independence assumptions between the features.

Example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AGE | INCOME | STUDENT | CREDIT\_RATING | BUYS\_COMPUTER |
| <=30 | High | No | Fair | No |
| <=30 | High | No | Excellent | No |
| 31..40 | High | No | Fair | Yes |
| >40 | Medium | No | Fair | Yes |
| >40 | Low | Yes | Fair | Yes |
| >40 | Low | Yes | Excellent | No |
| 31..40 | Medium | Yes | Excellent | Yes |
| <=30 | Low | No | Fair | No |
| <=30 | Medium | Yes | Fair | Yes |
| >40 | Medium | Yes | Fair | Yes |
| <=30 | Medium | Yes | Excellent | Yes |
| 31..40 | Medium | No | Excellent | Yes |
| 31..40 | High | Yes | Fair | Yes |
| >40 | Medium | No | Excellent | No |

**CLASS:**

C1:buys\_computer = ‘yes’

C2:buys\_computer=’no’

**DATA TO BE CLASSIFIED:**

X= (age<=30, income=Medium, Student=Yes, credit\_rating=Fair)

* P(C1): P(buys\_computer=”yes”)= 9/14 = 0.643

P (buys\_computer=”no”) =5/14=0.357

* Compute P(X/C1) and p(x/c2) we get:

1. P( age=”<=30” |buys\_computer=”yes”)=2/9
2. P( age=”<=30”|buys\_computer=”no”)=3/5
3. P(income=”medium ”|buys\_computer=”yes”)=4/9
4. P(income=”medium ”|buys\_computer=”no”)=2/5
5. P(student=”yes ”|buys\_computer=”yes”)=6/9
6. P(student=”yes” |buys\_computer=”no”)=1/5=0.2
7. P(credit\_rating=”fair ”|buys\_computer=”yes”)=6/9
8. P(credit\_rating=”fair” | buys\_computer=”no”)=2/5

* X=(age<=30, income=medium, student=yes, credit\_rating=fair)

P(X/C1): P (X/buys\_computer=”yes”)=2/9\*4/9\*6/9\*6/9=32/1134

P(X/C2): P(X/buys\_computer=”no”)=3/5\*2/5\*1/5\*2/5=12/125

P(C1/X) = P(X/C1)P(C1) \* P(X/buys\_computer=”yes”) \* P(buys\_computer=”yes”)

=(32/1134)\*(9/14)

=0.019

P(C2/X) = p(x/c2)\*p(c2)

P (X/buys\_computer=”no”)\*P(buys\_computer=”no”)

= (12/125)\*(5/14)

= 0.007

Therefore, conclusion is that the given data belongs to C1 since P(C1/X)>P(C2/X)

Checking the result in the WEKA tool:

In order to check the result in the tool we need to follow a procedure.

Step 1:

Create a csv file with the above table considered in the example.the arff file will look as shown below:

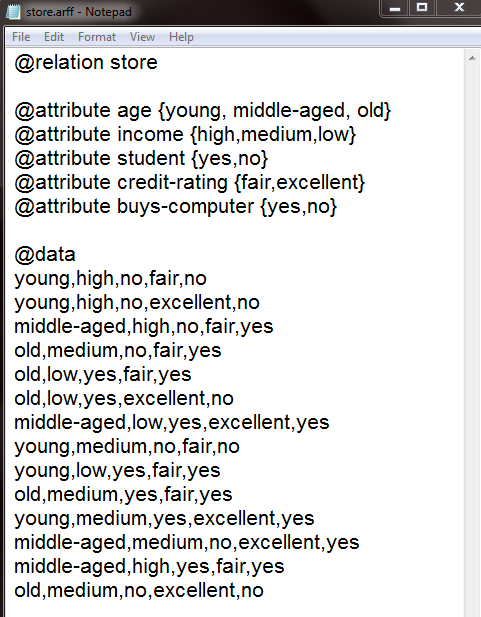


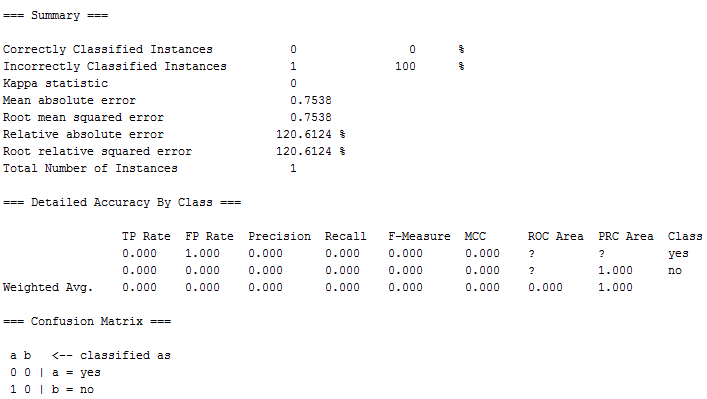
Fig: 10.1 Input data

Step 2:

Now open weka explorer and then select all the attributes in the table.

Step 3:

Select the classifier tab in the tool and choose baye’s folder and then naïve baye’s classifier to see the result as shown below.



**R program for Naïve Bayes Classifier**

library(e1071)

library(rjson)

#Load Data

input.dat <- fromJSON(file = "credit-g.json")

#Train the Model

model <- naiveBayes(BUYS\_COMPUTER ~ AGE + INCOME + STUDENT + CREDIT\_RATING, data = input.dat)

#Predict using that model

input.dat[1,]

predict(model, input.dat[1,-5])

Output:

AGE INCOME STUDENT CREDIT\_RATING BUYS\_COMPUTER

1 Youth High No Fair No

2 Youth High No Excellent No

3 Middle aged High No Fair Yes

4 Senior Medium No Fair Yes

5 Senior Low Yes Fair Yes

6 Senior Low Yes Excellent No

7 Middle aged Medium Yes Excellent Yes

8 Youth Low No Fair No

9 Youth Medium Yes Fair Yes

10 Senior Medium Yes Fair Yes

11 Youth Medium Yes Excellent Yes

12 Middle aged Medium No Excellent Yes

13 Middle aged High Yes Fair Yes

14 Senior Medium No Excellent No

> #Train the Model

> model <- naiveBayes(BUYS\_COMPUTER ~ AGE + INCOME + STUDENT + CREDIT\_RATING, data = input.dat)

> #Predict using that model

> input.dat[1,]

AGE INCOME STUDENT CREDIT\_RATING BUYS\_COMPUTER

1 Youth High No Fair No

> predict(model, input.dat[1,-5])

[1] No

Levels: No Yes

**11. K-means clustering**

**DESCRIPTION:**

K-means algorithm 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 partitioning of the data into Voronoi cells.

**ILLUSTRATION:**

As a simple illustration of a k-means algorithm, consider the following data set consisting of the scores of two variables on each of the five variables.

|  |  |  |
| --- | --- | --- |
| I | X1 | X2 |
| A | 1 | 1 |
| B | 1 | 0 |
| C | 0 | 2 |
| D | 2 | 4 |
| E | 3 | 5 |

This data set is to be grouped into two clusters: As a first step in finding a sensible partition, let the A & C values of the two individuals furthest apart (using the Euclidean distance measure), define the initial cluster means, giving:

|  |  |  |
| --- | --- | --- |
| Cluster | Individual | Mean Vector(Centroid) |
| Cluster1 | A | (1,1) |
| Cluster2 | C | (0,2) |

The remaining individuals are now examined in sequence and allocated to the cluster to which they are closest, in terms of Euclidean distance to the cluster mean. The mean vector is recalculated each time a new member is added. This leads to the following series of steps:

|  |  |  |
| --- | --- | --- |
|  | A | C |
| A | 0 | 1.4 |
| B | 1 | 2.5 |
| C | 1.4 | 0 |
| D | 3.2 | 2.82 |
| E | 4.5 | 4.2 |

Initial partitions have changed, and the two clusters at this stage having the following characteristics.

|  |  |  |
| --- | --- | --- |
|  | Individual | Mean vector( Centroid) |
| Cluster 1 | A,B | (1,0.5) |
| Cluster 2 | C,D,E | (1.7,3.7) |

But we cannot yet be sure that each individual has been assigned to the right cluster. So, we compare each individual’s distance to its own cluster mean and to that of the opposite cluster. And, we find:

|  |  |  |
| --- | --- | --- |
| I | A | C |
| A | 0.5 | 2.7 |
| B | 0.5 | 3.7 |
| C | 1.8 | 2.4 |
| D | 3.6 | 0.5 |
| E | 4.9 | 1.9 |

The individuals C is now relocated to Cluster 1 due to its less mean distance with the centroid points. Thus, its relocated to cluster 1 resulting in the new partition

|  |  |  |
| --- | --- | --- |
|  | Individual | Mean vector(Centroid) |
| Cluster 1 | A,B,C | (0.7,1) |
| Cluster 2 | D,E | (2.5,4.5) |

The iterative relocation would now continue from this new partition until no more relocation occurs. However, in this example each individual is now nearer its own cluster mean than that of the other cluster and the iteration stops, choosing the latest partitioning as the final cluster solution.

Also, it is possible that the k-means algorithm won’t find a final solution. In this case, it would be a better idea to consider stopping the algorithm after a pre-chosen maximum number of iterations.

Checking the solution in weka:

In order to check the result in the tool we need to follow a procedure.

Step 1:

Create a csv file with the above table considered in the example.the csv file will look as shown below:

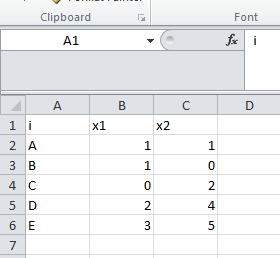


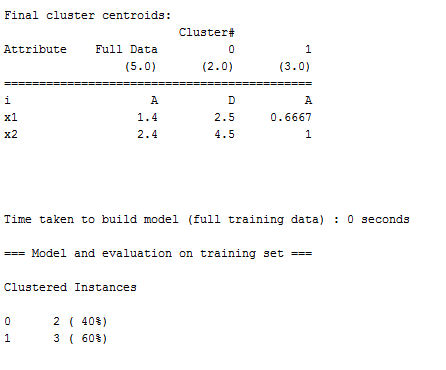
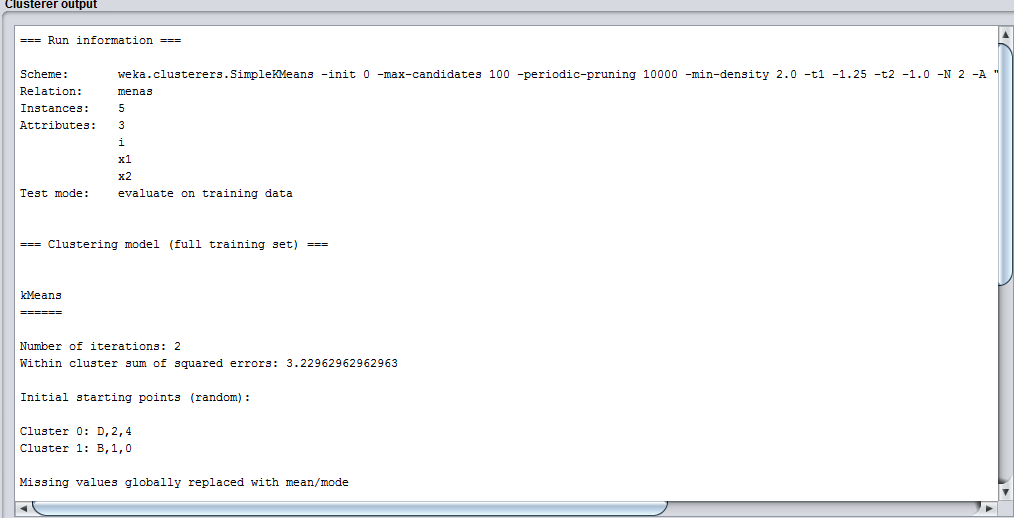
Fig: 11.1 Input data

Step 2:

Now open weka explorer and then select all the attributes in the table.

Step 3:

Select the cluster tab in the tool and choose normal k-means technique to see the result as shown below.



**Rscript for K-means:**

#load the required packages

#ggplot is used to create plots

library(ggplot2)

#plot the iris data set

ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) + geom\_point()

#clustering

set.seed(20)

#forming a kmeans cluster

irisCluster <- kmeans(iris[,3:4], 3, nstart=20)

irisCluster

#plot to see the clusters

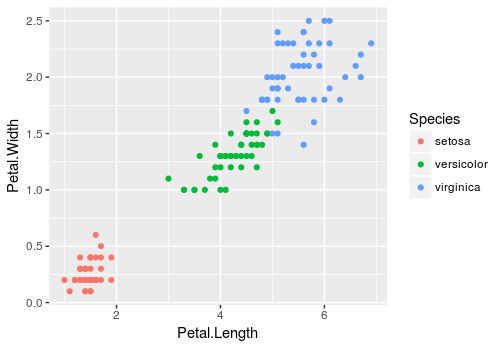
irisCluster$cluster <- as.factor(irisCluster$cluster)

ggplot(iris, aes(Petal.Length, Petal.Width, color = irisCluster$cluster)) + geom\_point()

*Iris data set*

**

*Plotting iris data set*

**

*Output:*

K-means clustering with 3 clusters of sizes 50, 52, 48

Cluster means:

Petal.Length Petal.Width

1 1.462000 0.246000

2 4.269231 1.342308

3 5.595833 2.037500

Clustering vector:

[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

[67] 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 2 3 3 3 3 3

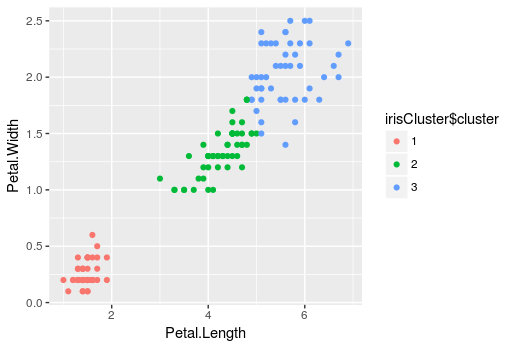
[133] 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3

Within cluster sum of squares by cluster:

[1] 2.02200 13.05769 16.29167

(between\_SS / total\_SS = 94.3 %)

*Clusters Formed:*



**12. Regression analysis**

**Aim:** To generate a linear regression using R

**Description:**

Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variable is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.

The general mathematical equation for a linear regression is −

y = ax + b

Following is the description of the parameters used −

* **y** is the response variable.
* **x** is the predictor variable.
* **a** and **b** are constants which are called the coefficients.

## Steps to Establish a Regression

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is −

* Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
* Create a relationship model using the **lm()** functions in R.
* Find the coefficients from the model created and create the mathematical equation using these
* Get a summary of the relationship model to know the average error in prediction. Also called **residuals**.
* To predict the weight of new persons, use the **predict()** function in R.

**Rscript:**

# Create the predictor and response variable.

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

relation <- lm(y~x)

# Give the chart file a name.

png(file = "linearregression.png")

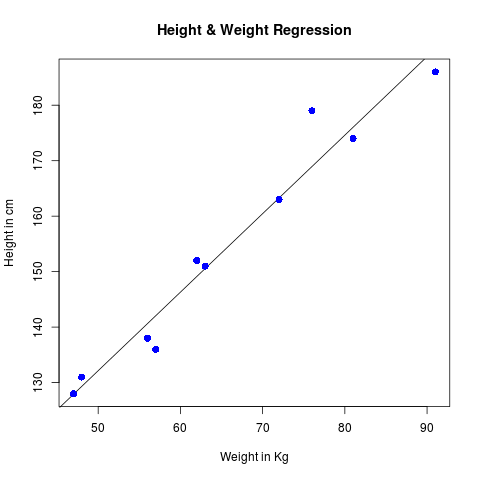
plot(y,x,col = "blue",main = "Height & Weight Regression",

abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")

# Save the file.

dev.off()

**Output:**

****