WEEK-1

About WEKA

WEKA - Open source software provides tools for data preprocessing, implementation of several Machine Learning algorithms, and visualization tools so that you can develop machine learning techniques and apply them to real-world data mining problems. What WEKA offers is summarized in the following diagram –

If you observe the beginning of the flow of the image, you will understand that there are many stages in dealing with Big Data to make it suitable for machine learning −

First, you will start with the raw data collected from the field. This data may contain several null values and irrelevant fields. You use the data preprocessing tools provided in WEKA to cleanse the data.

Then, you would save the preprocessed data in your local storage for applying ML algorithms.

Next, depending on the kind of ML model that you are trying to develop you would select one of the options such as **Classify, Cluster**, or **Associate**. The **Attributes Selection** allows the automatic selection of features to create a reduced dataset.

Note that under each category, WEKA provides the implementation of several algorithms. You would select an algorithm of your choice, set the desired parameters and run it on the dataset.

Then, WEKA would give you the statistical output of the model processing. It provides you a visualization tool to inspect the data.

The various models can be applied on the same dataset. You can then compare the outputs of different models and select the best that meets your purpose.

Thus, the use of WEKA results in a quicker development of machine learning models on the whole.

Now that we have seen what WEKA is and what it does, in the next chapter let us learn how to install WEKA on your local computer.

Weka – Installation:

To install WEKA on your machine, visit WEKA's official website and download the installation file. WEKA supports installation on Windows, Mac OS X and Linux. You just need to follow the instructions on this page to install WEKA for your OS.

The steps for installing on Mac are as follows −

- Download the Mac installation file.
- Double click on the downloaded **weka-3-8-3-corretto-jvm.dmg file**.

You will see the following screen on successful installation.

WEEK 1

List of Experiments: 1. Creation of a Data Warehouse.

I. **Build Data Warehouse/Data Mart (using open source tools like Pentaho Data Integration Tool, Pentaho Business Analytics; or other data warehouse tools like Microsoft-SSIS, Informatica, Business Objects,etc.,)**

II. **Design multi-dimensional data models namely Star, Snowflake and Fact Constellation schemas for any one enterprise (ex. Banking, Insurance, Finance, Healthcare, manufacturing, Automobiles, sales etc).**

III. **Write ETL scripts and implement using data warehouse tools.**

IV. **Perform Various OLAP operations such slice, dice, roll up, drill up and pivot**

A. Build Data Warehouse/Daata Mart (using open source tools like Pentaho Data Integration Tool, Pentaho Business Analytics; or other data warehouse tools like Microsoft-SSIS,Informatica,Business Objects,etc.,)

A.(i) Identify source tables and populate sample data.

The data warehouse contains 4 **ta**bles:

1. Date dimension: contain**s** every single date from 2006 to 2016.
2. Customer dimension: co**n**tains 100 customers. To be simple we'll make **it** type 1 so we don't create a new row f**o**r each change.
3. Van dimension: contains 20 vans. To be simple we'll make it type 1 so **w**e don't create a new row for each change.
4. Hire fact table: contains **1**000 hire transactions since 1st Jan 2011. It is a **d**aily snapshot fact table so that every day w**e** insert 1000 rows into this fact table. So over t**i**me we can track the changes of total bill, **v**an charges, satnav income, etc.

Create the source tables and populate them

So now we are going to create t**h**e 3 tables in HireBase database: Customer, Va**n, a**nd Hire. Then we populate them.

And here is the script to create and populate them:

```
-- Create database create
database HireBase go
use HireBase go

-- Create customer table

if exists (select * from sys.tables where name = 'Customer')
drop table Customer
go
```

```
create table Customer
( CustomerId varchar(20) not null primary key,

 CustomerName varchar(30), DateOfBirth date, Town varchar(50),
 TelephoneNo varchar(30), DrivingLicenceNo varchar(30), Occupation varchar(30)
)
Go
-- Populate Customer truncate table Customer go
declare @i int, @si varchar(10), @startdate date set
@i = 1

while @i <= 100
begin

 set @si = right('0'+CONVERT(varchar(10), @i),2)
 insert into Customer
```

```
 ( CustomerId, CustomerName, DateOfBirth, Town, TelephoneNo, DrivingLicenceNo,
Occupation)
 values

 ( 'N'+@si, 'Customer'+@si, DATEADD(d,@i-1,'2000-01-01'), 'Town'+@si, 'Phone'+@si,
'Licence'+@si, 'Occupation'+@si)

 set @i = @i + 1
end
go

select * from Customer

-- Create Van table

if exists (select * from sys.tables where name = 'Van')
drop table Van
go

create table Van
( RegNo varchar(10) not null primary key,

 Make varchar(30), Model varchar(30), [Year] varchar(4),
 Colour varchar(20), CC int, Class varchar(10)
)
go

-- Populate Van table
truncate table Van
go
declare @i int, @si varchar(10) set
@i = 1

while @i <= 20
begin

 set @si = convert(varchar, @i)
 insert into Van

 ( RegNo, Make, Model, [Year], Colour, CC, Class)
 values
 ( 'Reg'+@si, 'Make'+@si, 'Model'+@si,

 case @i%4 when 0 then 2008 when  1 then 2009 when 2 then 2010 when 3 then 2011 end, case
 when @i%5<3 then 'White' else 'Black' end,
 case @i%3 when 0 then 2000 when 1 then 2500 when 2 then 3000 end,

 case @i%3 when 0 then 'Small' when 1 then 'Medium' when 2 then 'Large' end) set @i =
 @i + 1

end go

select * from Van
```

```
-- Create Hire table

if exists (select * from sys.tables where name = 'Hire')
drop table Hire
go

create table Hire

( HireId varchar(10) not null primary key,
 HireDate date not null,
 CustomerId varchar(20) not null,

 RegNo varchar(10), NoOfDays int, VanHire money, SatNavHire money,
 Insurance money, DamageWaiver money, TotalBill money
)

go

-- Populate Hire table
truncate table Hire
go

declare @i int, @si varchar(10), @DaysFrom1stJan int, @CustomerId int, @RegNo int, @mi int set @i =
1

while @i <= 1000
begin
 set @si = right('000'+convert(varchar(10), @i),4) -- string of i

 set @DaysFrom1stJan = (@i-1)%200 --The Hire Date is derived from i modulo 200 set
 @CustomerId = (@i-1)%100+1 --The CustomerId is derived from i modulo 100 set @RegNo =
 (@i-1)%20+1 --The Van RegNo is derived from i modulo 20
 set @mi = (@i-1)%3+1 --i modulo 3

 insert into Hire (HireId, HireDate, CustomerId, RegNo, NoOfDays, VanHire, SatNavHire, Insurance,
DamageWaiver, TotalBill)

 values ('H'+@si, DateAdd(d, @DaysFrom1stJan, '2011-01-01'),
 left('N0'+CONVERT(varchar(10),@CustomerId),3), 'Reg'+CONVERT(varchar(10), @RegNo), @mi,
 @mi*100, @mi*10, @mi*20, @mi*40, @mi*170)

 set @i += 1
end
go

select * from Hire
```

Create the Data Warehouse

So now we are going to create the 3 dimension tables and 1 fact table in the data warehouse: DimDate, DimCustomer, DimVan and FactHire. We are going to populate the 3 dimensions but we'll leave the fact table empty. The purpose of this article is to show how to populate the fact table using SSIS.

And then we do it. This is the script to create and populate those dim and fact tables:

```
-- Create the data warehouse
create database TopHireDW go use
TopHireDW go

-- Create Date Dimension

if exists (select * from sys.tables where name = 'DimDate')
drop table DimDate
go

create table DimDate
( DateKey int not null primary key,

 [Year] varchar(7), [Month] varchar(7), [Date] date, DateString varchar(10)) go

-- Populate Date Dimension
truncate table DimDate
go

declare @i int, @Date date, @StartDate date, @EndDate date, @DateKey int,
 @DateString varchar(10), @Year varchar(4),

 @Month varchar(7), @Date1 varchar(20) set
@StartDate = '2006-01-01'

set @EndDate = '2016-12-31' set
@Date = @StartDate

insert into DimDate (DateKey, [Year], [Month], [Date], DateString)
 values (0, 'Unknown', 'Unknown', '0001-01-01', 'Unknown') --The unknown row

while @Date <= @EndDate
begin
 set @DateString = convert(varchar(10), @Date, 20)

 set @DateKey = convert(int, replace(@DateString,'-','')) set
 @Year = left(@DateString,4)
 set @Month = left(@DateString, 7)

 insert into DimDate (DateKey, [Year], [Month], [Date], DateString)
 values (@DateKey, @Year, @Month, @Date, @DateString)

 set @Date = dateadd(d, 1, @Date)
end
go

select * from DimDate

-- Create Customer dimension

if exists (select * from sys.tables where name = 'DimCustomer')
drop table DimCustomer
```

```
go

create table DimCustomer

( CustomerKey int not null identity(1,1) primary key,
 CustomerId varchar(20) not null,
 CustomerName varchar(30), DateOfBirth date, Town varchar(50),
 TelephoneNo varchar(30), DrivingLicenceNo varchar(30), Occupation varchar(30)
)
go

insert into DimCustomer (CustomerId, CustomerName, DateOfBirth, Town, TelephoneNo,
 DrivingLicenceNo, Occupation)
select * from HireBase.dbo.Customer

select * from DimCustomer

-- Create Van dimension

if exists (select * from sys.tables where name = 'DimVan')
drop table DimVan
go

create table DimVan

( VanKey int not null identity(1,1) primary key,
 RegNo varchar(10) not null,
 Make varchar(30), Model varchar(30), [Year] varchar(4),
 Colour varchar(20), CC int, Class varchar(10)
)

go
insert into DimVan (RegNo, Make, Model, [Year], Colour, CC, Class)
select * from HireBase.dbo.Van
go

select * from DimVan

-- Create Hire fact table

if exists (select * from sys.tables where name = 'FactHire')
drop table FactHire
go

create table FactHire
( SnapshotDateKey int not null, --Daily periodic snapshot fact table
 HireDateKey int not null, CustomerKey int not null, VanKey int not null, --Dimension Keys HireId
 varchar(10) not null, --Degenerate Dimension
 NoOfDays int, VanHire money, SatNavHire money, Insurance
 money, DamageWaiver money, TotalBill money
)
go

select * from FactHire
```

A.(ii). Design multi-demesional data models namely Star, Snowflake and Fact Constellation schemas for any one enterprise (ex. Banking,Insurance, Finance, Healthcare, manufacturing, Automobiles,sales etc).

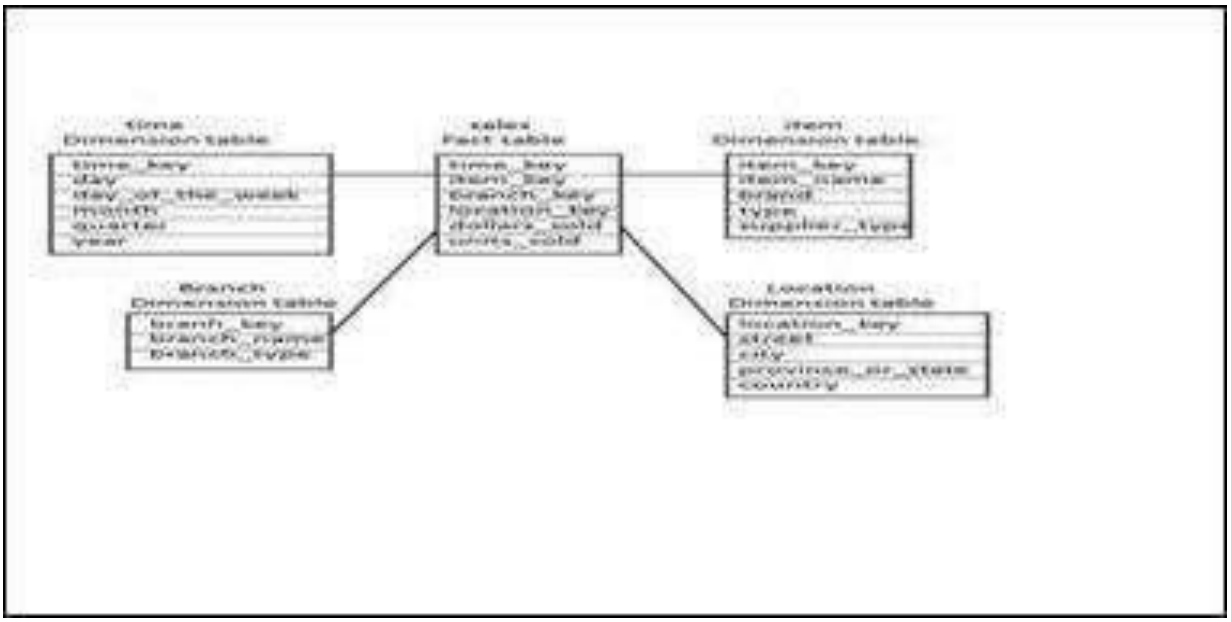
Ans:

Schema Definition

Multidimensional schema is defined using Data Mining Query Language (DMQL). The two primitives, cube definition and dimension definition, can be used for defining the data warehouses and data marts.

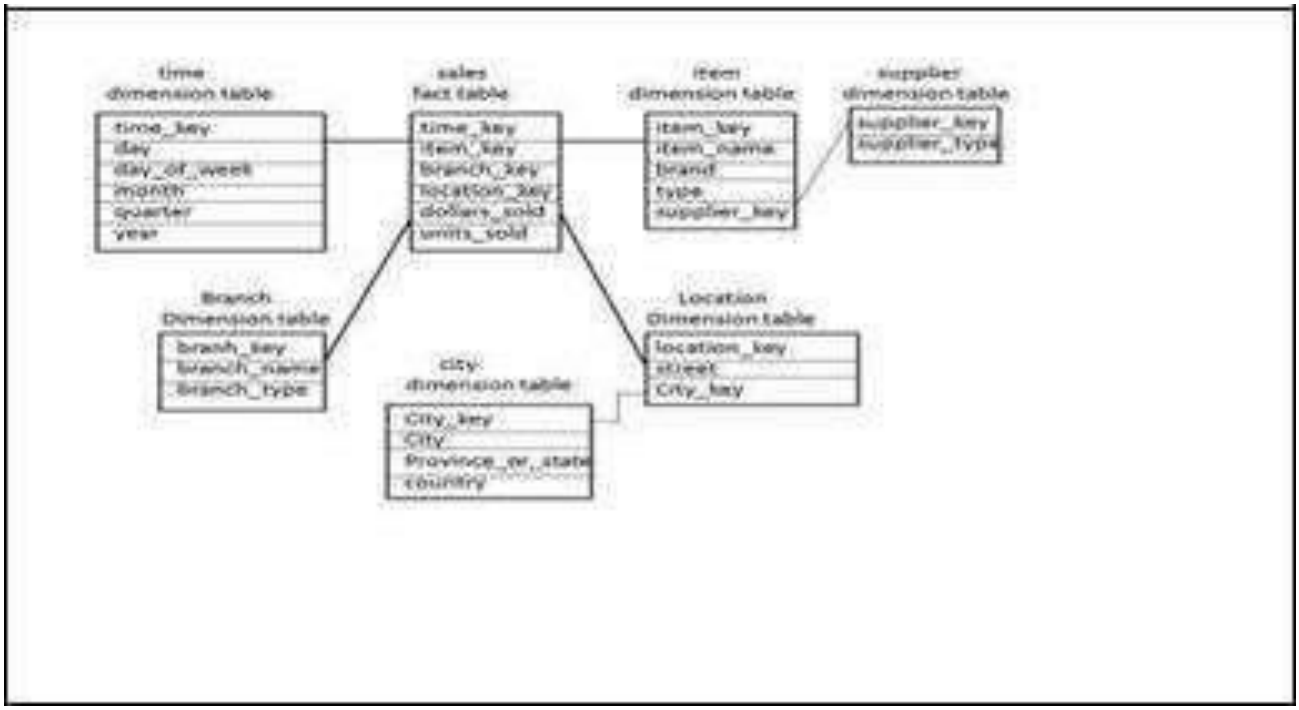
Star Schema

- Each dimension in a star schema is represented with only one-dimension table.
- This dimension table contains the set of attributes.
- The following diagram shows the sales data of a company with respect to the four dimensions, namely time, item, branch, and location.
- There is a fact table at the center. It contains the keys to each of four dimensions.
- The fact table also contains the attributes, namely dollars sold and units sold.



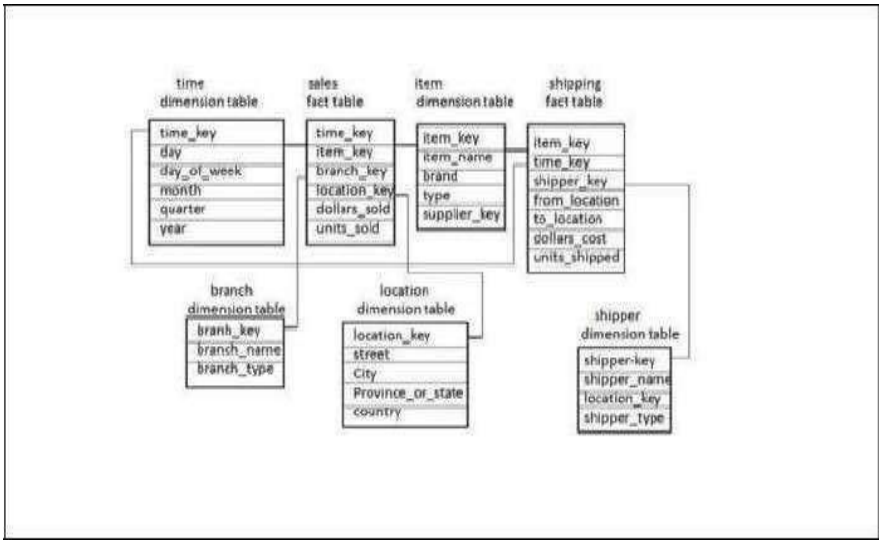
Snowflake Schema

- Some dimension tables in the Snowflake schema are normalized.
- The normalization splits up the data into additional tables.
- Unlike Star schema, the dimensions table in a snowflake schema is normalized. For example, the item dimension table in star schema is normalized and split into two dimension tables, namely item and supplier table.
- Now the item dimension table contains the attributes item_key, item_name, type, brand, and supplier-key.
- The supplier key is linked to the supplier dimension table. The supplier dimension table contains the attributes supplier_key and supplier_type.



Fact Constellation Schema

- fact constellation has multiple fact tables. It is also known as galaxy schema.
- The following diagram shows two fact tables, namely sales and shipping.
- The sales fact table is same as that in the star schema.
- The shipping fact table has the five dimensions, namely item_key, time_key, shipper_key, from_location, to_location.
- The shipping fact table also contains two measures, namely dollars sold and units sold.
- It is also possible to share dimension tables between fact tables. For example, time, item, and
- location dimension tables are shared between the sales and shipping fact table.



A.(iii) Write ETL scripts and implement using data warehouse tools.

Ans:

ETL comes from Data Warehousing and stands for Extract-Transform-Load. ETL covers a process of how the data are loaded from the source system to the data warehouse. Extraction– transformation– loading (ETL) tools are pieces of software responsible for the extraction of data from several sources, its cleansing, customization, reformatting, integration, and insertion into a data warehouse.

Building the ETL process is potentially one of the biggest tasks of building a warehouse; it is complex, time consuming, and consumes most of data warehouse project's implementation efforts, costs, and resources.

Building a data warehouse requires focusing closely on understanding three main areas:

1. Source Area- The source area has standard models such as entity relationship diagram.

2. Destination Area- The destination area has standard models such as star schema.

3. Mapping Area- But the mapping area has not a standard model till now.

Abbreviations

ETL-extraction–transformation–loading
DW-data warehouse
DM- data mart
OLAP- on-line analytical processing
DS-data sources
ODS- operational data store
DSA- data staging area
DBMS- database management system
OLTP-on-line transaction processing
CDC-change data capture
SCD-slowly changing dimension
FCME- first-class modeling elements
EMD-entity mapping diagram
DSA-data storage area

ETL Process:

Extract

The Extract step covers the data extraction from the source system and makes it accessible for further processing. The main objective of the extract step is to retrieve all the required data from the source system with as little resources as possible. The extract step should be designed in a way that it does not negatively affect the source system in terms or performance, response time or any kind of locking. There are several ways to perform the extract:

- Update notification - if the source system is able to provide a notification that a record has been changed and describe the change, this is the easiest way to get the data.
- Incremental extract - some systems may not be able to provide notification that an update has occurred, but they are able to identify which records have been modified and provide an extract of such records. During further ETL steps, the system needs to identify changes and propagate it down.

Note, that by using daily extract, we may not be able to handle deleted records properly

- Full extract - some systems are not able to identify which data has been changed at all, so a full extract is the only way one can get the data out of the system. The full extract requires keeping a copy of the last extract in the same format in order to be able to identify changes. Full extract handles deletions as well.

Transform

The transform step applies a set of rules to transform the data from the source to the target. This includes converting any measured data to the same dimension (i.e. conformed dimension) using the same units so that they can later be joined. The transformation step also requires joining data from several sources, generating aggregates, generating surrogate keys, sorting, deriving new calculated values, and applying advanced validation rules.

Load

During the load step, it is necessary to ensure that the load is performed correctly and with as little resources as possible. The target of the Load process is often a database. In order to make the load process efficient, it is helpful to disable any constraints and indexes before the load and enable them back only after the load completes. The referential integrity needs to be maintained by ETL tool to ensure consistency.

ETL method – nothin' but SQL

ETL as scripts that can just be run on the database.These scripts must be re-runnable: they should be able to be run without modification to pick up any changes in the legacy data, and automatically work out how to merge the changes into the new schema.

In order to meet the requirements, my scripts must:

1. INSERT rows in the new tables based on any data in the source that hasn't already been created in the destination

2. UPDATE rows in the new tables based on any data in the source that has already been inserted in the destination
3. DELETE rows in the new tables where the source data has been deleted

Now, instead of writing a whole lot of INSERT, UPDATE and DELETE statements, I thought "surely MERGE would be both faster and better" – and in fact, that has turned out to be the case. By writing all the transformations as MERGE statements, I've satisfied all the criteria, while also making my code very easily modified, updated, fixed and rerun. If I discover a bug or a change in requirements, I simply change the way the column is transformed in the MERGE statement, and re-run the statement. It then takes care of working out whether to insert, update or delete each row.

My next step was to design the architecture for my custom ETL solution. I went to the dba with the following design, which was approved and created for me:

1. create two new schemas on the new 11g database: LEGACY and MIGRATE
2. take a snapshot of all data in the legacy database, and load it as tables in the LEGACY schema
3. grant read-only on all tables in LEGACY to MIGRATE
4. Grant CRUD on all tables in the target schema to MIGRATE. For example, in the legacy database we have a table:

```
LEGACY.BMS_PARTIES(
 par_id       NUMBER          PRIMARY KEY,
 par_domain          VARCHAR2(10) NOT NULL,
 par_first_name     VARCHAR2(100) ,
```

```
  par_last_name       VARCHAR2(100),
  par_dob     DATE,
  par_business_name VARCHAR2(250),
  created_by
VARCHAR2(30) NOT NULL,
  creation_date
DATE          NOT NULL,        last_updated_by  VARCHAR2(30),
  last_update_date  DATE)
```

In the new model, we have a new table that represents the same kind of information:

```
NEW.TBMS_PARTY(
  party_id                NUMBER(9)       PRIMARY KEY,
  party_type_code  VARCHAR2(10) NOT NULL,
  first_name VARCHAR2(50),
  surname    VARCHAR2(100),
  date_of_birth         DATE,
  business_name       VARCHAR2(300),
  db_created_by       VARCHAR2(50) NOT NULL,
  db_created_on       DATE
DEFAULT SYSDATE NOT NULL,
  db_modified_by VARCHAR2(50),
  db_modified_on     DATE,
  version_id NUMBER(12)  DEFAULT 1 NOT NULL)
```

This was the simplest transformation you could possibly think of – the mapping from one to the other is 1:1, and the columns almost mean the same thing.

The solution scripts start by creating an intermediary table:

```
MIGRATE.TBMS_PARTY(
  old_par_id              NUMBER          PRIMARY KEY,
  party_id                NUMBER(9)       NOT NULL,
  party_type_code VARCHAR2(10) NOT NULL,
  first_name          VARCHAR2(50),
  surname             VARCHAR2(100),
  date_of_birth       DATE,
  business_name       VARCHAR2(300),
  db_created_by       VARCHAR2(50),
  db_created_on       DATE,
  db_modified_by      VARCHAR2(50),
  db_modified_on      DATE,
  deleted             CHAR(1))
```

The second step is the E and T parts of "ETL": I query the legacy table, transform the data right there in the query, and insert it into the intermediary table. However, since I want to be able to re-run this script as often as I want, I wrote this as a MERGE statement:

```
MERGE INTO MIGRATE.TBMS_PARTY dest

USING (
        CASE par_domain
          WHEN 'P' THEN 'PE' /*Person*/
        WHEN 'O' THEN 'BU' /*Business*/
        par_business_name AS business_name,
```

```
        SELECT par_id        AS old_par_id,  par_id  AS party_id,
               END
                    AS party_type_code,
               par_first_name AS first_name,
               par_last_name
                          AS surname,
               par_dob
         AS date_of_birth,
              created_by
      AS db_created_by,
              creation_date
       AS db_created_on,
              last_updated_by AS db_modified_by,
              last_update_date AS db_modified_on
        FROM
LEGACY.BMS_PARTIES s
        WHERE NOT EXISTS (          SELECT null        FROM MIGRATE.TBMS_PARTY d
          WHERE d.old_par_id = s.par_id
          AND
(d.db_modified_on = s.last_update_date          OR (d.db_modified_on IS NULL AND s.last_update_date IS


NULL))) ) src ON (src.OLD_PAR_ID = dest.OLD_PAR_ID)       WHEN MATCHED THEN UPDATE SET       party_id =

src.party_id ,          party_type_code = src.party_type_code ,      first_name = src.first_name,
        surname= src.surname,
        date_of_birth= src.date_of_birth ,
        business_name = src.business_name,
        db_created_by= src.db_created_by,
        db_created_on = src.db_created_on
        db_modified_by = src.db_modified_by ,
```

A.(iv) Perform Various OLAP operations such slice, dice, roll up, drill up and pivot.

Ans: **OLAP OPERATIONS**

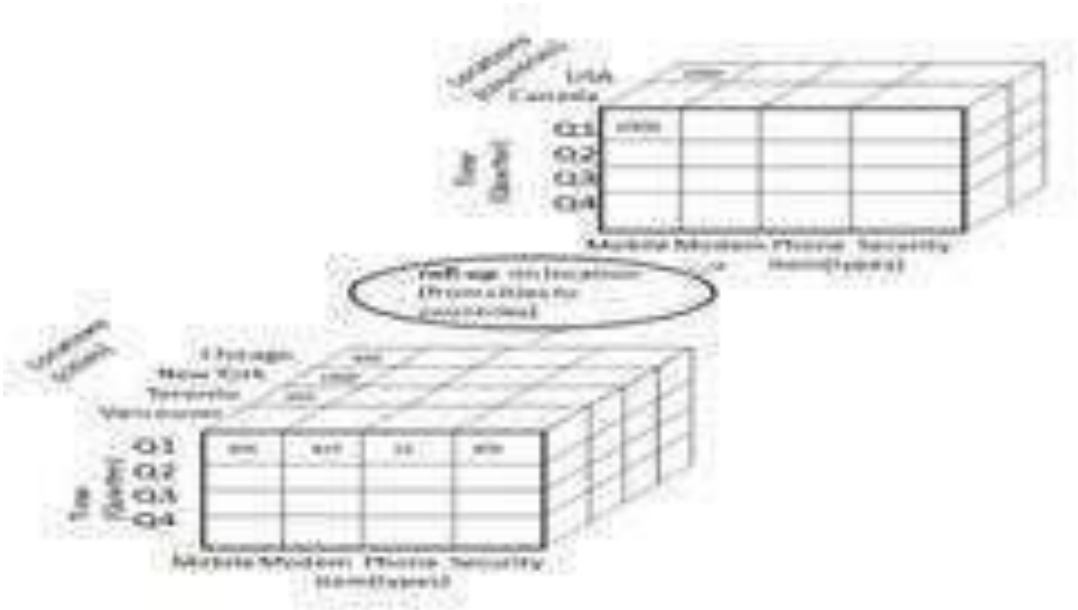
Online Analytical Processing Server (OLAP) is based on the multidimensional data model. It allows managers, and analysts to get an insight of the information through fast, consistent, and interactive access to information.

OLAP operations in multidimensional data.

Here is the list of OLAP operations:

- Roll-up
- Drill-down
- Slice and dice
- Pivot
- (rotate) Roll-up
- Roll-up performs aggregation on a data cube in any of the following ways:
  By climbing up a concept hierarchy for a dimension
- By dimension reduction

The following diagram illustrates how roll-up works.



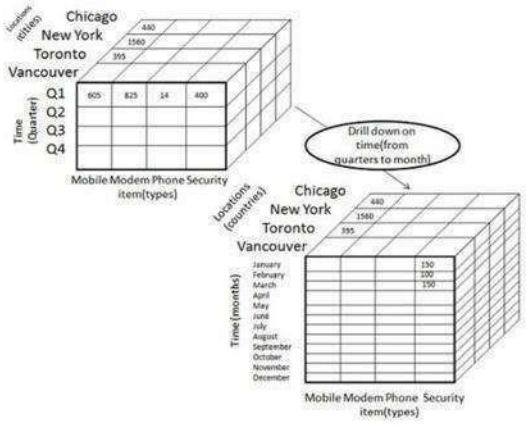
Roll-up is performed by climbing up a concept hierarchy for the dimension location.
Initially the concept hierarchy was "street < city < province < country".
On rolling up, the data is aggregated by ascending the location hierarchy from the level of city to the level of country.
The data is grouped into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

Drill-down

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

By stepping down a concept hierarchy for a dimension
By introducing a new dimension.

The following diagram illustrates how drill-down works:



Drill-down is performed by stepping down a concept hierarchy for the dimension time.

Initially the concept hierarchy was "day < month < quarter < year."

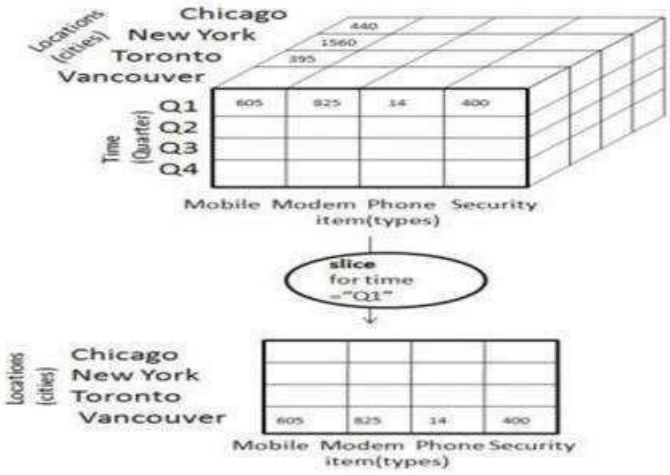
On drilling down, the time dimension is descended from the level of quarter to the level of month.

When drill-down is performed, one or more dimensions from the data cube are added.

It navigates the data from less detailed data to highly detailed data.

Slice

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



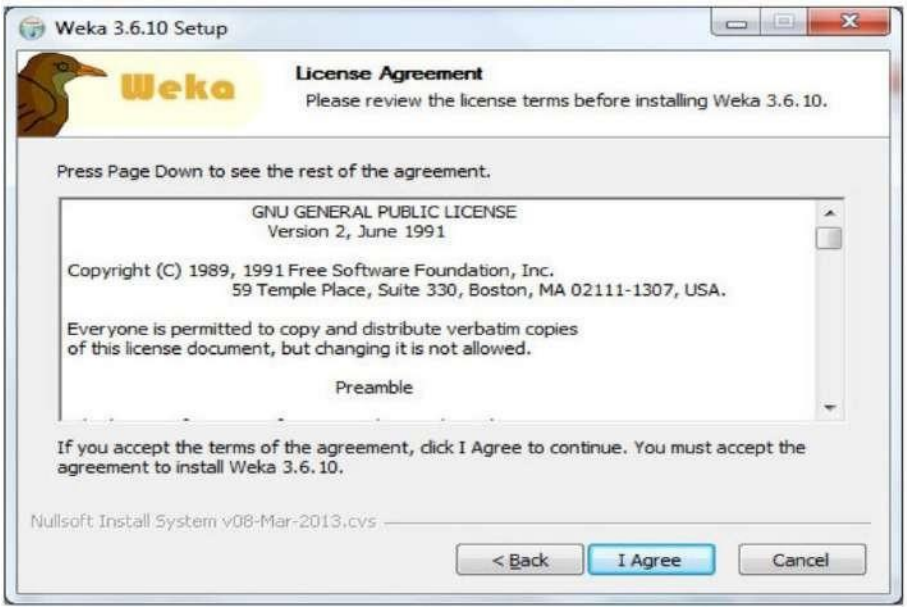
B.Explore WEKA Data Mining/Machine Learning Toolkit

B.(i) Downloading and/or installation of WEKA data mining toolkit.

Ans: Install Steps for WEKA a Data Mining Tool

1. Download the software as your requirements from the below given link.
   http://www.cs.waikato.ac.nz/ml/weka/downloading.html
2. The Java is mandatory for installation of WEKA so if you have already Java on your machine then download only WEKA else download the software with JVM.
3. Then open the file location and double click on the file
4. Click Next

5. Click I Agree.

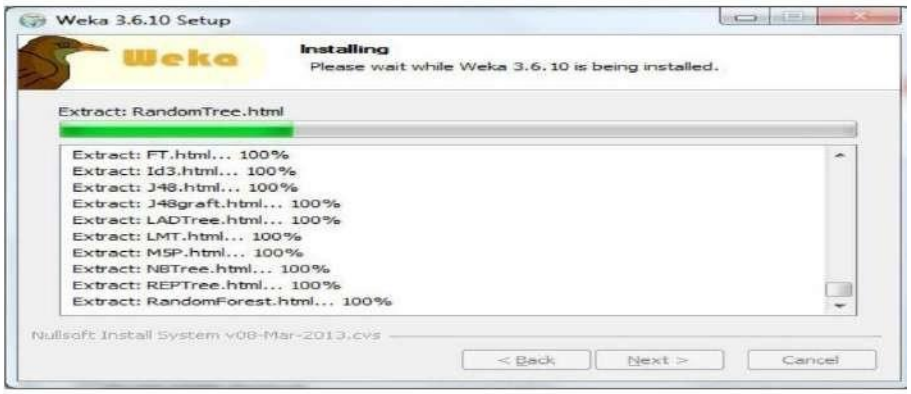


6. As your requirement do the necessary changes of settings and click Next. Full and Associate files are the recommended settings.

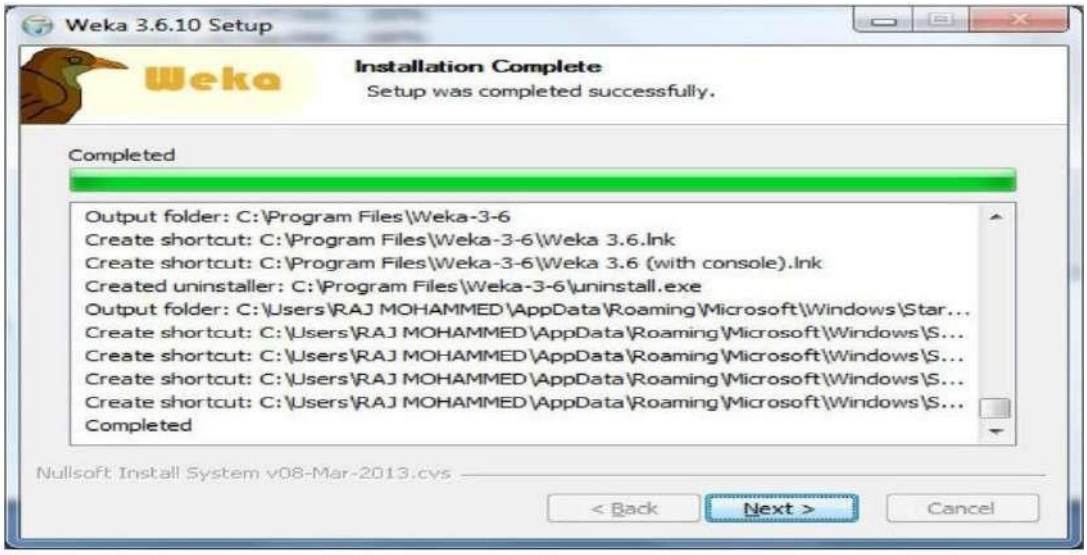
7. Change to your desire installation location.

8. If you want a shortcut then check the box and click Install.

9. The Installation will start wait for a while it will finish within a minute.



10. After complete installation click on Next.



11. Hurray !!!!!!! That's all click on the Finish and take a shovel and start Mining. Best of Luck.

This is the GUI you get when started. You have 4 options Explorer, Experimenter, KnowledgeFlow and Simple CLI.

B.(ii)Understand the features of WEKA tool kit such as Explorer, Knowledge flow interface, Experimenter, command-line interface.

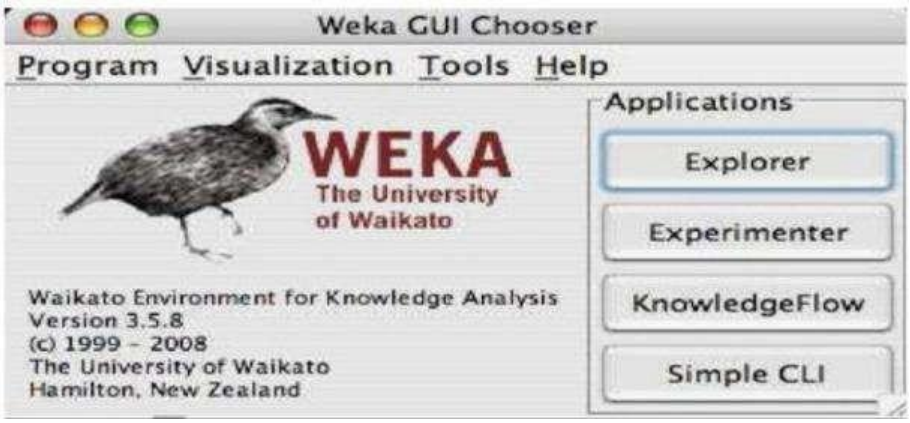
Ans: **WEKA**

Weka is created by researchers at the university WIKATO in New Zealand. University of Waikato, Hamilton, New Zealand Alex Seewald (original Command-line primer) David Scuse (original Experimenter tutorial)

- It is java based application.
- It is collection often source, Machine Learning Algorithm.
- The routines (functions) are implemented as classes and logically arranged in packages.
- It comes with an extensive GUI Interface.
- Weka routines can be used standalone via the command line interface.

The Graphical User Interface;-

The Weka GUI Chooser (class weka.gui.GUIChooser) provides a starting point for launching Weka's main GUI applications and supporting tools. If one prefers a MDI ("multiple document interface") appearance, then this is provided by an alternative launcher called "Main"(class weka.gui.Main). The GUI Chooser consists of four buttons—one for each of the four major Weka applications—and four menus.



The buttons can be used to start the following applications:

1. **Explorer An environment** for exploring data with WEKA (the rest of this Documentation deals with this application in more detail).
2. **Experimenter** An environment for performing experiments and conducting statistical tests between learning schemes.
3. **Knowledge Flow** This environment supports essentially the same functions as the Explorer but with a drag-and-drop interface. One advantage is that it supports incremental learning.
4. **SimpleCLI Provides** a simple command-line interface that allows direct execution of WEKA commands for operating systems that do not provide their own command line interface.
5. **Explorer** The Graphical user interface

Section Tabs

At the very top of the window, just below the title bar, is a row of tabs. When the Explorer is first started only the first tab is active; the others are grayed out. This is because it is necessary to open (and potentially pre-process) a data set before starting to explore the data.

The tabs are as follows:

1. **Preprocess.** Choose and modify the data being acted on.

2. **Classify.** Train & test learning schemes that classify or perform regression

3. **Cluster.** Learn clusters for the data.

4. **Associate.** Learn association rules for the data.

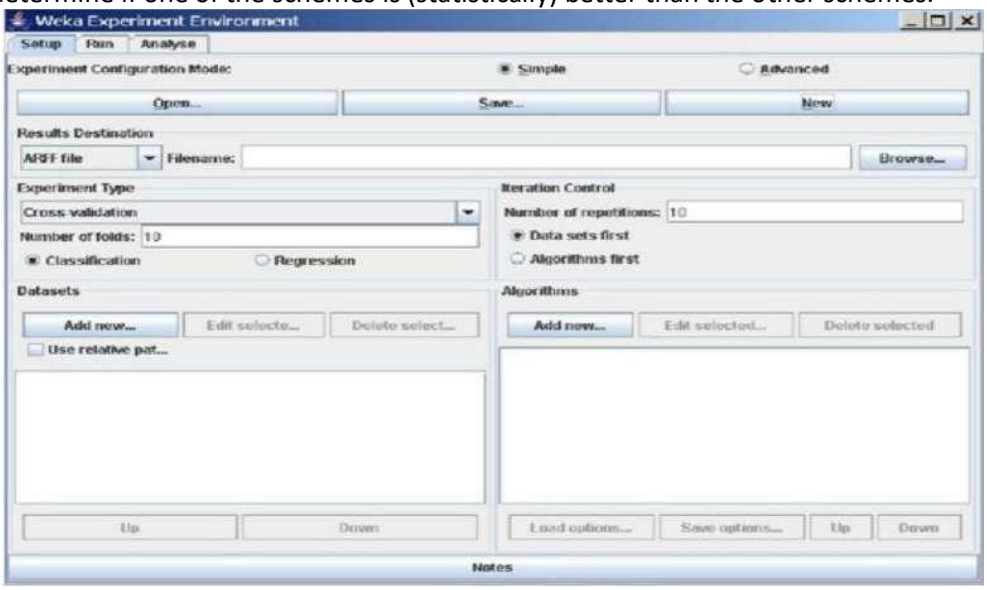
5. **Select attributes.** Select the most relevant attributes in the data.

6. **Visualize.** View an interactive 2D plot of the data.

Once the tabs are active, clicking on them flicks between different screens, on which the respective actions can be performed. The bottom area of the window (including the status box, the log button, and the Weka bird) stays visible regardless of which section you are in. The Explorer can be easily extended with custom tabs. The Wiki article **"Adding tabs in the Explorer" explains this in detail.**

2. Weka Experimenter:-

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.



The Experiment Environment can be run from the command line using the Simple CLI. For example, the following commands could be typed into the CLI to run the OneR scheme on the Iris dataset using a basic train and test process. (Note that the commands would be typed on one line into the CLI.) While commands can be typed directly into the CLI, this technique is not particularly convenient and the experiments are not easy to modify. The Experimenter comes in two flavors', either with a simple interface that provides most of the functionality one needs for experiments, or with an interface **with full access to the Experimenter's capabilities. You can** choose between those two with the Experiment Configuration Mode radio buttons:

1. Simple
2. Advanced

Both setups allow you to setup standard experiments that are run locally on a single machine, or remote experiments, which are distributed between several hosts. The distribution of experiments cuts down the time the experiments will take until completion, but on the other hand the setup takes more time. The next section covers the standard experiments (both, simple and advanced), followed by the remote experiments and finally the analyzing of the results.

Knowledge Flow

Introduction

The Knowledge Flow provides an alternative to the Explorer as a graphical front end to **WEKA's core algorithms.**

The Knowledge Flow presents a data-flow inspired interface to WEKA. The user can select WEKA components from a palette, place them on a layout canvas and connect them together in order to form a knowledge flow for processing and analyzing data. At present, all of **WEKA's classifiers, filters, clusterers, associators, loaders and savers a**re available in the Knowledge Flow along with some extra tools.

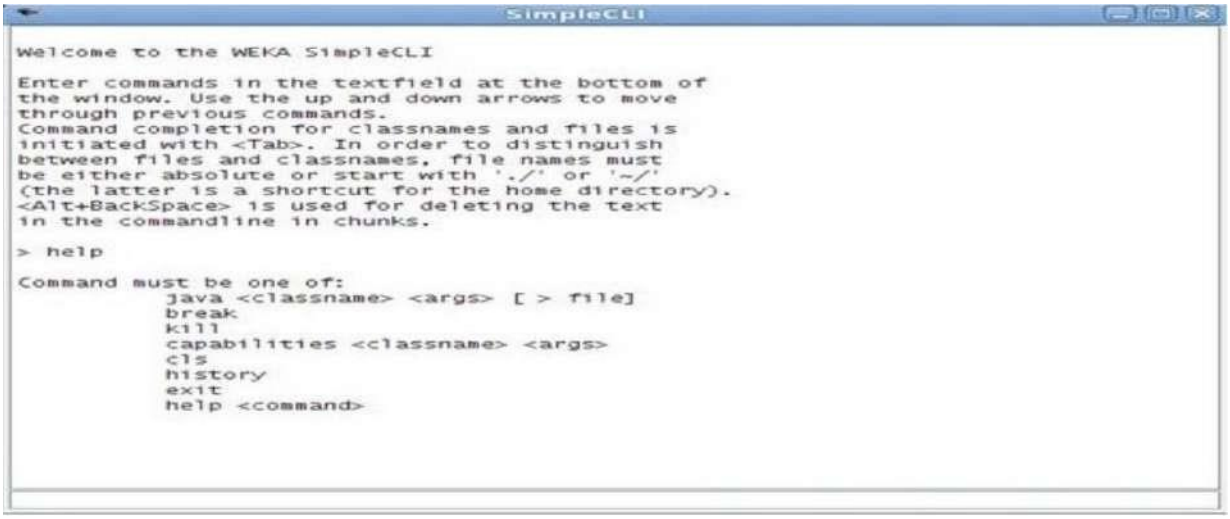
The Knowledge Flow can handle data either incrementally or in batches (the Explorer handles batch data only). Of course learning from data incremen- tally requires a classifier that can be updated on an instance by instance basis. Currently in WEKA there are ten classifiers that can handle data incrementally.

The Knowledge Flow offers the following features:

1. **Intuitive** data flow style layout.
2. **Process** data in batches or incrementally
3. **Process multiple batches** or streams in parallel (each separate flow executes in its own thread).
4. **Process multiple streams sequentially** via a user-specified order of execution.
5. **Chain filters** together.
6. **View models** produced by classifiers for each fold in a cross validation.
7. **Visualize performance** of incremental classifiers during processing (scrolling plots of classification accuracy, RMS error, predictions etc.).
8. **Plugin "perspectives" that add major new functionality (e.g. 3D data** visualization, time series forecasting environment etc.).

4. Simple CLI

The Simple CLI provides full access to all Weka classes, i.e., classifiers, filters, clusterers, etc., but without the hassle of the CLASSPATH (it facilitates the one, with which Weka was started). It offers a simple Weka shell with separated command line and output.



Commands

The following commands are available in the Simple CLI:

- **Java <classname> [<args>]**
  Invokes a java class with the given arguments (if any).

- **Break**

Stops the current thread, e.g., a running classifier, in a friendly manner kill stops the current thread in an unfriendly fashion.

- **Cls**
  Clears the output area

- Capabilities <classname> [<args>]
  Lists the capabilities of the specified class, e.g., for a classifier with its.

- **option:**
  Capabilities weka.classifiers.meta.Bagging -W weka.classifiers.trees.Id3

- **exit**
  Exits the Simple CLI

- **help [<command>]**

  Provides an overview of the available commands if without a command name as argument, otherwise more help on the specified command

Invocation

In order to invoke a Weka class, one has only to prefix the class with "java". This command tells the Simple CLI to load a class and execute it with any given parameters. E.g., the J48 classifier can be invoked on the iris dataset with the following command:

java weka.classifiers.trees.J48 -t c:/temp/iris.arff

Command redirection

Starting with this version of Weka one can perform a basic redirection: java weka.classifiers.trees.J48 -t test.arff > j48.txt

Note: the > must be preceded and followed by a space, otherwise it is not recognized as redirection, but part of another parameter.

Command completion

Commands starting with java support completion for classnames and filenames via Tab (Alt+BackSpace deletes parts of the command again). In case that there are several matches, Weka lists all possible matches.

- **Package Name Completion java weka.cl<Tab>**

Results in the following output of possible matches of
package names: Possible matches:
weka.classifiers
weka.clusterers

- **Classname completion**

java weka.classifiers.meta.A<Tab> lists the following classes

Possible matches:
weka.classifiers.meta.AdaBoostM1
weka.classifiers.meta.AdditiveRegression
weka.classifiers.meta.AttributeSelectedClassifier

- **Filename Completion**

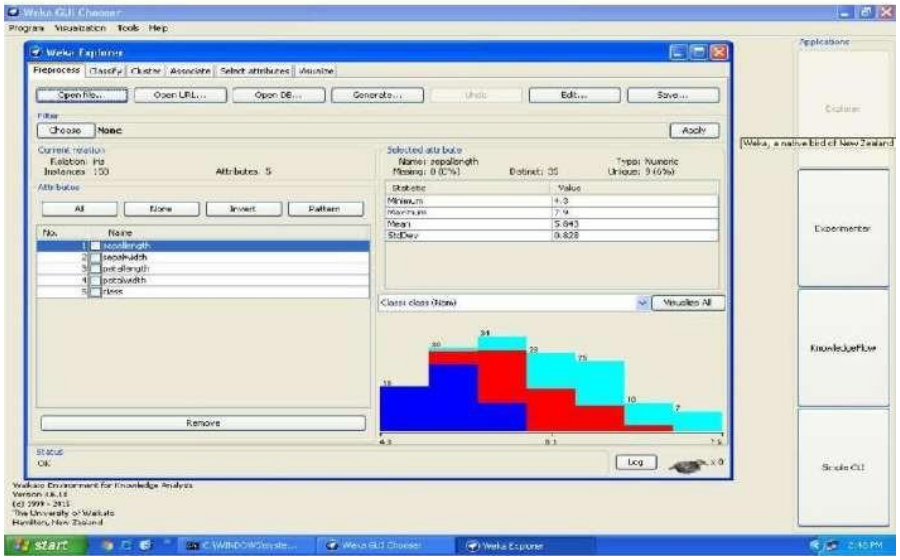
In order for Weka to determine whether a the string under the cursor is a classname or a filename, filenames need to be absolute (Unix/Linx: /some/path/file;Windows: C:\Some\Path\file) or relative and starting with a dot (Unix/Linux:./some/other/path/file; Windows:

.\Some\Other\Path\file).

B.(iii)Navigate the options available in the WEKA(ex.select attributes panel,preprocess panel,classify panel,cluster panel,associate panel and visualize)

Ans: Steps for identify options in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose iris data set and open file.
8. All tabs available in WEKA home page.



B. (iv) Study the ARFF file format

Ans:

ARFF File Format

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 are not the only format one can load, but all files that can be converted with **Weka's "core converters". The following formats are currently** supported:

- ARFF (+ compressed)
- C4.5
- CSV
- Libsvm
- binary serialized instances
- XRFF (+ compressed)

Overview

ARFF files have two distinct sections. The first section is the **Header** information, which is followed the **Data** information. The Header of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types.

An example header on the standard IRIS dataset looks like this:

1. **Title: Iris Plants Database**

2. **Sources:**
   (a) Creator: R.A. Fisher
   (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
   (c) Date: July, 1988

@RELATION iris
@ATTRIBUTE sepal length NUMERIC
@ATTRIBUTE sepal width NUMERIC
@ATTRIBUTE petal length NUMERIC
@ATTRIBUTE petal width NUMERIC
@ATTRIBUTE class {Iris-setosa, Iris-versicolor, Iris-irginica} The Data of the ARFF file looks like the following:

@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
5.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa

Lines that begin with a % are comments.
The @RELATION, @ATTRIBUTE and @DATA declarations are case insensitive.

The ARFF Header Section

The ARFF Header section of the file contains the relation declaration and at-tribute declarations.

The @relation Declaration

The relation name is defined as the first line in the ARFF file. The format is: @relation <relation-name>

where <relation-name> is a string. The string must be quoted if the name includes spaces.

The @attribute Declarations

Attribute declarations take the form of an ordered sequence of @attribute statements. Each attribute in the data set has its own @attribute statement which uniquely defines the name **ofthat attribute and it's data type. The order** the attributes are declared indicates the column position in the data section of the file. For example, if an attribute is the third one declared then Weka expects that all that attributes values will be found in the third comma delimited column.

The format for the @attribute statement is:

@attribute <attribute-name> <datatype>

where the <attribute-name> must start with an alphabetic character. If spaces are to be included in the name then the entire name must be quoted.

The <datatype> can be any of the four types supported by Weka:

1. numeric
2. integer is treated as numeric
3. real is treated as numeric
4. <nominal-specification>
5. string
6. date [<date-format>]

relational for multi-instance data (for future use) where <nominal-specification> and <date-format> are defined below. The keywords numeric, real, integer, string and date are case insensitive.

Numeric attributes

Numeric attributes can be real or integer numbers.

Nominal attributes

Nominal values are defined by providing an <nominal-specification> listing the possible values: <nominal-name1>, <nominal-name2>, <nominal-name3>,

For example, the class value of the Iris dataset can be defined as follows: @ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica} Values that contain spaces must be quoted.

String attributes

String attributes allow us to create attributes containing arbitrary textual values. This is very useful in text-mining applications, as we can create datasets with string attributes, then write Weka Filters to manipulate strings (like String- ToWordVectorFilter). String attributes are declared as follows:

@ATTRIBUTE LCC string

Date attributes

Date attribute declarations take the form: @attribute <name> date [<date-format>] where <name> is the name for the attribute and <date-format> is an optional string specifying how date values should be parsed and printed (this is the same format used by SimpleDateFormat). The default format string accepts

the ISO-8601 combined date and time format: yyyy-MM-**dd'T'HH:mm:ss.** Dates must be specified in the data section as the corresponding string representations of the date/time (see example below).

Relational attributes

Relational attribute declarations take the form: @attribute <name> relational
<further attribute definitions> @end <name>
For the multi-instance dataset MUSK1 the definition would look like thi**s ("..."** denotes an omission):
@attribute molecule_name {MUSK-jf78,...,NON-MUSK-199} @attribute bag relational
@attribute f1 numeric
...
@attribute f166 numeric @end bag
@attribute class {0,1}

The ARFF Data Section

The ARFF Data section of the file contains the data declaration line and the actual instance lines.

The @data Declaration

The @data declaration is a single line denoting the start of the data segment in the file. The format is:

@data

The instance data

Each instance is represented on a single line, with carriage returns denoting the end of the instance. A percent sign (%) introduces a comment, which continues to the end of the line.

Attribute values for each instance are delimited by commas. They must appear in the order that they were declared in the header section (i.e. the data corresponding to the nth @attribute declaration is always the nth field of the attribute).

Missing values are represented by a single question mark, as in:

@data 4.4,?,1.5,?,Iris-setosa

Values of string and nominal attributes are case sensitive, and any that contain space or the comment-delimiter character % must be quoted. (The code suggests that double-quotes are acceptable and that a backslash will escape individual characters.)

An example follows: @relation LCCvsLCSH @attribute LCC string @attribute LCSH string
@data

AG5, 'Encyclopedias and dictionaries.;Twentieth century.' AS262, 'Science -- Soviet Union -- **History.'**
AE5, 'Encyclopedias and dictionaries.'
AS281, 'Astronomy, Assyro-Babylonian.;Moon -- **Phases.'**
AS28**1, 'Astronomy, Assyro**-Babylonian.;Moon -- **Tables.'**

Dates must be specified in the data section using the string representation specified in the attribute declaration.

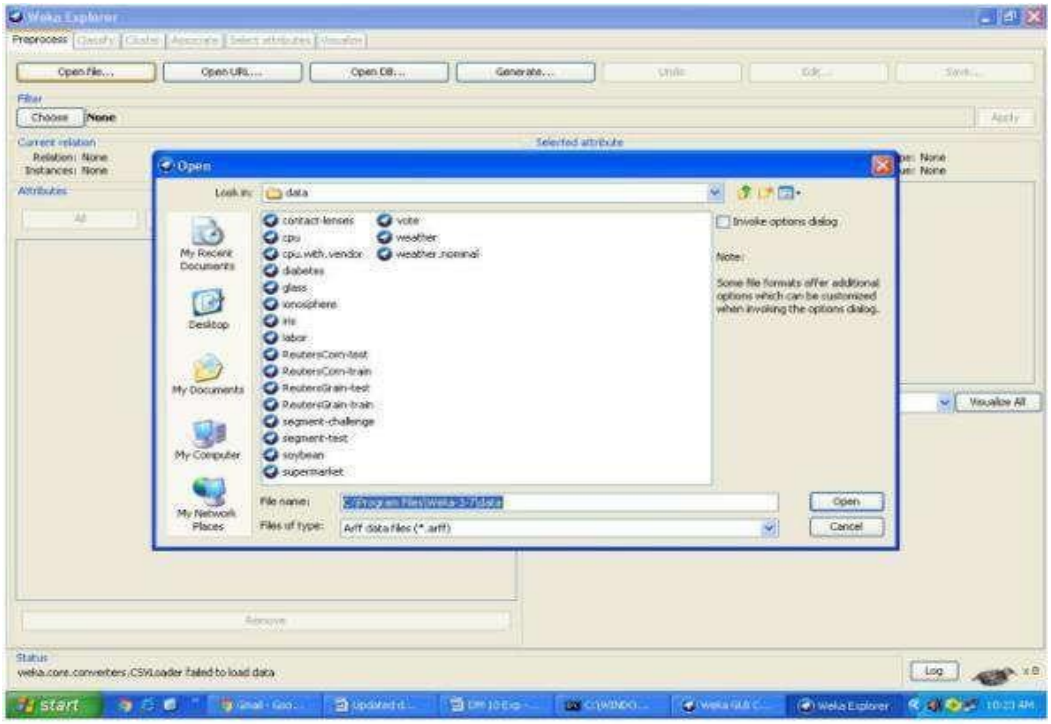
For example: @RELATION
Timestamps
@ATTRIBUTE timestamp DATE "yyyy-MM-dd HH:mm:ss" @DATA
"2001-04-03 12:12:12"
"2001-05-03 12:59:55"

Relational data must be enclosed within double quotes ". For example an instance of the MUSK1 **dataset ("..." denotes an omission):**
MUSK-188,"42,...,30",1

B.(v) Explore the available data sets in WEKA.

Ans: Steps for identifying data sets in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on open file button.
4. Choose WEKA folder in C drive.
5. Select and Click on data option button.



<u>Sample Weka Data Sets</u>

Below are some sample WEKA data sets, in arff format.

- contact-lens.arff
- cpu.arff
- cpu.with-vendor.arff
- diabetes.arff
- glass.arff
- ionospehre.arff
- iris.arff
- labor.arff
- ReutersCorn-train.arff
- ReutersCorn-test.arff
- ReutersGrain-train.arff
- ReutersGrain-test.arff
- segment-challenge.arff
- segment-test.arff
- soybean.arff
- supermarket.arff
- vote.arff
- weather.arff
- weather.nominal.arff

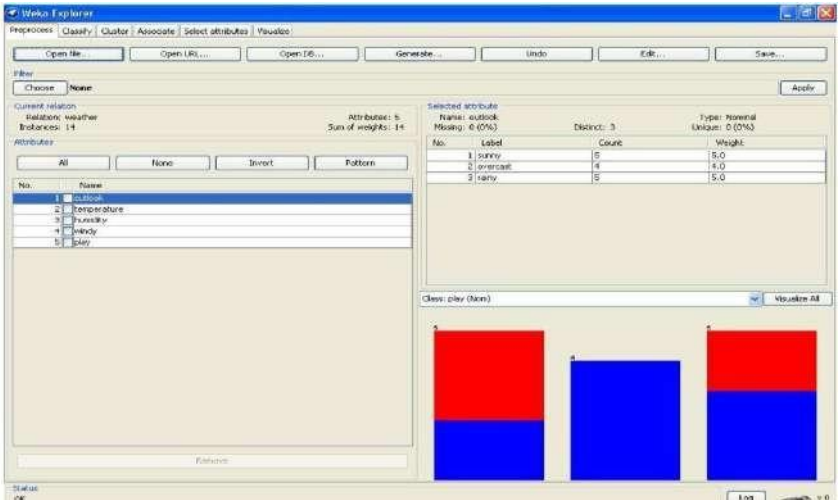
B. (vi) Load a data set (ex.Weather dataset,Iris dataset,etc.)

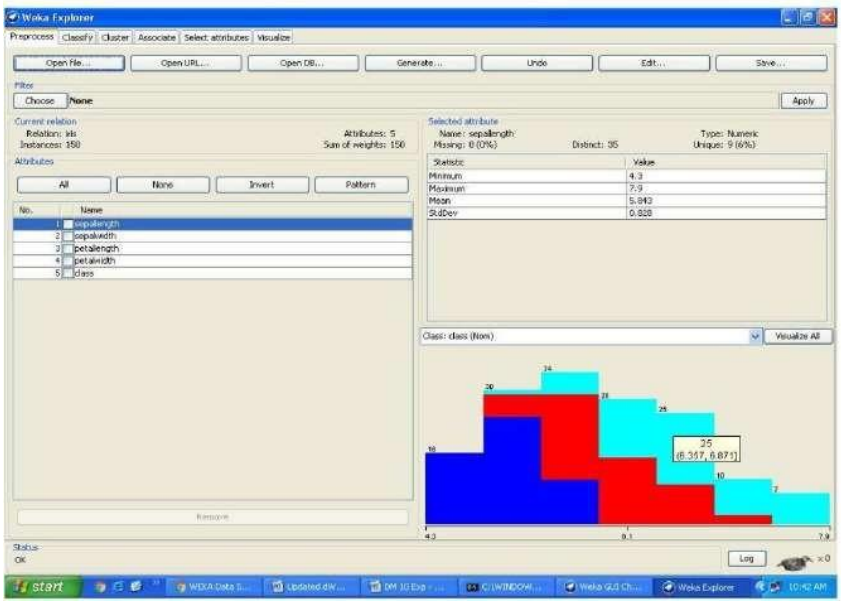
Ans: Steps for load the Weather data set.

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on open file button.
4. Choose WEKA folder in C drive.
5. Select and Click on data option button.
6. Choose Weather.arff file and Open the file.

Steps for load the Iris data set.

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on open file button.
4. Choose WEKA folder in C drive.
5. Select and Click on data option button.
6. Choose Iris.arff file and Open the file.





B. (vii) Load each dataset and observe the following:

(vii.i) List attribute names and they types

Ans: Example dataset-Weather.arff

List out the attribute names:

1. outlook
2. temperature
3. humidity
4. windy
5. play

B. (vii.ii) Number of records in each dataset. Ans:

@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

B. (vii.iii) Identify the class attribute (if any)

Ans: class attributes

1. sunny
2. overcast
3. rainy

B. (vii.iv) Plot Histogram

Ans: Steps for identify the plot histogram

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Visualize button.
4. Click on right click button.
5. Select and Click on polyline option button.

B. (vii.v) Determine the number of records for each class

Ans: @relation weather.symbolic
@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

B. (vii.vi) Visualize the data in various dimensions

Click on Visualize All button in WEKA Explorer.

Unit-II Perform data preprocessing tasks and Demonstrate performing association rule mining on data sets

A. Explore various options in Weka for Preprocessing data and apply (like Discretization Filters, Resample filter, etc.) n each dataset.
Ans:

Preprocess Tab

1. Loading Data

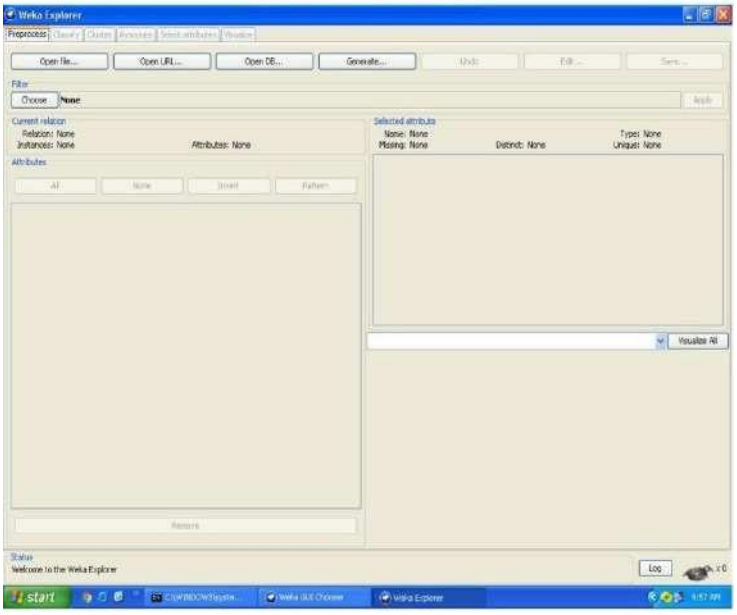
The first four buttons at the top of the preprocess section enable you to load data into WEKA:

1. **Open file....** Brings up a dialog box allowing you to browse for the data file on the local file system.

2. **Open URL....** Asks for a Uniform Resource Locator address for where the data is stored.

3. **Open DB ....**Reads data from a database. (Note that to make this work you might have to edit the file in weka/experiment/DatabaseUtils.props.)

4. **Generate....** Enables you to generate artificial data from a variety of Data Generators. Using the Open file... button you can read files in a variety of formats: **WEKA's ARFF format, CSV** format, C4.5 format, or serialized Instances format. ARFF files typically have a .arff extension, CSV files a .csv extension, C4.5 files a .data and .names extension, and serialized Instances objects a .bsi extension.



Current Relation: Once some data has been loaded, the Preprocess panel shows a variety of in**formation. The Current relation box (the "current relation" is the** currently loaded data, which can be interpreted as a single relational table in database terminology) has three entries:

1. **Relation.** The name of the relation, as given in the file it was loaded from. Filters (described below) modify the name of a relation.

2. **Instances.** The number of instances (data points/records) in the data.

3. **Attributes.** The number of attributes (features) in the data.

Working with Attributes

Below the Current relation box is a box titled Attributes. There are four buttons, and beneath them is a list of the attributes in the current relation.

The list has three columns:

1. **No.** A number that identifies the attribute in the order they are specified in the data file.
2. **Selection tick boxes**. These allow you select which attributes are present in the relation.
3. **Name.** The name of the attribute, as it was declared in the data file. When you click on different rows in the list of attributes, the fields change in the box to the right titled selected attribute.

This box displays the characteristics of the currently highlighted attribute in the list:

1. **Name.** The name of the attribute, the same as that given in the attribute list.
2. **Type.** The type of attribute, most commonly Nominal or Numeric.
3. **Missing.** The number (and percentage) of instances in the data for which this attribute is missing (unspecified).
4. **Distinct.** The number of different values that the data contains for this attribute.
5. **Unique.** The number (and percentage) of instances in the data having a value for this attribute that no other instances have.

Below these statistics is a list showing more information about the values stored in this attribute, which differ depending on its type. If the attribute is nominal, the list consists of each possible value for the attribute along with the number of instances that have that value. If the attribute is numeric, the list gives four statistics describing the distribution of values in the data— the minimum, maximum, mean and standard deviation. And below these statistics there is a coloured histogram, colour-coded according to the attribute chosen as the Class using the box above the histogram. (This box will bring up a drop-down list of available selections when clicked.) Note that only nominal Class attributes will result in a colour-coding. Finally, after pressing the Visualize All button, histograms for all the attributes in the data are shown in a separate window.

Returning to the attribute list, to begin with all the tick boxes are unticked.They can be toggled on/off by clicking on them individually. The four buttons above can also be used to change the selection:

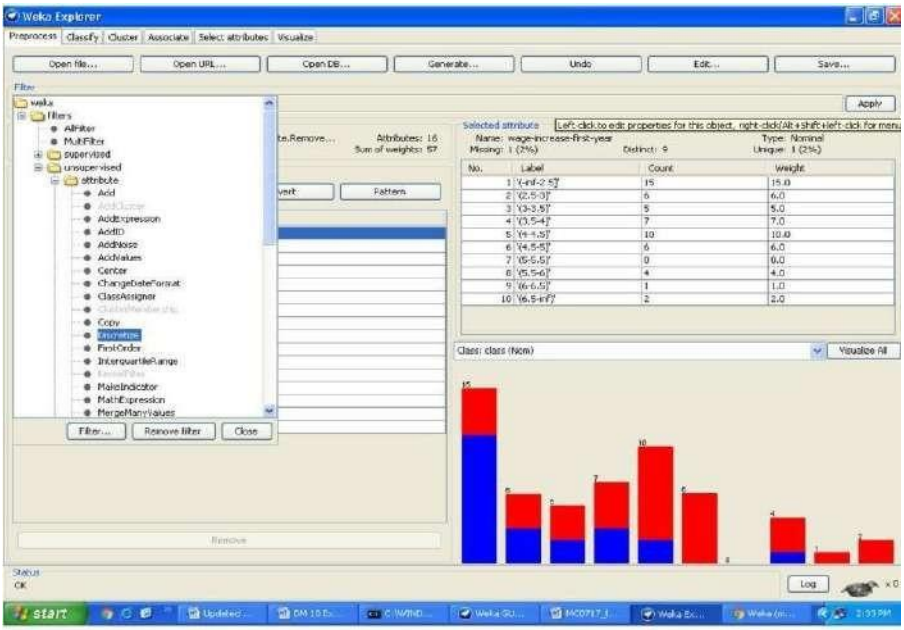
PREPROCESSING

1. **All.** All boxes are ticked.
2. **None.** All boxes are cleared (unticked).
3. **Invert.** Boxes that are ticked become unticked and vice versa.
4. **Pattern.** Enables the user to select attributes based on a Perl 5 Regular Expression. E.g., .* id selects all attributes which name ends with id.

Once the desired attributes have been selected, they can be removed by clicking the Remove button below the list of attributes. Note that this can be undone by clicking the Undo button, which is located next to the Edit button in the top-right corner of the Preprocess panel.

Working with Filters:-

The preprocess section allows filters to be defined that transform the data in various ways. The Filter box is used to set up the filters that are required. At the left of the Filter box is a Choose button. By clicking this button it is possible to select one of the filters in WEKA. Once a filter has been selected, its name and options are shown in the field next to the Choose button. Clicking on this box with the left mouse button brings up a GenericObjectEditor dialog box. A click with the right mouse button (or Alt+Shift+left click) brings up a menu where you can choose, either to display the properties in a GenericObjectEditor dialog box, or to copy the current setup string to the clipboard.



The GenericObjectEditor Dialog Box

The GenericObjectEditor dialog box lets you configure a filter. The same kind of dialog box is used to configure other objects, such as classifiers and clusterers. The fields in the window reflect the available options. Right-clicking (or Alt+Shift+Left-Click) on such a field will bring up a popup menu, listing the following options:

1. **Show properties...** has the same effect as left-clicking on the field, i.e., a dialog appears allowing you to alter the settings.
2. **Copy configuration** to clipboard copies the currently displayed configuration string to the **system's clipboar**d and therefore can be used anywhere else in WEKA or in the console. This is rather handy if you have to setup complicated, nested schemes.
3. **Enter configuration... is the "receiving" end for configurations that** got copied to the clipboard earlier on. In this dialog you can enter a class name followed by options (if the class supports these). This also allows you to transfer a filter setting from the Preprocess panel to a Filtered Classifier used in the Classify panel.

Left-Clicking on any of these gives an opportunity to alter the filters settings. For example, the setting may take a text string, in which case you type the string into the text field provided. Or it may give a drop-down box listing several states to choose from. Or it may do something else, depending on the information required. Information on the options is provided in a tool tip if you let the mouse pointer hover of the corresponding field. More information on the filter and its options can be obtained by clicking on the More button in the About panel at the top of the GenericObjectEditor window.

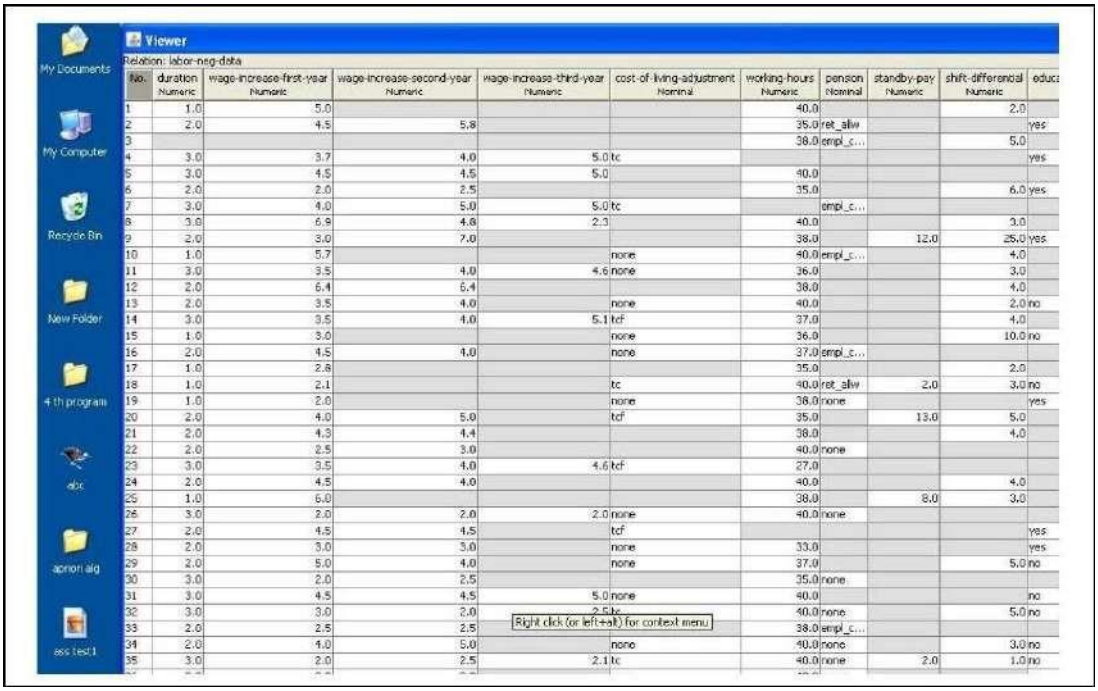
Applying Filters

Once you have selected and configured a filter, you can apply it to the data by pressing the Apply button at the right end of the Filter panel in the Preprocess panel. The Preprocess panel will then show the transformed data. The change can be undone by pressing the Undo button. You can also use the Edit...button to modify your data manually in a dataset editor. Finally, the Save...button at the top right of the Preprocess panel saves the current version of the relation in file formats that can represent the relation, allowing it to be kept for future use.

Steps for run preprocessing tab in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose labor data set and open file.
8. Choose filter button and select the Unsupervised-Discritize option and apply

Dataset labor.arff



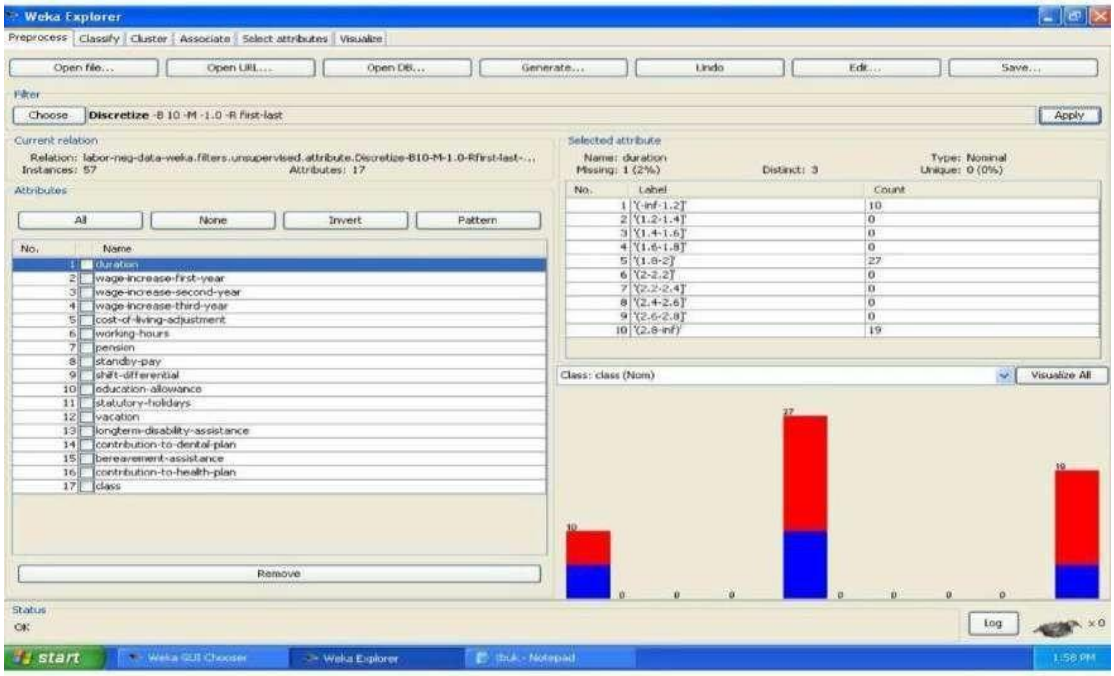
Viewer

Relation: labor-neg-data

No.	duration Numeric	wage-increase-first-year Numeric	wage-increase-second-year Numeric	wage-increase-third-year Numeric	cost-of-living-adjustment Nominal	working-hours Numeric	pension Nominal	standby-pay Numeric	shift-differential Numeric	educ
1	1.0	5.0				40.0			2.0	
2	2.0	4.5	5.8			35.0	ret_allw			yes
3						38.0	empl_c...		5.0	
4	3.0	3.7	4.0	5.0	tc					yes
5	3.0	4.5	4.5	5.0		40.0				
6	2.0	2.0	2.5			35.0			6.0	yes
7	3.0	4.0	5.0	5.0	tc		empl_c...			
8	3.0	6.9	4.8	2.3		40.0			3.0	
9	2.0	3.0	7.0			38.0		12.0	25.0	yes
10	1.0	5.7			none	40.0	empl_c...		4.0	
11	3.0	3.5	4.0	4.6	none	36.0			3.0	
12	2.0	6.4	6.4			38.0			4.0	
13	2.0	3.5	4.0		none	40.0			2.0	no
14	3.0	3.5	4.0	5.1	tcf	37.0			4.0	
15	1.0	3.0			none	36.0			10.0	no
16	2.0	4.5	4.0		none	37.0	empl_c...			
17	1.0	2.8				35.0			2.0	
18	1.0	2.1			tc	40.0	ret_allw	2.0	3.0	no
19	1.0	2.0			none	38.0	none			yes
20	2.0	4.0	5.0		tcf	35.0		13.0	5.0	
21	2.0	4.3	4.4			38.0			4.0	
22	2.0	2.5	3.0			40.0	none			
23	3.0	3.5	4.0	4.6	tcf	27.0				
24	2.0	4.5	4.0			40.0			4.0	
25	1.0	6.0				38.0		8.0	3.0	
26	3.0	2.0	2.0	2.0	none	40.0	none			
27	2.0	4.5	4.5		tcf					yes
28	2.0	3.0	3.0		none	33.0				yes
29	2.0	5.0	4.0		none	37.0			5.0	no
30	3.0	2.0	2.5			35.0	none			
31	3.0	4.5	4.5	5.0	none	40.0				no
32	3.0	3.0	2.0	2.5	tc	40.0	none		5.0	no
33	2.0	2.5	2.5			38.0	empl_c...			
34	2.0	4.0	5.0		none	40.0	none		3.0	no
35	3.0	2.0	2.5	2.1	tc	40.0	none	2.0	1.0	no

Right click (or left+alt) for context menu

The following screenshot shows the effect of discretization



Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Open file... | Open URL... | Open DB... | Generate... | Undo | Edit... | Save...

Filter: Choose **Discretize** -B 10 -M -1.0 -R first-last — Apply

Current relation: Relation: labor-neg-data-weka.filters.unsupervised.attribute.Discretize-B10-M-1.0-Rfirst-last-... Instances: 57 Attributes: 17

Attributes: All | None | Invert | Pattern

No.	Name
1	duration
2	wage-increase-first-year
3	wage-increase-second-year
4	wage-increase-third-year
5	cost-of-living-adjustment
6	working-hours
7	pension
8	standby-pay
9	shift-differential
10	education-allowance
11	statutory-holidays
12	vacation
13	longterm-disability-assistance
14	contribution-to-dental-plan
15	bereavement-assistance
16	contribution-to-health-plan
17	class

Remove

Selected attribute: Name: duration Missing: 1 (2%) Distinct: 3 Type: Nominal Unique: 0 (0%)

No.	Label	Count
1	'(-inf-1.2]'	10
2	'(1.2-1.4]'	0
3	'(1.4-1.6]'	0
4	'(1.6-1.8]'	0
5	'(1.8-2]'	27
6	'(2-2.2]'	0
7	'(2.2-2.4]'	0
8	'(2.4-2.6]'	0
9	'(2.6-2.8]'	0
10	'(2.8-inf)'	19

Class: class (Nom) — Visualize All

Status: OK — Log — x 0

start | Weka GUI Chooser | Weka Explorer | (bulk - Notepad) | 1:58 PM

B. Load each dataset into Weka and run Aprior algorithm with different support and confidence values. Study the rules generated.

Ans:

Steps for run Apriori algorithm in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose Weather data set and open file.
8. Click on Associate tab and Choose Aprior algorithm
9. Click on start button.

Association Rule:

An association rule has two parts, an antecedent (if) and a consequent (then). An antecedent is an item found in the data. A consequent is an item that is found in combination with the antecedent.

Association rules are created by analyzing data for frequent if/then patterns and using the criteriasupport and confidence to identify the most important relationships. Support is an indication of how frequently the items appear in the database. Confidence indicates the number of times the if/then statements have been found to be true.

In data mining, association rules are useful for analyzing and predicting customer behavior. They play an important part in shopping basket data analysis, product clustering, catalog design and store layout.

Support and Confidence values:

Support count: The support count of an itemset X, denoted by $X.count$, in a data set T is the number of transactions in T that contain X. Assume T has n transactions.

Then,

$support$ ($X \cup Y$).$count$ n

$confidence$ support = support({$A \cup C$})

($X \cup Y$).$count$

confidence = support({$A \cup C$})/support({A})

C.Apply different discretization filters on numerical attributes and run the Aprior association rule algorithm. Study the rules generated. Derive interesting insights and observe the effect of discretization in the rule generation process.

Ans: Steps for run Aprior algorithm in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose Weather data set and open file.
8. Choose filter button and select the Unsupervised-Discritize option and apply
9. Click on Associate tab and Choose Aprior algorithm
10. Click on start button.

WEEK-3:

3. Perform data preprocessing tasks and Demonstrate performing association rule mining on data sets
   - Explore various options available in Weka for preprocessing data and apply Unsupervised filters like Discretization, Resample filter, etc. on each dataset
   - Load weather. nominal, Iris, Glass datasets into Weka and run Apriori Algorithm with different support and confidence values.
   - Study the rules generated. Apply different discretization filters on numerical attributes and run the Apriori association rule algorithm. Study the rules generated.
   - Derive interesting insights and observe the effect of discretization in the rule generation process.

Perform data preprocessing tasks and Demonstrate performing association rule mining on data sets

A. Explore various options in Weka for Preprocessing data and apply (like Discretization Filters, Resample filter, etc.) n each dataset.
Ans:

Preprocess Tab

Preprocess Tab

1. Loading Data

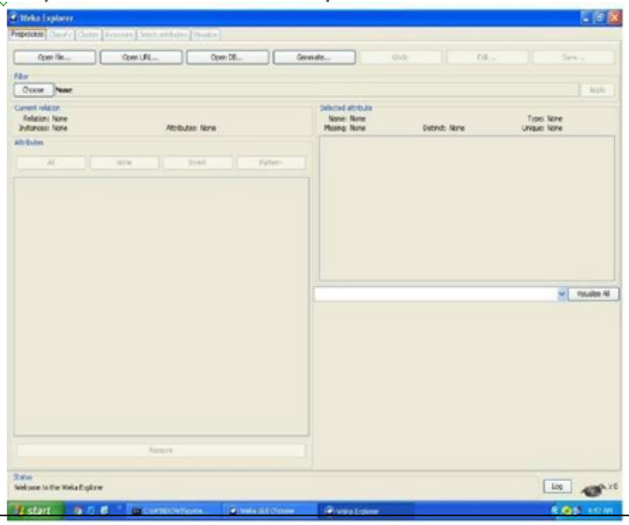
The first four buttons at the top of the preprocess section enable you to load data into WEKA:

1. **Open file....** Brings up a dialog box allowing you to browse for the data file on the local file system.

2. **Open URL....** Asks for a Uniform Resource Locator address for where the data is stored.

3. **Open DB ...** Reads data from a database. (Note that to make this work you might have to edit the file in weka/experiment/DatabaseUtils.props.)

4. **Generate....** Enables you to generate artificial data from a variety of Data Generators. Using the Open file... button you can read files in a variety of formats: **WEKA's ARFF format, CSV** format, C4.5 format, or serialized Instances format. ARFF files typically have a .arff extension, CSV files a .csv extension, C4.5 files a .data and .names extension, and serialized Instances objects a .bsi extension.



Current Relation: Once some data has been loaded, the Preprocess panel shows a variety of in**formation. The Current relation box (the "current relation" is the** currently loaded data, which can be interpreted as a single relational table in database terminology) has three entries:

1. **Relation.** The name of the relation, as given in the file it was loaded from. Filters (described below) modify the name of a relation.

2. **Instances.** The number of instances (data points/records) in the data.

3. **Attributes.** The number of attributes (features) in the data.

Working with Attributes

Below the Current relation box is a box titled Attributes. There are four buttons, and beneath them is a list of the attributes in the current relation.

The list has three columns:

1. **No.** A number that identifies the attribute in the order they are specified in the data file.
2. **Selection tick boxes**. These allow you select which attributes are present in the relation.
3. **Name.** The name of the attribute, as it was declared in the data file. When you click on different rows in the list of attributes, the fields change in the box to the right titled selected attribute.

This box displays the characteristics of the currently highlighted attribute in the list:

1. **Name.** The name of the attribute, the same as that given in the attribute list.
2. **Type.** The type of attribute, most commonly Nominal or Numeric.
3. **Missing.** The number (and percentage) of instances in the data for which this attribute is missing (unspecified).
4. **Distinct.** The number of different values that the data contains for this attribute.
5. **Unique.** The number (and percentage) of instances in the data having a value for this attribute that no other instances have.

Below these statistics is a list showing more information about the values stored in this attribute, which differ depending on its type. If the attribute is nominal, the list consists of each possible value for the attribute along with the number of instances that have that value. If the attribute is numeric, the list gives four statistics describing the distribution of values in the data— the minimum, maximum, mean and standard deviation. And below these statistics there is a coloured histogram, colour-coded according to the attribute chosen as the Class using the box above the histogram. (This box will bring up a drop-down list of available selections when clicked.) Note that only nominal Class attributes will result in a colour-coding. Finally, after pressing the Visualize All button, histograms for all the attributes in the data are shown in a separate window.

Returning to the attribute list, to begin with all the tick boxes are unticked.They can be toggled on/off by clicking on them individually. The four buttons above can also be used to change the selection:

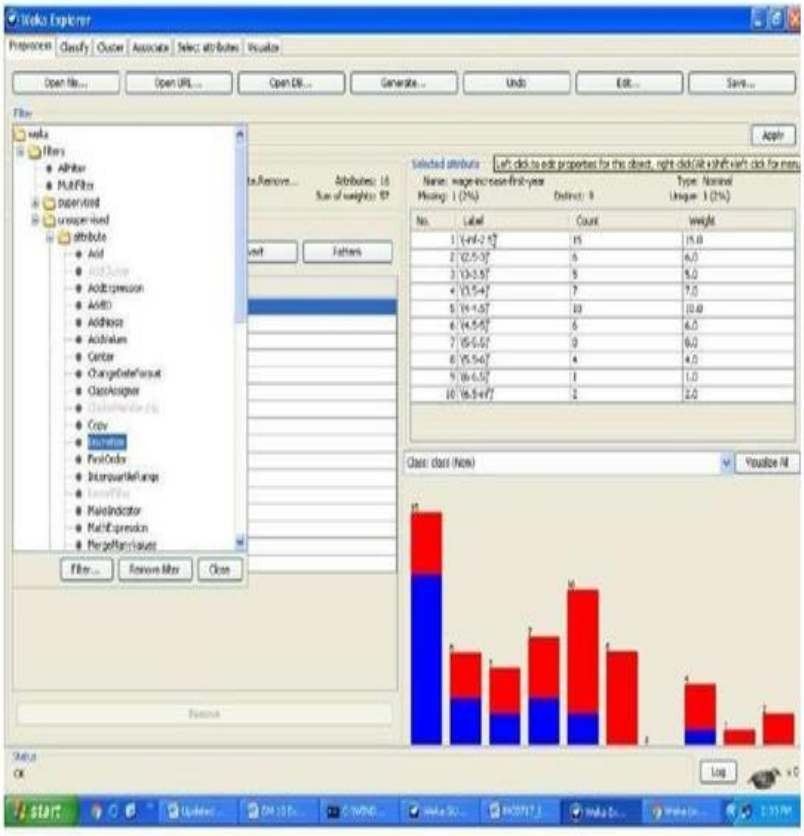
PREPROCESSING

1. **All.** All boxes are ticked.
2. **None.** All boxes are cleared (unticked).
3. **Invert.** Boxes that are ticked become unticked and vice versa.
4. **Pattern.** Enables the user to select attributes based on a Perl 5 Regular Expression. E.g.,.* id selects all attributes which name ends with id.

Once the desired attributes have been selected, they can be removed by clicking the Remove button below the list of attributes. Note that this can be undone by clicking the Undo button, which is located next to the Edit button in the top-right corner of the Preprocess panel.

Working with Filters:-

The preprocess section allows filters to be defined that transform the data in various ways. The Filter box is used to set up the filters that are required. At the left of the Filter box is a Choose button. By clicking this button it is possible to select one of the filters in WEKA. Once a filter has been selected, its name and options are shown in the field next to the Choose button. Clicking on this box with the left mouse button brings up a GenericObjectEditor dialog box. A click with the right mouse button (or Alt+Shift+left click) brings up a menu where you can choose, either to display the properties in a GenericObjectEditor dialog box, or to copy the current setup string to the clipboard.



The GenericObjectEditor Dialog Box

The GenericObjectEditor dialog box lets you configure a filter. The same kind of dialog box is used to configure other objects, such as classifiers and clusterers. The fields in the window reflect the available options. Right-clicking (or Alt+Shift+Left-Click) on such a field will bring up a popup menu, listing the following options:

1. **Show properties...** has the same effect as left-clicking on the field, i.e., a dialog appears allowing you to alter the settings.
2. **Copy configuration** to clipboard copies the currently displayed configuration string to the **system's clipboar**d and therefore can be used anywhere else in WEKA or in the console. This is rather handy if you have to setup complicated, nested schemes.
3. **Enter configuration... is the "receiving" end for configurations that** got copied to the clipboard earlier on. In this dialog you can enter a class name followed by options (if the class supports these). This also allows you to transfer a filter setting from the Preprocess panel to a Filtered Classifier used in the Classify panel.

Left-Clicking on any of these gives an opportunity to alter the filters settings. For example, the setting may take a text string, in which case you type the string into the text field provided. Or it may give a drop-down box listing several states to choose from. Or it may do something else, depending on the information required. Information on the options is provided in a tool tip if you let the mouse pointer hover of the corresponding field. More information on the filter and its options can be obtained by clicking on the More button in the About panel at the top of the GenericObjectEditor window.

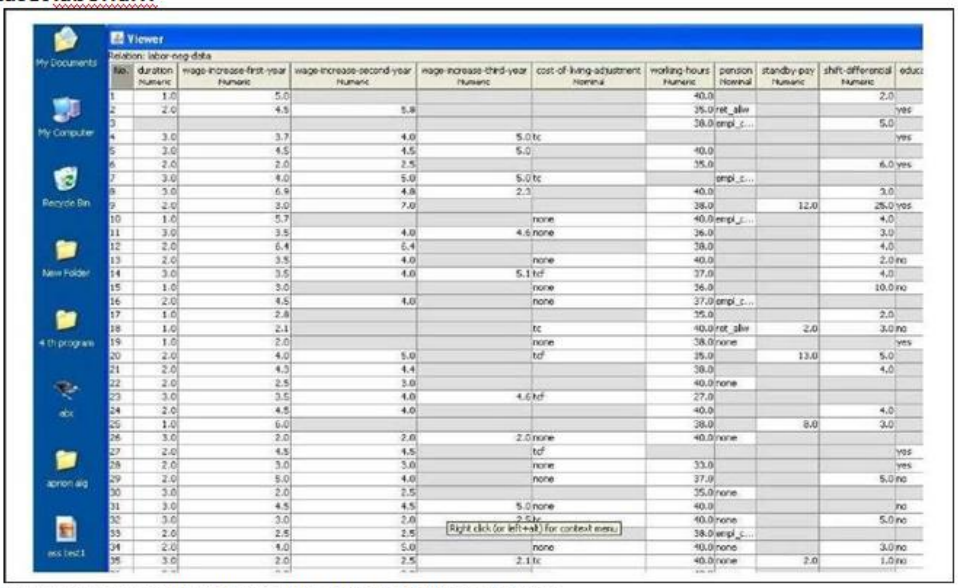
Applying Filters

Once you have selected and configured a filter, you can apply it to the data by pressing the Apply button at the right end of the Filter panel in the Preprocess panel. The Preprocess panel will then show the transformed data. The change can be undone by pressing the Undo button. You can also use the Edit...button to modify your data manually in a dataset editor. Finally, the Save...button at the top right of the Preprocess panel saves the current version of the relation in file formats that can represent the relation, allowing it to be kept for future use.

Steps for run preprocessing tab in WEKA

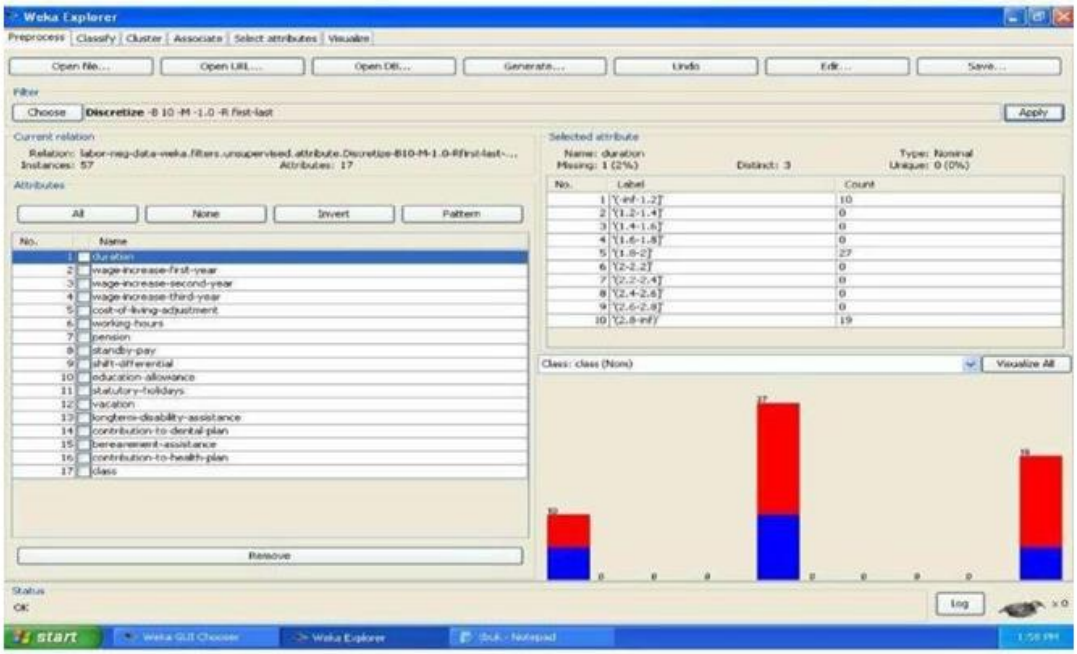
1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose labor data set and open file.
8. Choose filter button and select the Unsupervised-Discritize option and apply

Dataset labor.arff



The following screenshot shows the effect of discretization

The following screenshot shows the effect of discretization



B. Load each dataset into Weka and run Aprior algorithm with different support and confidence values. Study the rules generated.

Ans:
Steps for run Apriori algorithm in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose Weather data set and open file.
8. Click on Associate tab and Choose Aprior algorithm
9. Click on start button.

Association Rule:

An association rule has two parts, an antecedent (if) and a consequent (then). An antecedent is an item found in the data. A consequent is an item that is found in combination with the antecedent.

Association rules are created by analyzing data for frequent if/then patterns and using the criteriasupport and confidence to identify the most important relationships. Support is an indication of how frequently the items appear in the database. Confidence indicates the number of times the if/then statements have been found to be true.

In data mining, association rules are useful for analyzing and predicting customer behavior. They play an important part in shopping basket data analysis, product clustering, catalog design and store layout.

Support and Confidence values:

Support count: The support count of an itemset X, denoted by $X.count$, in a data set T is the number of transactions in T that contain X. Assume T has n transactions.

Then,

$support(X \cup Y).count\ n$

$confidence\ \text{support} = \text{support}(\{A \cup C\})$

$(X \cup Y).count$

$\text{confidence} = \text{support}(\{A \cup C\})/\text{support}(\{A\})$

C.Apply different discretization filters on numerical attributes and run the Aprior association rule algorithm. Study the rules generated. Derive interesting insights and observe the effect of discretization in the rule generation process.

Ans: Steps for run Aprior algorithm in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose Weather data set and open file.
8. Choose filter button and select the Unsupervised-Discritize option and apply
9. Click on Associate tab and Choose Aprior algorithm
10. Click on start button.

WEEK-4

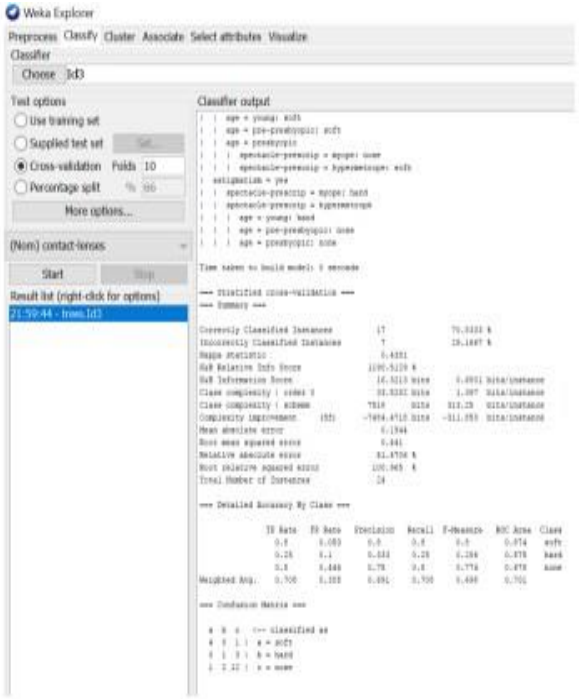
4. Demonstrate performing classification on data sets

- Load each dataset into Weka and run 1d3, J48 classification algorithm. Study the classifier output. Compute entropy values, Kappa statistic.
- Extract if-then rules from the decision tree generated by the classifier, Observe the confusion matrix.
- Load each dataset into Weka and perform Naïve-bayes classification and k-Nearest Neighbour classification. Interpret the results obtained.
- Plot RoC Curves
- Compare classification results of ID3, J48, Naïve-Bayes and k-NN classifiers for each dataset, and deduce which classifier is performing best and poor for each dataset and justify.

A. Load each dataset into Weka and run Id3, J48 classification algorithm. Study the classifier output. Compute entropy values, Kappa statistic.

Procedure for Id3:

1. Load the dataset (Contact-lenses.arff) into weka tool
2. Go to classify option & in left-hand navigation bar we can see differentclassification algorithms under tree section.
3. In which we selected Id3 algorithm, in more options select the output entropy evaluation measures& click on start option.
4. Then we will get classifier output, entropy values& Kappa Statistic as represented below.



5. In the above screenshot, we can run classifiers with different test options (Cross-validation, Use Training Set, Percentage Split, Supplied Test set).

The result of applying the chosen classifier will be tested according to the optionsthat are set by clicking in the Test options box. There are four test modes:

A. Use training set: The classifier is evaluated on how well it predicts theclass of the instances it was trained on.

B. Supplied test set: The classifier is evaluated on how well it predicts theclass of a set ofinstances loaded from a file. Clicking the Set... buttonbrings up a dialog allowing you to choose the file to test on.

C. Cross-validation: The classifier is evaluated by cross-validation, usingthe number of folds that are entered in the Folds text field.

D. Percentage split: The classifier is evaluated on how well it predicts acertain percentage of the data which is held out for testing. The amountof data held out depends on the value entered in the % field.

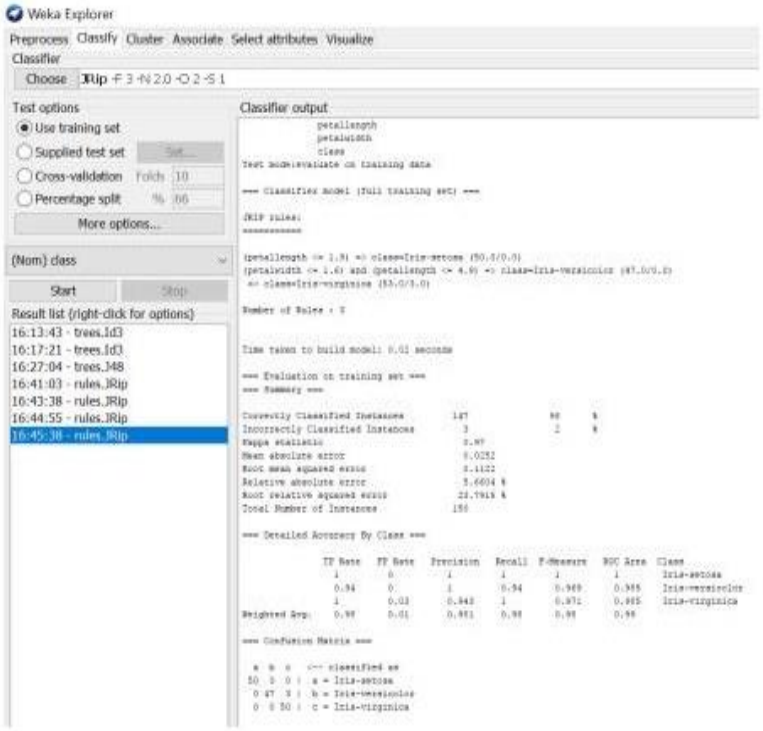
Procedure for J48:

1. Load the dataset (Contact-lenses.arff) into weka tool
2. Go to classify option & in left-hand navigation bar we can see differentclassification algorithms under tree section.
3. In which we selected J48 algorithm,in more options select the output entropy evaluation measures& click on start option.
4. Then we will get classifier output, entropy values & Kappa Statistic as represented below.
5. In the below screenshot, we can run classifiers with different test options (Cross-validation, Use Training Set, Percentage Split, Supplied Test set).

B. Extract if-then rules from the decision tree generated by the classifier, Observethe confusion matrix and derive Accuracy, F-measure, TPrate, FPrate, Precision and Recall values. Apply cross-validation strategy with various fold levels and compare the accuracy results.

Procedure:

1. Load the dataset (Iris-2D. arff) into weka tool
2. Go to classify option & in left-hand navigation bar we can see differentclassification algorithms under rules section.
3. In which we selected JRip (If-then) algorithm & click on start option with "use training set" test option enabled.
4. Then we will get detailed accuracy by class consists ofF-measure, TP rate, FP rate, Precision, Recall values& Confusion Matrix as represented below.

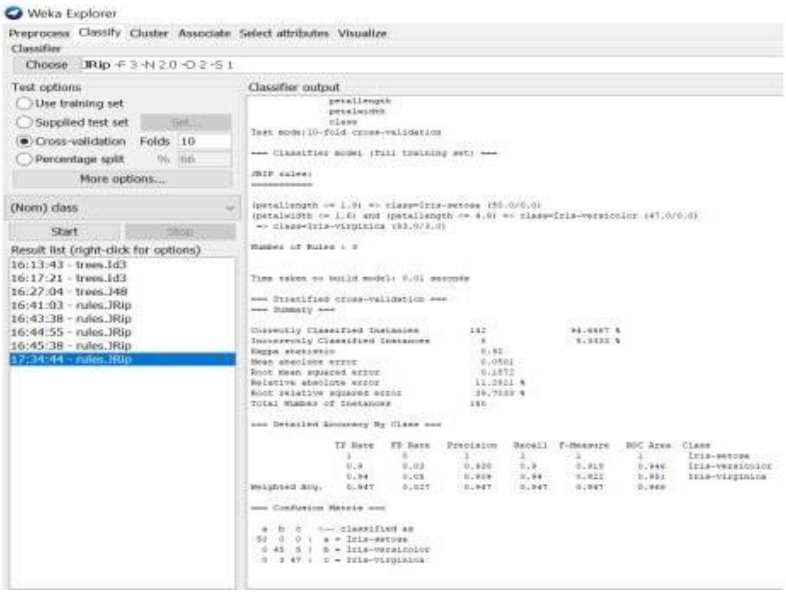


Using Cross-Validation Strategy with 10 folds:

Here, we enabled cross-validation test option with 10 folds & clicked start button as represented below.

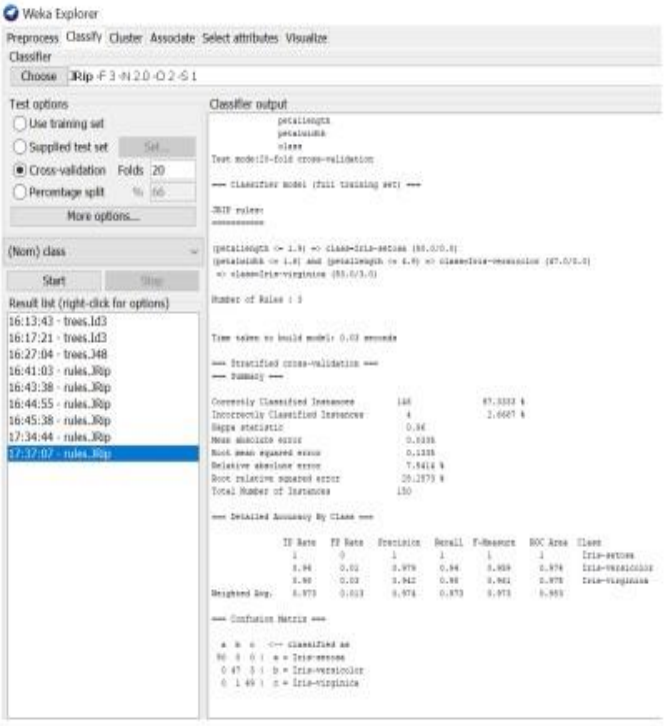
Using Cross-Validation Strategy with 10 folds:

Here, we enabled cross-validation test option with 10 folds & clicked start button as represented below.



Using Cross-Validation Strategy with 20 folds:

Here, we enabled cross-validation test option with 20 folds & clicked start button as represented below.

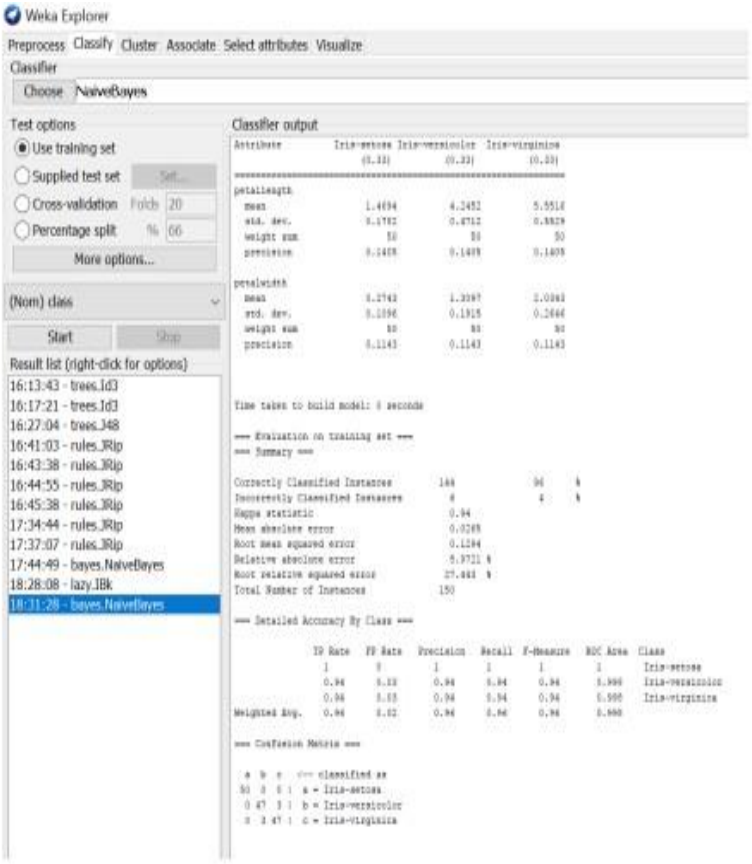


If we see the above results of cross validation with 10 folds & 20 folds. As per our observation the error rate is lesser with 20 folds got 97.3% correctness when compared to 10 folds got 94.6% correctness.

C. Load each dataset into Weka and perform Naive-bayes classification and k-Nearest Neighbour classification. Interpret the results obtained.

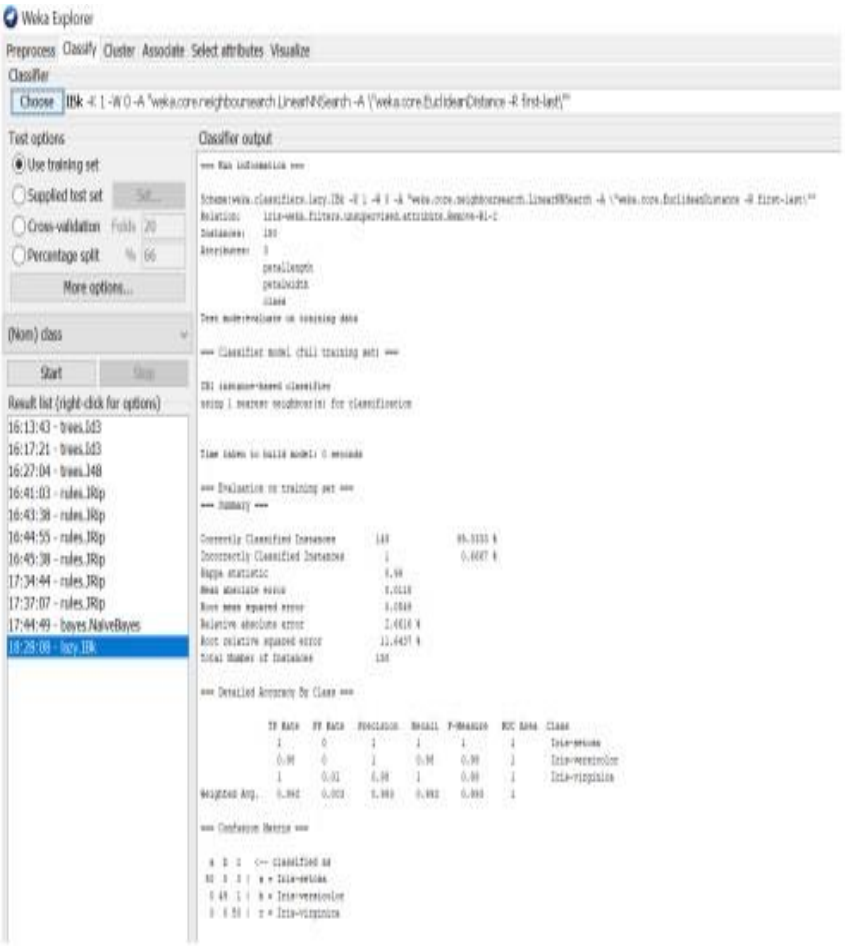
Procedure for Naïve-Bayes:

1. Load the dataset (Iris-2D. arff) into weka tool
2. Go to classify option & in left-hand navigation bar we can see differentclassification algorithms under bayes section.
3. In which we selected Naïve-Bayes algorithm & click on start option with "use training set" test option enabled.
4. Then we will get detailed accuracy by class consists of F-measure, TP rate, FP rate, Precision, Recall values& Confusion Matrix as represented below.



Procedure for K-Nearest Neighbour (IBK):

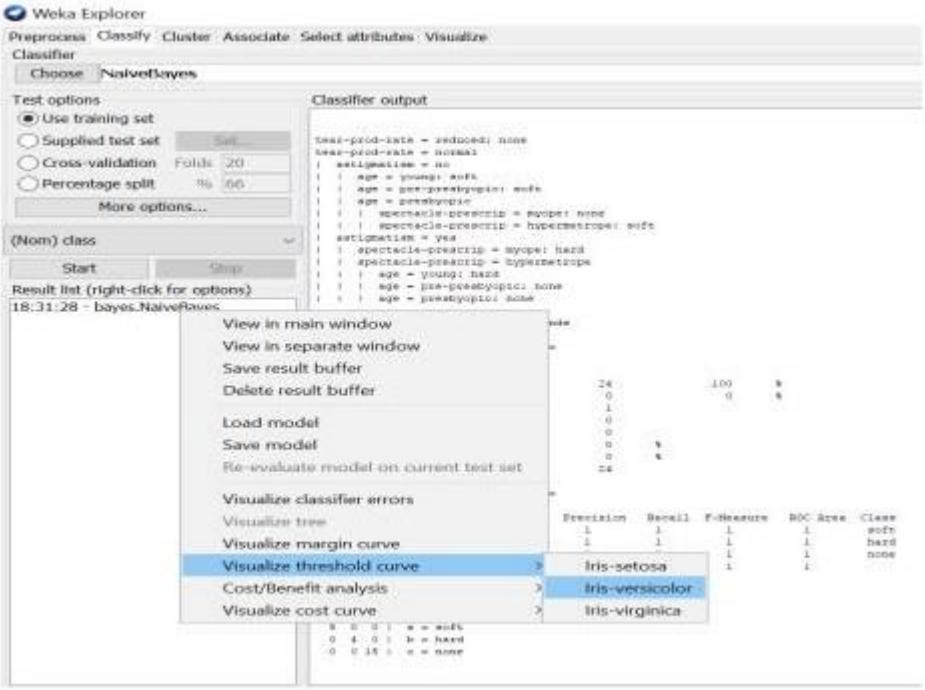
1. Load the dataset (Iris-2D. arff) into weka tool
2. Go to classify option & in left-hand navigation bar we can see differentclassification algorithms under lazy section.
3. In which we selected K-Nearest Neighbour (IBK) algorithm & click on start option with "use training set" test option enabled.
4. Then we will get detailed accuracy by class consists of F-measure, TP rate, FP rate, Precision, Recall values& Confusion Matrix as represented below.

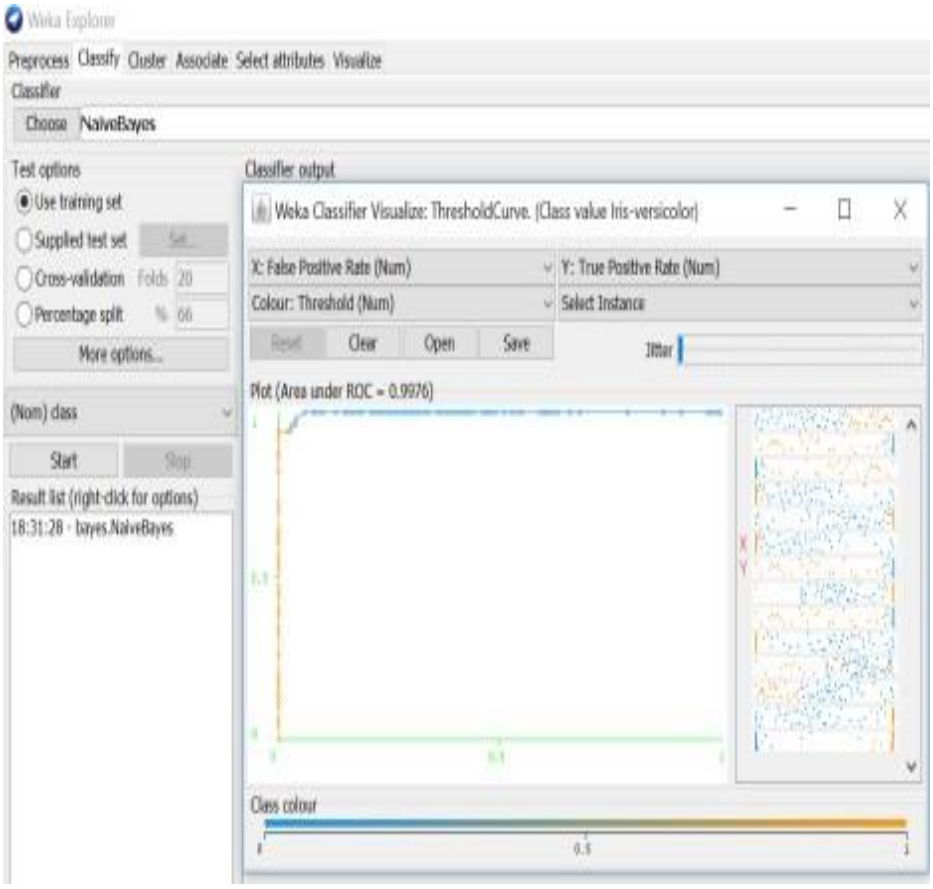


D. Plot RoC Curves

Procedure:

1. Load the dataset (Iris-2D. arff) into weka tool
2. Go to classify option & in left-hand navigation bar we can see differentclassification algorithms under bayes section.
3. In which we selected Naïve-Bayes algorithm & click on start option with "use training set" test option enabled.
4. Then we will get detailed accuracy by class consists of F-measure, TP rate, FP rate, Precision, Recall values& Confusion Matrix.
5. For plotting RoC Curves, we need to right click on "bayes.NaiveBayes" for getting more options, In which we will select the "Visualize Threshold Curve" & go to any class (Iris-setosa, Iris-versicolor, Iris-Virgincia) as shown in below snapshot.
6. After selecting an class, RoC (Receiver Operating Characteristic) Curve plot will be displayed which has X-Axis –False Positive (FP) rate and Y-Axis – True Positive (TP) rate.


Weka Explorer
Preprocess Classify Cluster Associate Select attributes Visualize
Classifier
Choose NaiveBayes
Test options
Use training set
Supplied test set
Cross-validation Folds 20
Percentage split % 66
More options...
(Nom) class
Start
Result list (right-click for options)
18:31:28 - bayes.NaiveBayes
Classifier output
View in main window
View in separate window
Save result buffer
Delete result buffer
Load model
Save model
Re-evaluate model on current test set
Visualize classifier errors
Visualize tree
Visualize margin curve
Visualize threshold curve
Cost/Benefit analysis
Visualize cost curve
Iris-setosa
Iris-versicolor
Iris-virginica


Preprocess Classify Cluster Associate Select attributes Visualize
Classifier
Choose NaiveBayes
Test options
Use training set
Supplied test set
Cross-validation Folds 20
Percentage split % 66
More options...
(Nom) class
Start
Result list (right-click for options)
18:31:28 - bayes.NaiveBayes
Classifier output
Weka Classifier Visualize: ThresholdCurve. (Class value Iris-versicolor)
X: False Positive Rate (Num)
Y: True Positive Rate (Num)
Colour: Threshold (Num)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot (Area under ROC = 0.9976)
Class colour

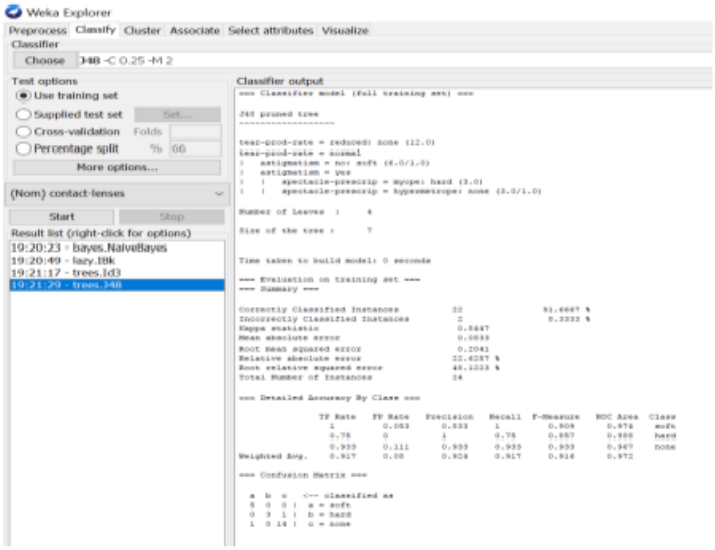
E. Compare classification results of ID3, J48, Naïve-Bayes and k-NN classifiers for each dataset, and deduce which classifier is performing best and poor for each dataset and justify.

Procedure for ID3:

1. Load the dataset (Contact-Lenses. arff) into weka tool
2. Go to classify option & in left-hand navigation bar we can see differentclassification algorithms under trees section.
3. In which we selected ID3 algorithm & click on start option with "use training set" test option enabled.
4. Then we will get detailed accuracy by class consists of F-measure, TP rate, FP rate, Precision, Recall values& Confusion Matrix as represented below.

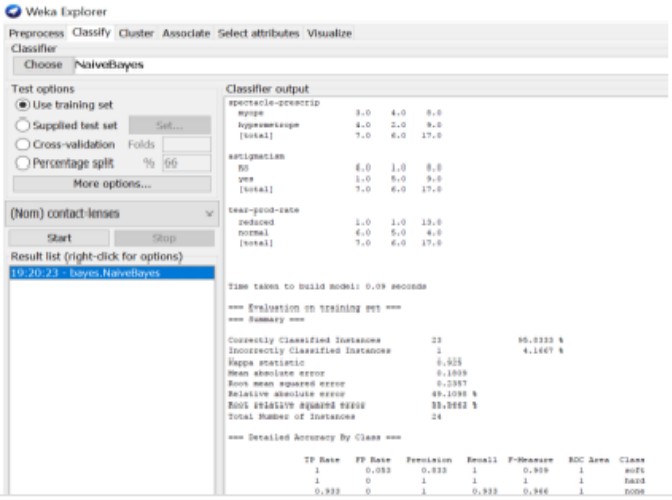
Procedure for J48:

1. Load the dataset (Contact-Lenses. arff) into weka tool
2. Go to classify option & in left-hand navigation bar we can see differentclassification algorithms under trees section.
3. In which we selected J48 algorithm & click on start option with "use training set" test option enabled.
4. Then we will get detailed accuracy by class consists of F-measure, TP rate, FP rate, Precision, Recall values& Confusion Matrix as represented below.



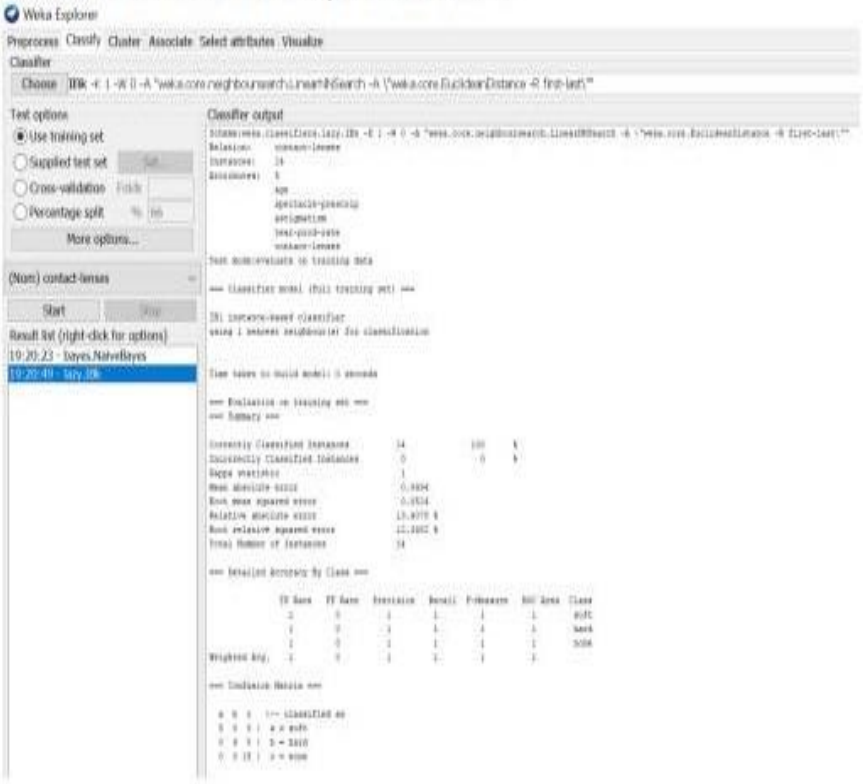
Procedure for Naïve-Bayes:

1. Load the dataset (Contact-Lenses. arff) into weka tool
2. Go to classify option & in left-hand navigation bar we can see differentclassification algorithms under bayes section.
3. In which we selected Naïve-Bayes algorithm & click on start option with "use training set" test option enabled.
4. Then we will get detailed accuracy by class consists of F-measure, TP rate, FP rate, Precision, Recall values& Confusion Matrix as represented below.



Procedure for K-Nearest Neighbour (IBK):

1. Load the dataset (Contact-Lenses. arff) into weka tool
2. Go to classify option & in left-hand navigation bar we can see differentclassifcation algorithms under lazy section.
3. In which we selected K-Nearest Neighbour (IBK) algorithm & click on start option with "use training set" test option enabled.
4. Then we will get detailed accuracy by class consists of F-measure, TP rate, FP rate, Precision, Recall values& Confusion Matrix as represented below.



By observing all these algorithms (ID3, K-NN, J48 & Naïve Bayes) results, we will conclude that

Hence,

ID3 Algorithm'saccuracy & performance is best.

J48 Algorithm's accuracy & performance is poor.

WEEK-5

Demonstrate performing clustering of data sets

- Load each dataset into Weka and run simple k-means clustering algorithm with different values of k (number of desired clusters).
- Study the clusters formed. Observe the sum of squared errors and centroids, and derive insights
- Explore other clustering techniques available in Weka
- Explore visualization features of Weka to visualize the clusters. Derive interesting insights and explain

K means clustering using Weka

Clustering: Clustering is the method of dividing a set of abstract objects into groups. Points to Keep in Mind A set of data objects can be viewed as a single entity. When performing cluster analysis, we divide the data set into groups based on data similarity, then assign labels to the groups.

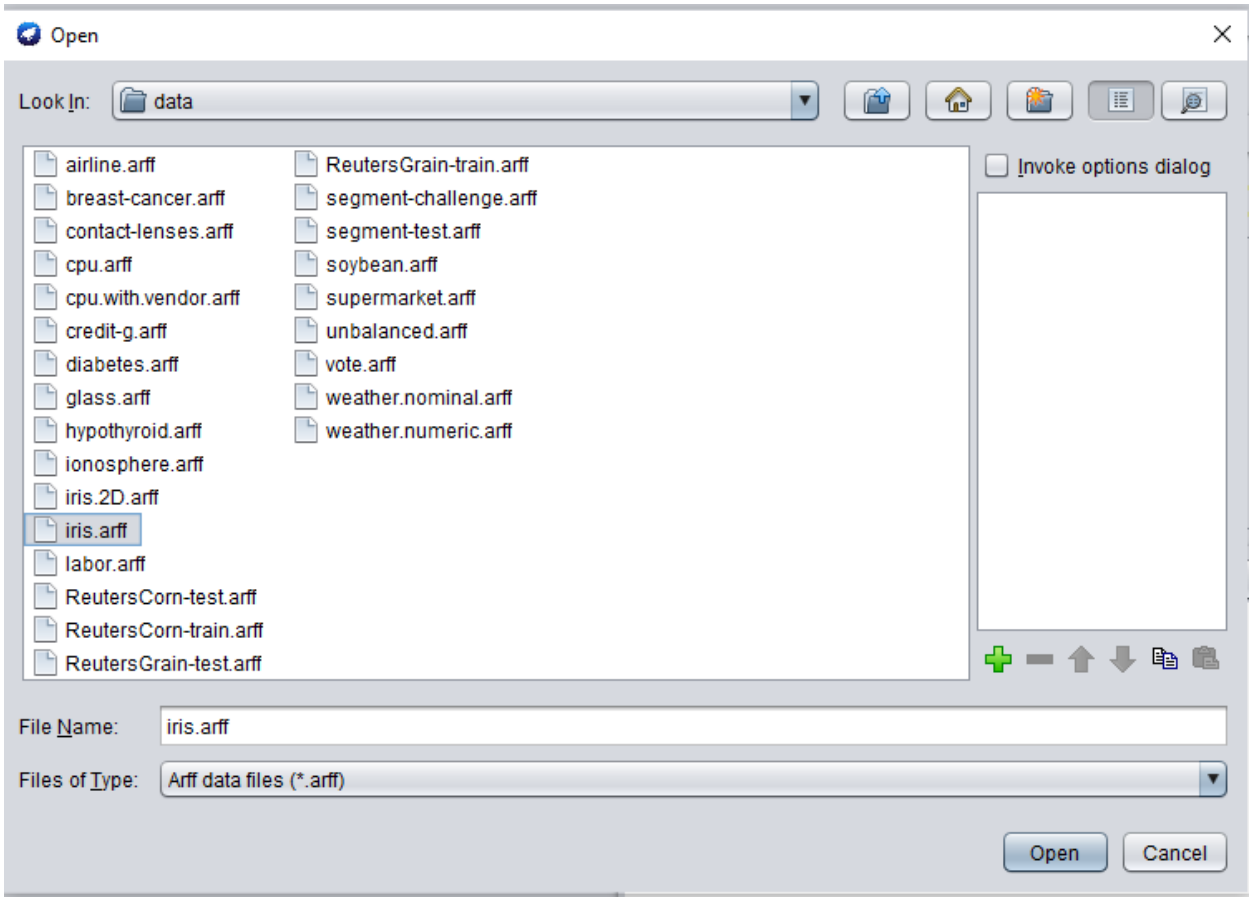
Simple-k means clustering: K-means clustering is a simple unsupervised learning algorithm. In this, the data objects (‘n’) are grouped into a total of ‘k’ clusters, with each observation belonging to the cluster with the closest mean. It defines ‘k’ sets, one for each cluster k n (the point can be thought of as the center of a one or two-dimensional figure). The clusters are separated by a large distance.

The data is then organized into acceptable data sets and linked to the nearest collection. If no data is pending, the first stage is more difficult to complete; in this case, an early grouping is performed. The ‘k’ new set must be recalculated as the barycenters of the clusters from the previous stage.

The same data set points and the nearest new sets are bound together after these ‘k’ new sets have been created. After that, a loop is created. The ‘k’ sets change their position step by step until no further changes are made as a result of this loop.

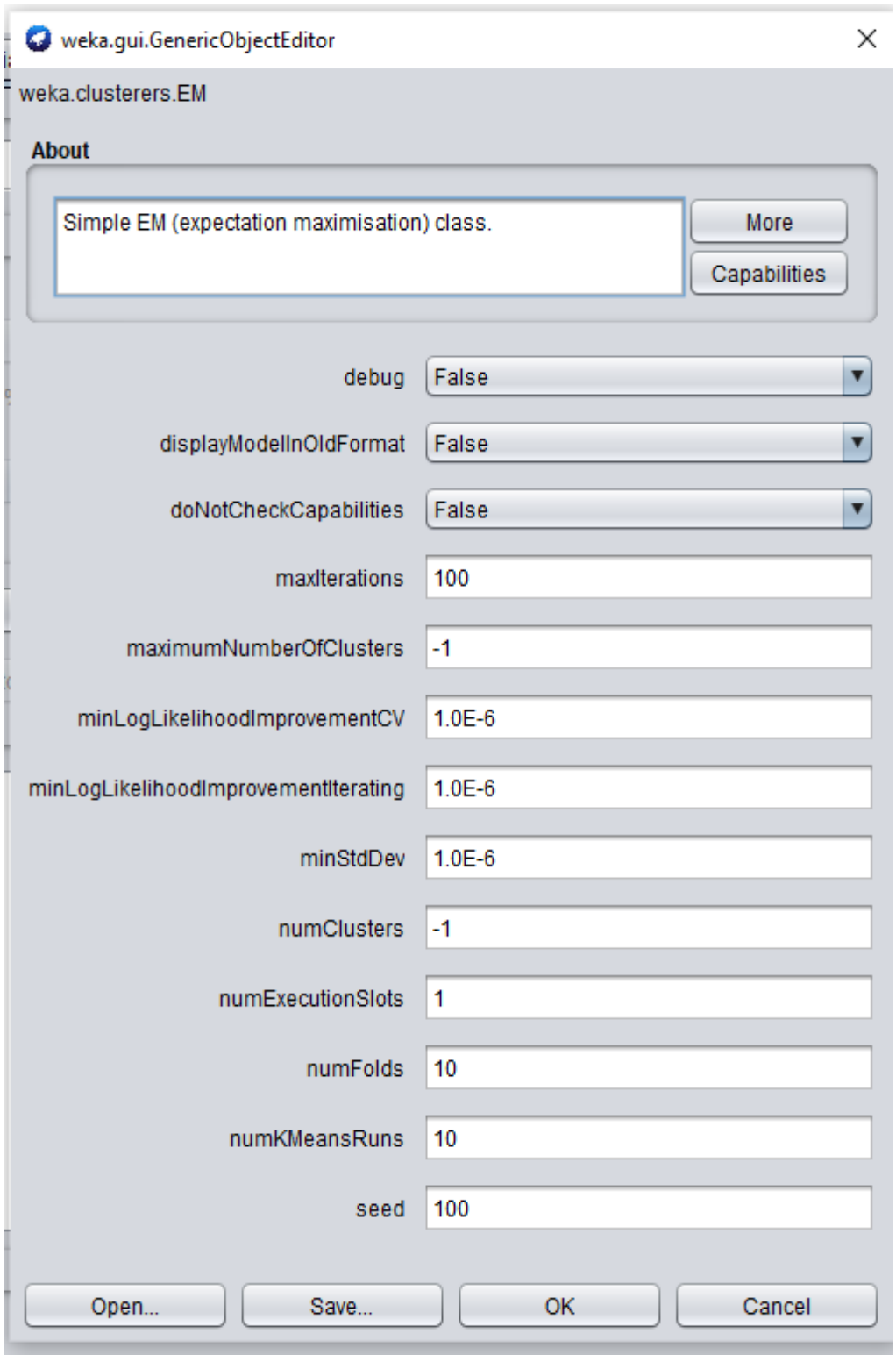
Steps to be followed:

Step 1: In the preprocessing interface, open the Weka Explorer and load the required dataset, and we are taking the iris.arff dataset.

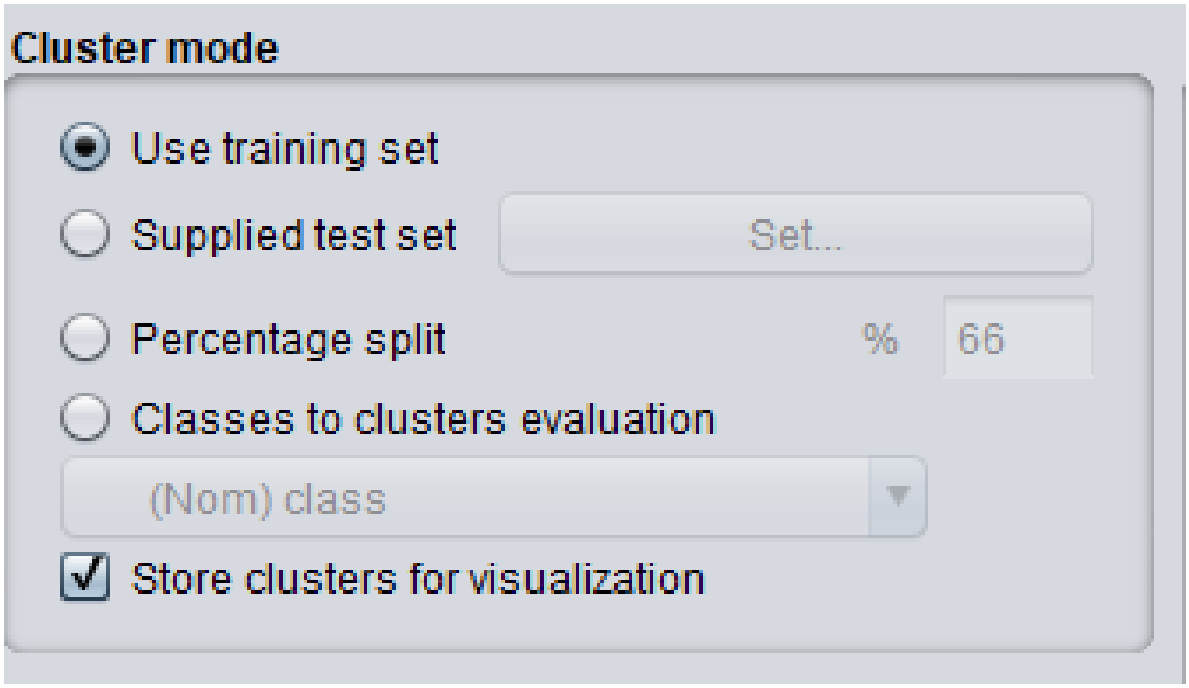


Step 2: Find the 'cluster' tab in the explorer and press the choose button to execute clustering. A dropdown list of available clustering algorithms appears as a result of this step and selects the simple-k means algorithm.

Step 3: Then, to the right of the choose icon, press the text button to bring up the popup window shown in the screenshots. We enter three for the number of clusters in this window and leave the seed value alone. The seed value is used to generate a random number that is used to make internal assignments of instances of clusters.


weka.gui.GenericObjectEditor
weka.clusterers.EM
About
Simple EM (expectation maximisation) class.
More
Capabilities
debug
False
displayModelInOldFormat
False
doNotCheckCapabilities
False
maxIterations
100
maximumNumberOfClusters
-1
minLogLikelihoodImprovementCV
1.0E-6
minLogLikelihoodImprovementIterating
1.0E-6
minStdDev
1.0E-6
numClusters
-1
numExecutionSlots
1
numFolds
10
numKMeansRuns
10
seed
100
Open...
Save...
OK
Cancel

Step 4: One of the choices has been chosen. We must ensure that they are in the 'cluster mode' panel before running the clustering algorithm. The choice to use a training set is selected, and then the 'start' button is pressed. The screenshots below display the process and the resulting window.

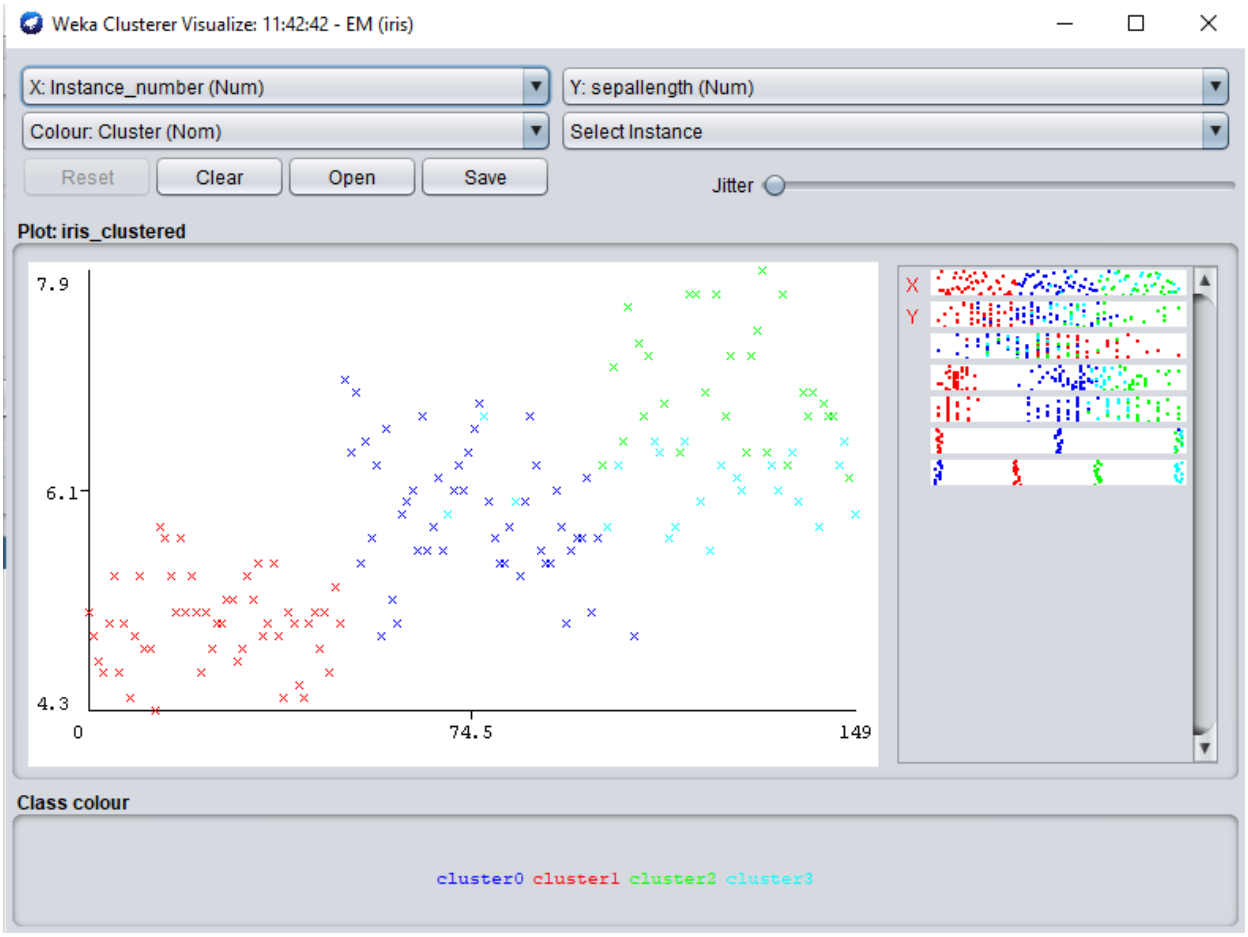


Step 5: The centroid of each cluster is shown in the result window, along with statistics on the number and percent of instances allocated to each cluster. Each cluster centroid is represented by a mean vector. This cluster can be used to describe a cluster

Number of clusters selected by cross validation: 4
Number of iterations performed: 16

Attribute	Cluster 0 (0.32)	1 (0.33)	2 (0.2)	3 (0.14)
sepallength				
mean	5.897	5.006	6.9426	6.1304
std. dev.	0.5279	0.3489	0.498	0.2943
sepalwidth				
mean	2.7519	3.418	3.1103	2.8088
std. dev.	0.3103	0.3772	0.2952	0.2361
petallength				
mean	4.2267	1.464	5.8559	5.0993
std. dev.	0.445	0.1718	0.4626	0.2462
petalwidth				
mean	1.3134	0.244	2.1495	1.8254
std. dev.	0.1864	0.1061	0.232	0.2152
class				
Iris-setosa	1	51	1	1
Iris-versicolor	48.1125	1	1.0182	3.8693
Iris-virginica	2.0983	1	31.0375	19.8641
[total]	51.2108	53	33.0557	24.7335

Step 6: Another way to grasp the characteristics of each cluster is to visualize them. To do so, right-click the result set on the result. Selecting to visualize cluster assignments from the list column.


Weka Clusterer Visualize: 11:42:42 - EM (iris)
X: Instance_number (Num)
Y: sepallength (Num)
Colour: Cluster (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: iris_clustered
7.9
6.1
4.3
0
74.5
149
X
Y
Class colour
cluster0 cluster1 cluster2 cluster3

WEEK-6

AIM:

Demonstrate knowledge flow application on data sets
Develop a knowledge flow layout for finding strong association rules by using Apriori, FP Growth
algorithms

Set up the knowledge flow to load an ARFF (batch mode) and perform a cross validation using J48
algorithm

Demonstrate plotting multiple ROC curves in the same plot window by using j48 and Random
forest tree

Develop a knowledge flow layout for finding strong association rules by using Apriori, FP GrowthAlgorithms

Aim: This experiment illustrates some of the basic elements of asscociation rule mining using WEKA. The sample dataset used for this example is contactlenses.arff

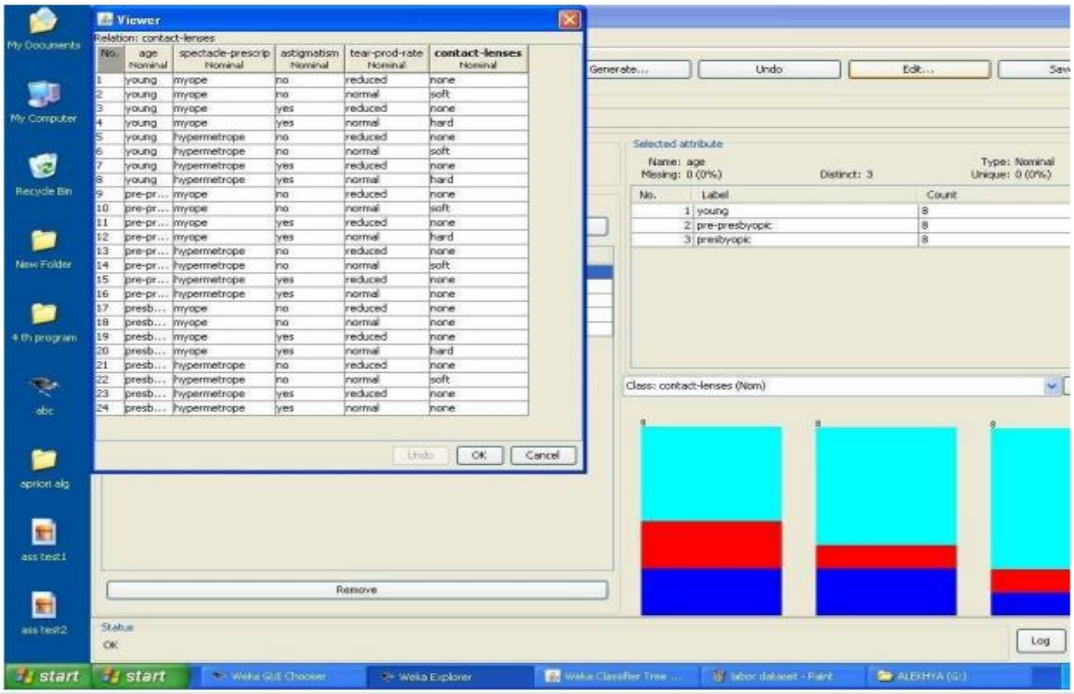
Step1: Open the data file in Weka Explorer. It is presumed that the required data fields have been discretized. In this example it is age attribute.

Step2: Clicking on the associate tab will bring up the interface for association rule algorithm.

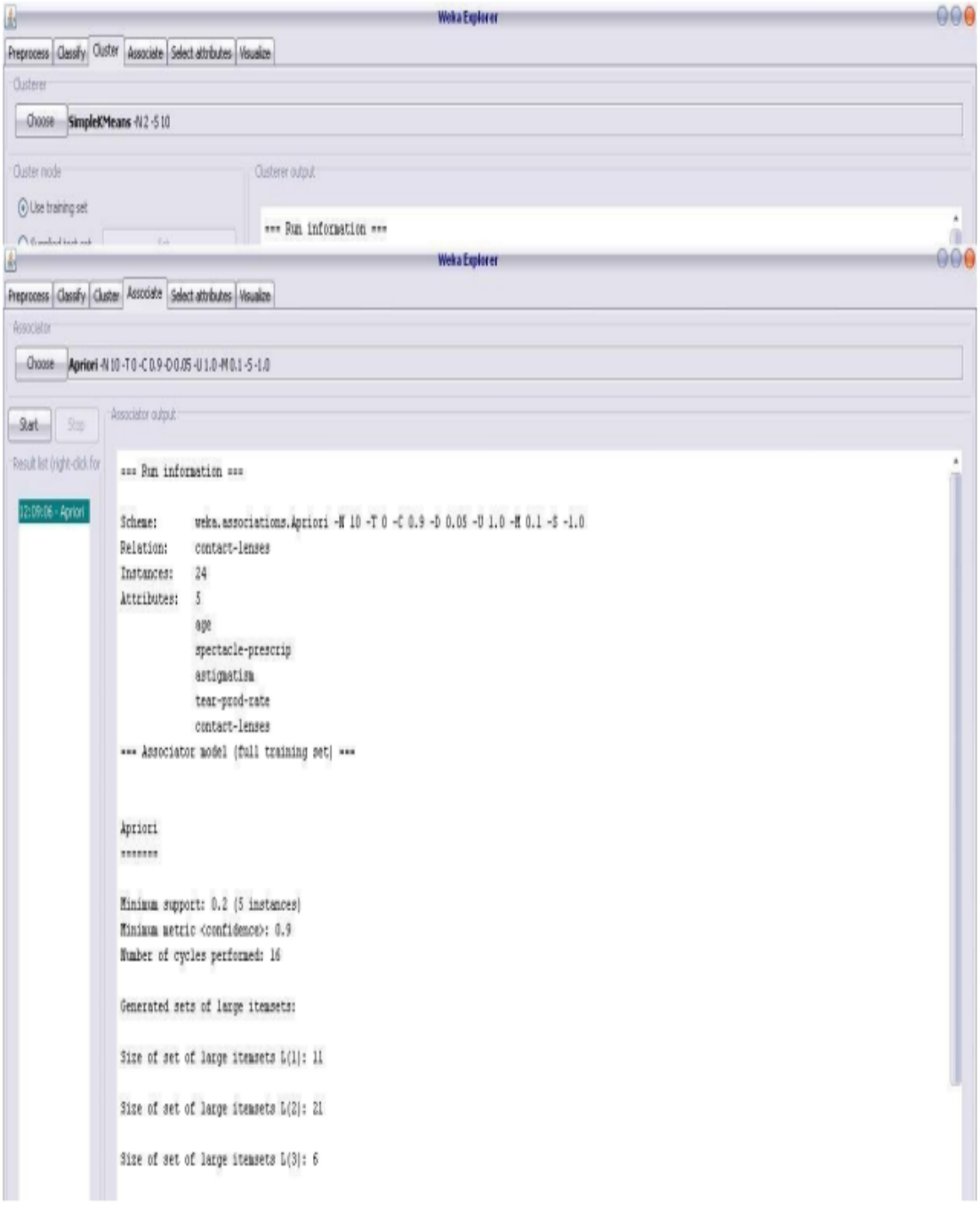
Step3: We will use apriori algorithm. This is the default algorithm.

Step4: Inorder to change the parameters for the run (example support, confidence etc) we click on the text box immediately to the right of the choose button.

Dataset contactlenses.arff



The following screenshot shows the association rules that were generated when apriori algorithm is applied on the given dataset.



Weka Explorer

Choose ISimpletMeans -N 2 -S 10

=== Run information ===

Weka Explorer

contact-lenses

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

Minimum support: 0.2 (5 instances)
Minimum metric <confidence>: 0.9

Generated sets of large itemsets:

Size of set of large itemsets L(1): 11

Size of set of large itemsets L(2): 21

Size of set of large itemsets L(3): 6

Best rules found:

1. tearprodrate=eJueJ 12 =› cont tlenses=none 12 ml:(1
2. atigmatism=yes tea-p J-rate=eJueJ 6 ==› cont tlenses=nOne 6 cOnf:(1)
3. atigmatism=no ten-p J-rate=eJueJ 6 ==› cont tlenses=nOne 6 cOnf:(1)
4. spectacle-prescrip=hypermetrope tear-proJ-rete=reJuAJ 6 ==) contact-lenses=nOne 6 conf:(1)
5. spectacle-prescrip=myope tear-proJ-rete=reJuceJ 6 ==) contat-lenses=nOne 6 cOnf:(1)
6. contact-lenses=soft 5 ==› astigmatism=no tear-proArate=nrmal 5 cOnf:(1)
7. astigBatism=no contact-lenses=soft A ==› tear-proArate=nrmal 5 cOnf:(1)
8. tear-prod-rate=normal contact-lenses=soft A ==› astigBatAm=nO 5 cOnf:(1)

FP-Growth Algorithm implementation in weka tool

Procedure:

Step1: Open the data file in Weka Explorer. It is presumed that the required data fields have been discretized. In this example it is age attribute.

Step2: Clicking on the associate tab will bring up the interface for association rule algorithm.

Step3: We will use FP-Growth algorithm. This is the default algorithm.

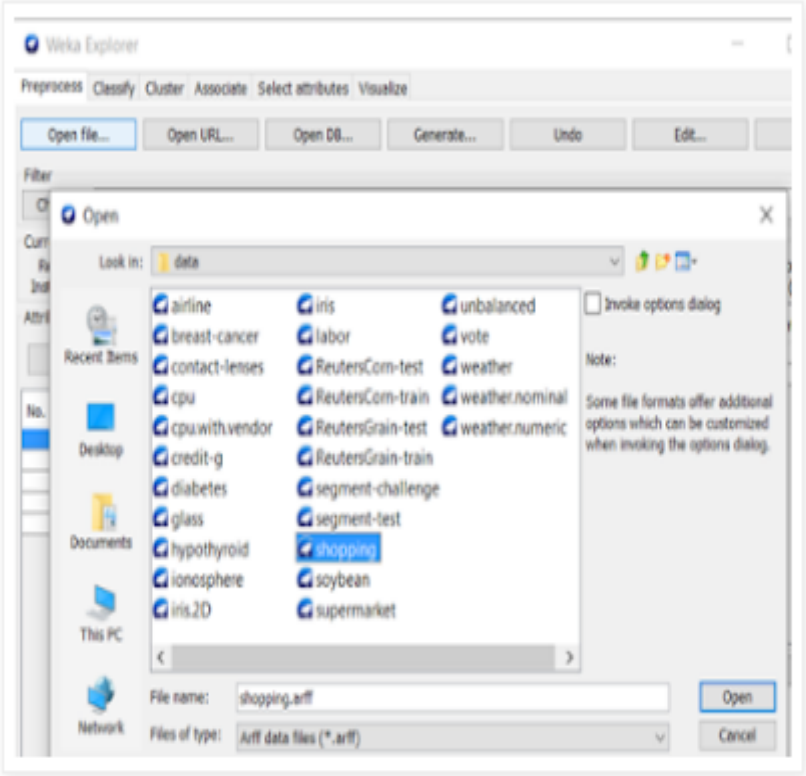
Step4: Inorder to change the parameters for the run (example support, confidence etc) we click on the text box immediately to the right of the choose button.

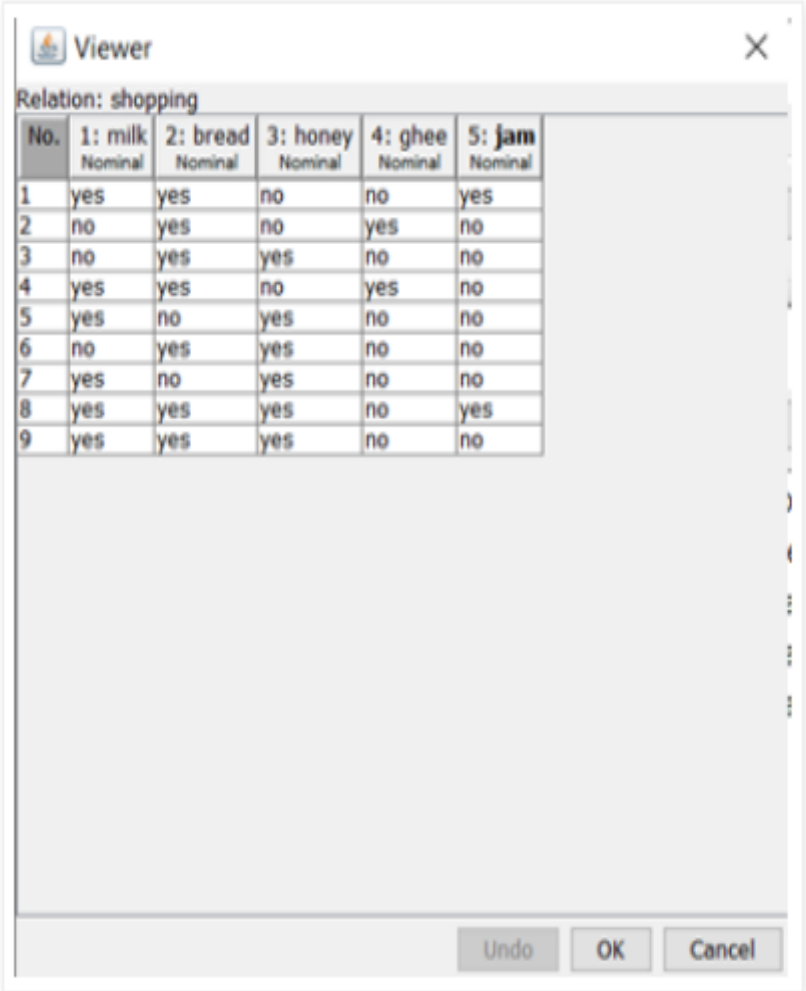
Data set:

Shopping.arff

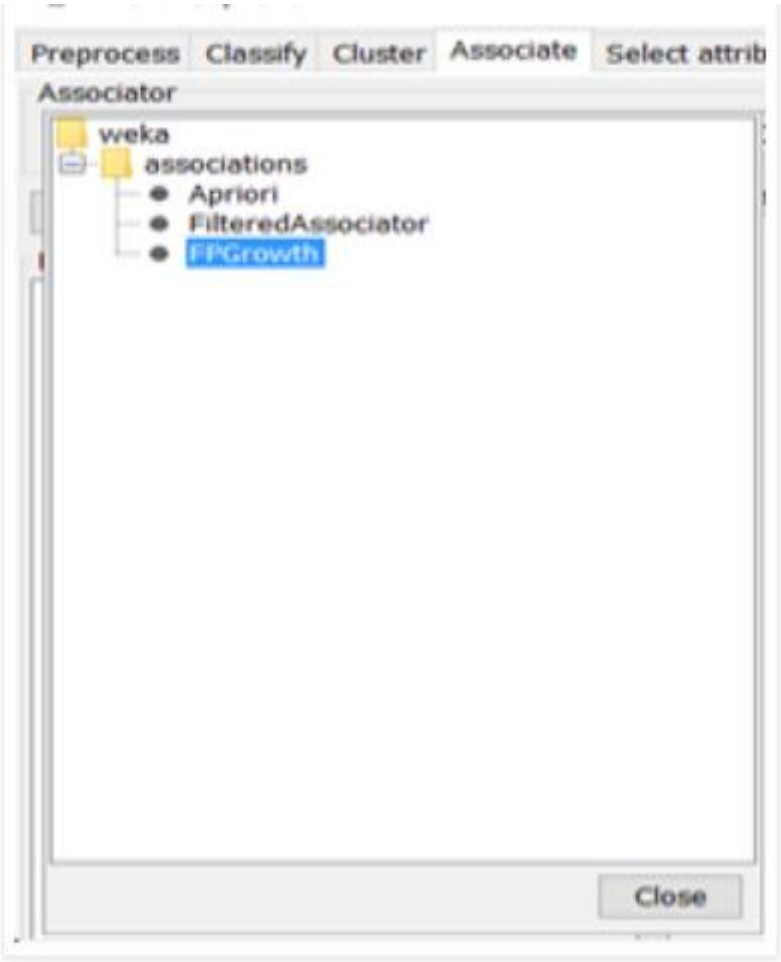
@relation shopping
@attribute milk{yes,no}
@attribute bread{yes,no}
@attribute honey{yes,no}
@attribute ghee{yes,no}
@attribute jam{yes,no}
@data
yes,yes,no,no,yes
no,yes,no,yes,no
no,yes,yes,no,no
yes,yes,no,yes,no
yes,no,yes,no,no
no,yes,yes,no,no
yes,no,yes,no,no
yes,yes,yes,no,yes

yes,yes,yes,no,no



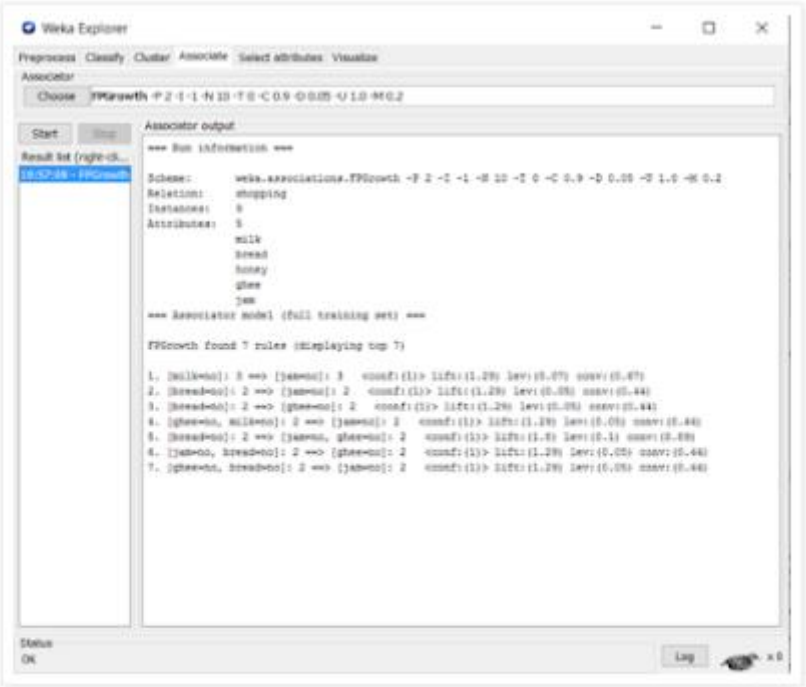


No.	1: milk Nominal	2: bread Nominal	3: honey Nominal	4: ghee Nominal	5: jam Nominal
1	yes	yes	no	no	yes
2	no	yes	no	yes	no
3	no	yes	yes	no	no
4	yes	yes	no	yes	no
5	yes	no	yes	no	no
6	no	yes	yes	no	no
7	yes	no	yes	no	no
8	yes	yes	yes	no	yes
9	yes	yes	yes	no	no


Preprocess
Classify
Cluster
Associate
Select attrib
Associator
weka
associations
Apriori
FilteredAssociator
FPGrowth
Close


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Associator
Choose
FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.2
weka.gui.GenericObjectEditor
weka.associations.FPGrowth
About
Class implementing the FP-growth algorithm for finding large item sets without candidate generation.
More
Capabilities
delta
0.05
doNotCheckCapabilities
False
findAllRulesForSupportLevel
False
lowerBoundMinSupport
0.2
maxNumberOfItems
-1
metricType
Confidence
minMetric
0.9
numRulesToFind
10
positiveIndex
2
rulesMustContain
transactionsMustContain
upperBoundMinSupport
1.0
useORForMustContainList
False
Status
OK
Open...
Save...
OK
Cancel

b


Weka Explorer
Preprocess Classify Cluster Associate Select attributes Visualize
Associator
Choose FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.2
Start Stop
Result list (right-cli...
18:57:08 - FPGrowth
Associator output
=== Run information ===
Scheme: weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.2
Relation: shopping
Instances: 5
Attributes: 5
milk
bread
honey
ghee
jam
=== Associator model (full training set) ===
FPGrowth found 7 rules (displaying top 7)
1. [milk=no]: 3 ==> [jam=no]: 3 <conf:(1)> lift:(1.25) lev:(0.07) conv:(0.67)
2. [bread=no]: 2 ==> [jam=no]: 2 <conf:(1)> lift:(1.25) lev:(0.05) conv:(0.44)
3. [bread=no]: 2 ==> [ghee=no]: 2 <conf:(1)> lift:(1.25) lev:(0.05) conv:(0.44)
4. [ghee=no, milk=no]: 2 ==> [jam=no]: 2 <conf:(1)> lift:(1.25) lev:(0.05) conv:(0.44)
5. [bread=no]: 2 ==> [jam=no, ghee=no]: 2 <conf:(1)> lift:(1.5) lev:(0.1) conv:(0.89)
6. [jam=no, bread=no]: 2 ==> [ghee=no]: 2 <conf:(1)> lift:(1.25) lev:(0.05) conv:(0.44)
7. [ghee=no, bread=no]: 2 ==> [jam=no]: 2 <conf:(1)> lift:(1.25) lev:(0.05) conv:(0.44)
Status
OK
Log
x 0

------→Set up the knowledge flow to load an ARFF (batch mode) and perform a cross validation using J48algorithm

1. To Create Data in .arff format

Input File: bank_data.csv

Procedure:

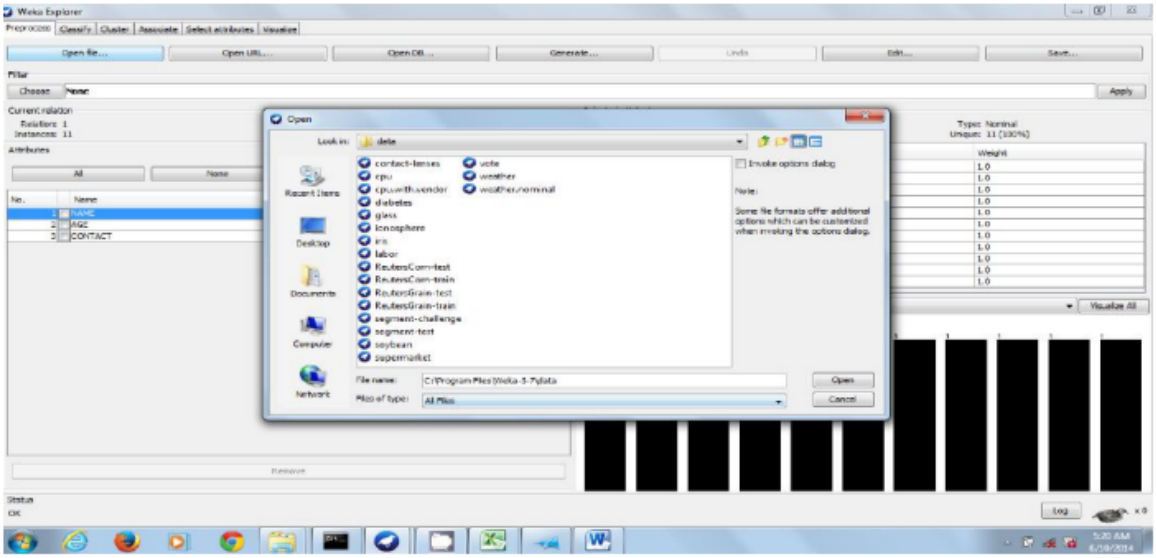
Open MS Excel. Create a new worksheet with respective headings and data.

Save the file with .csv extension

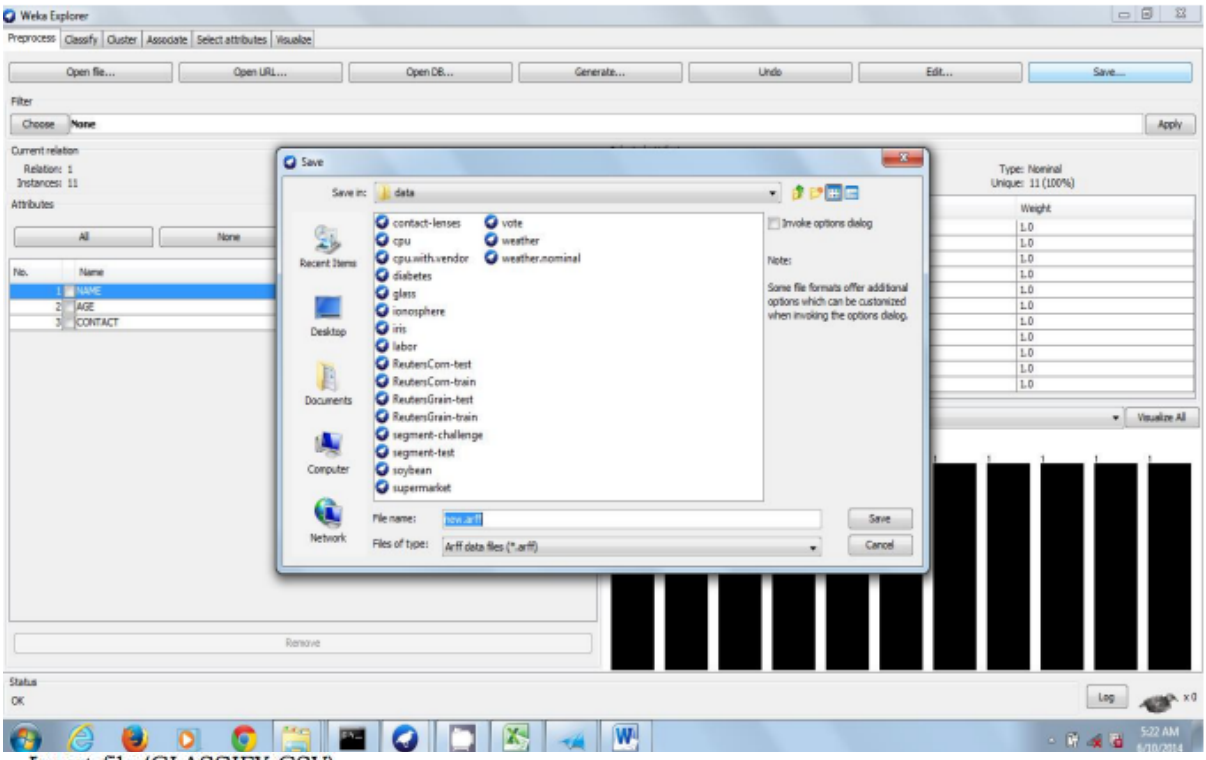
Open Preprocessor tab in explorer

Click open the file button and browse the file to open.

Load the desired .csv file using open File tab.



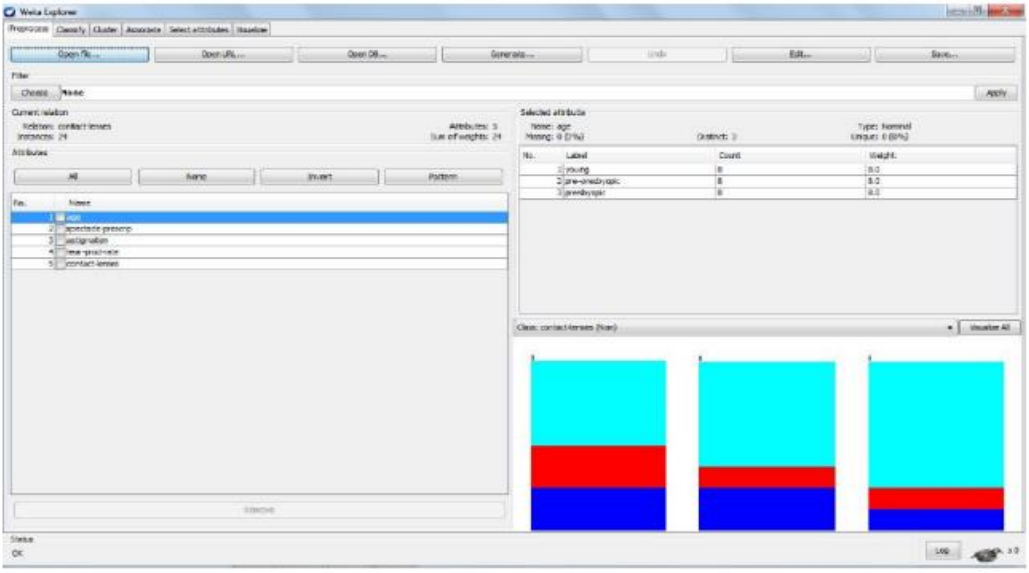
Click **SAVE** shown… dialog box opens save with extension as **.arff**



Input file (CLASSIFY.CSV)

Procedure:

In preprocessor tab, choose the input file

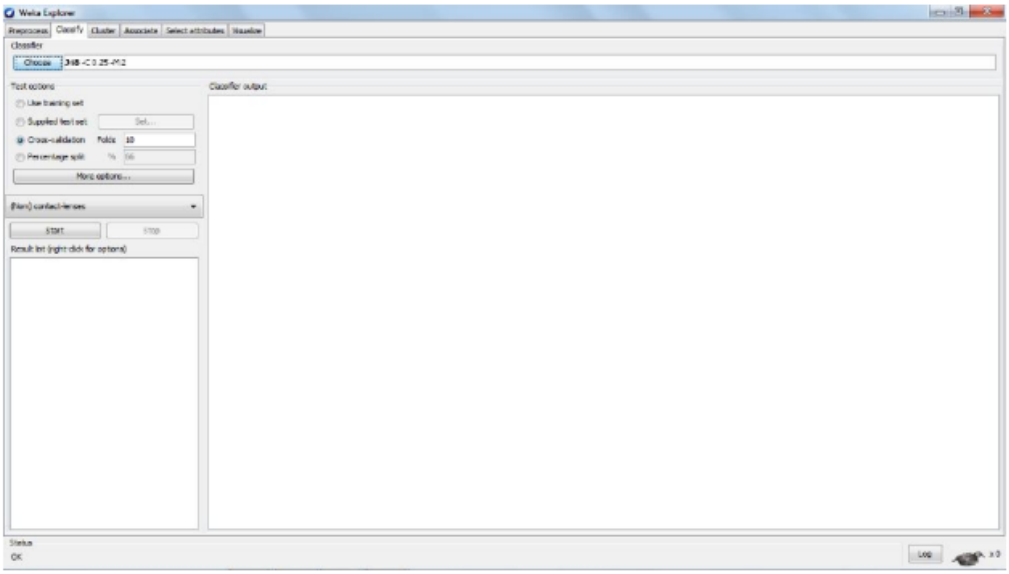


LOAD the file classify.csv

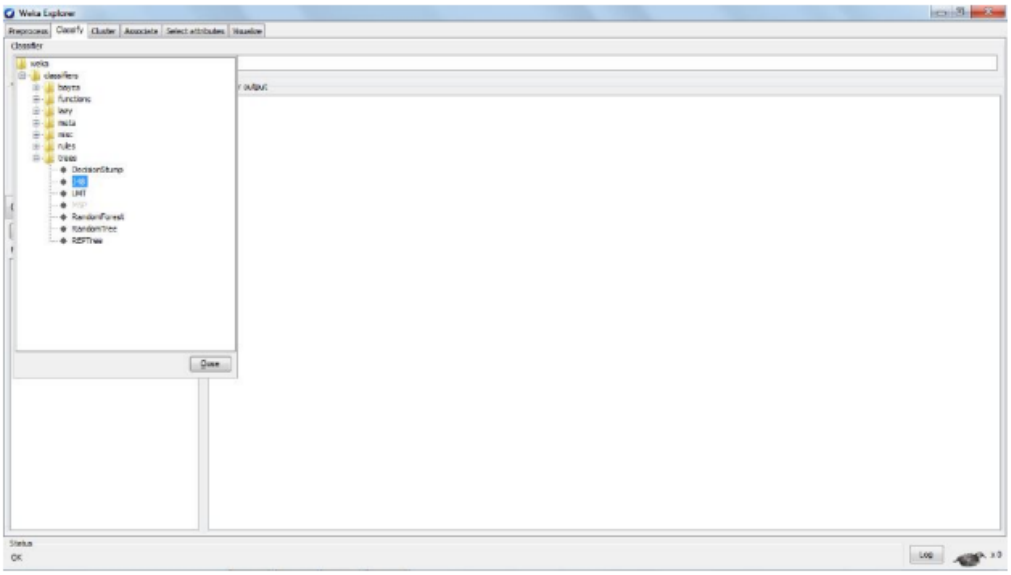
Choose the classify tab in the weka explorer window. Under the classify tab click on the choose button and select the j48 under tree as shown in the following.

LOAD the file classify.csv

Choose the classify tab in the weka explorer window. Under the classify tab click on the choose button and select the j48 under tree as shown in the following.



Select j48 algorithm (decision tree algorithm)

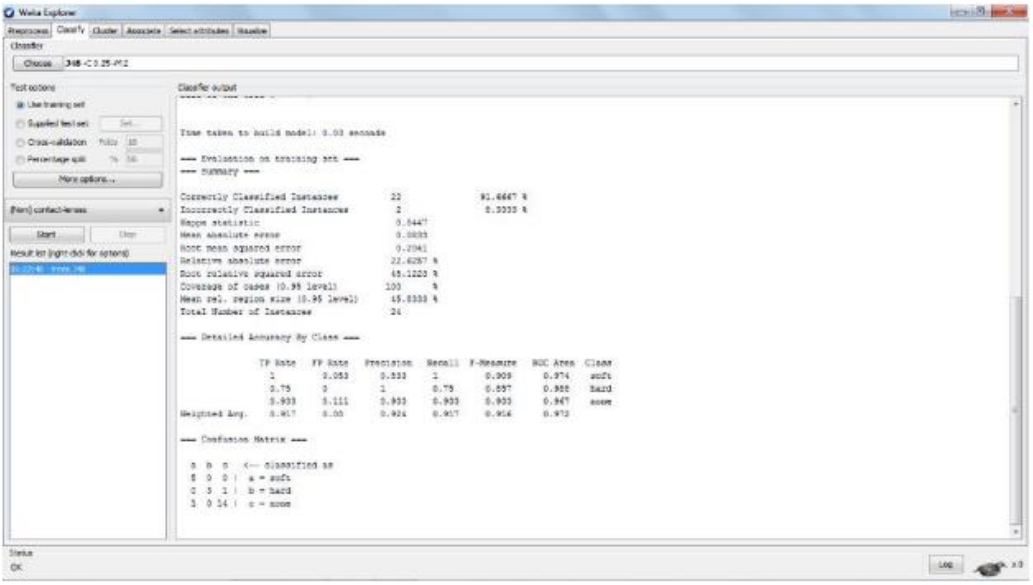


Now select the "use training set " under the test option located at the left of the weka explorer window and click on the on start button.

The output is presented in the classifier output window in weka explorer window.

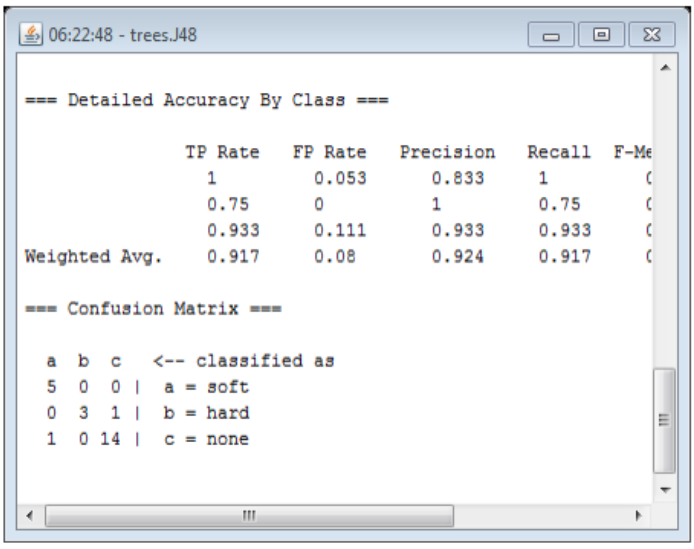
Now select the "use training set " under the test option located at the left of the weka explorer window and click on the on start button.

The output is presented in the classifier output window in weka explorer window.

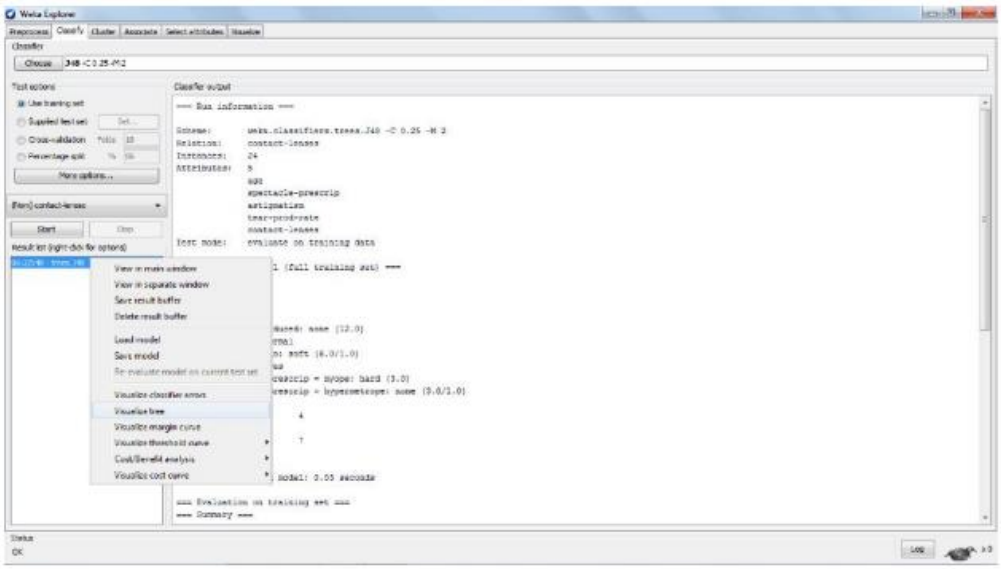


Shows output in classifier output window in weka explorer window.

We can also view the output in a separate window by right clicking on the option in result list clicking on "view in separate window"

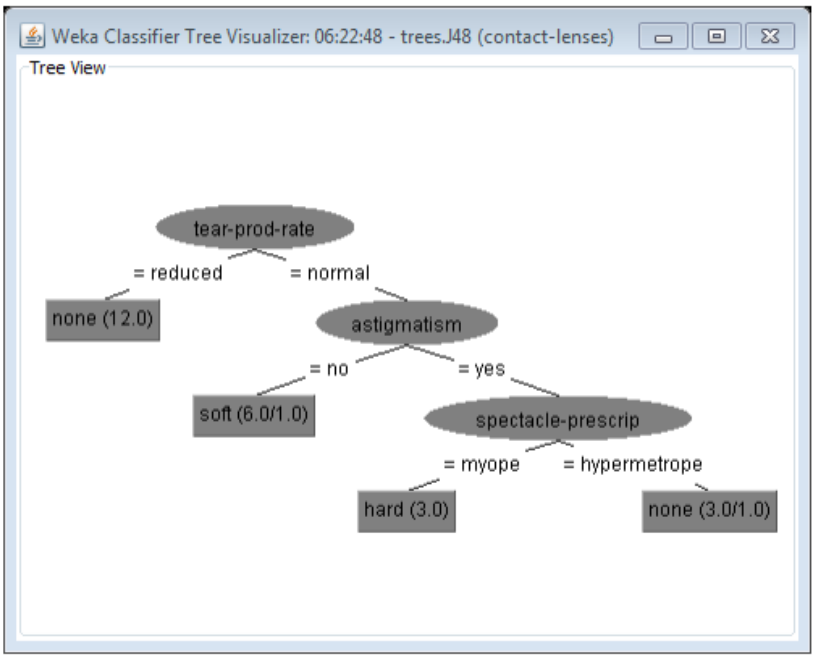


Under the result list right click on the item to get the options as shown and select the option "visualilize tree" option.



Output screen shows how to select visualize as tree option

After selecting the "visualize tree " option the output is represented as tree in a separate window shown



----->Demonstrate plotting multiple ROC curves in the same plot window by using j48 and Randomforest tree

One way to visualize the performance of classification models in machine learning is by creating a **ROC curve**, which stands for "receiver operating characteristic" curve.

Often you may want to fit several classification models to one dataset and create a ROC curve for each model to visualize which model performs best on the data.

The following step-by-step example shows how plot multiple ROC curves in Python.

Step 1: Import Necessary Packages

First, we'll import several necessary packages in Python:

```
from sklearn import metrics
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
import numpy as np
import matplotlib.pyplot as plt
```

Step 2: Create Fake Data

Next, we'll use the make_classification() function from sklearn to create a fake dataset with 1,000 rows, four predictor variables, and one binary response variable:

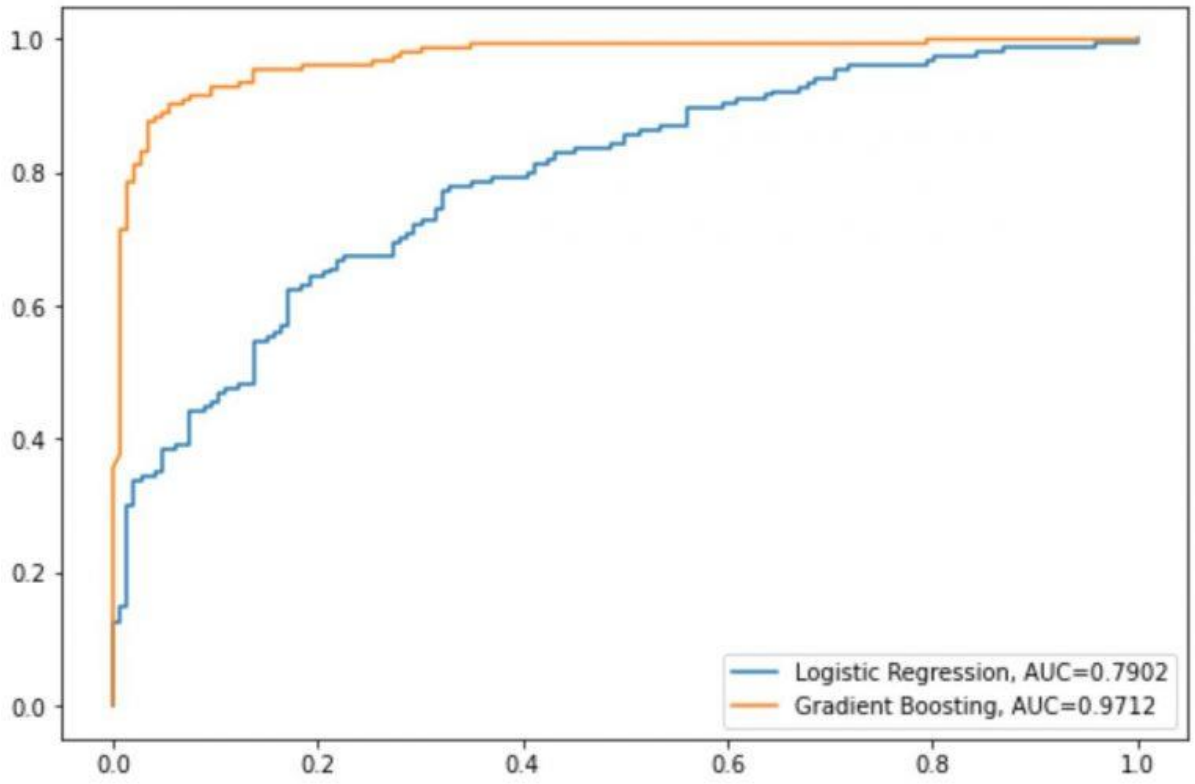
```
#create fake dataset
X, y = datasets.make_classification(n_samples=1000,
                                    n_features=4,
                                    n_informative=3,
                                    n_redundant=1,
                                    random_state=0)

#split dataset into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3,random_state=0)
```

Step 3: Fit Multiple Models & Plot ROC Curves

Next, we'll fit a logistic regression model and then a gradient boosted model to the data and plot the ROC curve for each model on the same plot:

```
#set up plotting area
plt.figure(0).clf()
```

```
#fit logistic regression model and plot ROC curve
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
plt.plot(fpr,tpr,label="Logistic Regression, AUC="+str(auc))

#fit gradient boosted model and plot ROC curve
model = GradientBoostingClassifier()
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
plt.plot(fpr,tpr,label="Gradient Boosting, AUC="+str(auc))

#add legend
plt.legend()
```



The blue line shows the ROC curve for the logistic regression model and the orange line shows the ROC curve for the gradient boosted model.

The more that a ROC curve hugs the top left corner of the plot, the better the model does at classifying the data into categories.

To quantify this, we can calculate the AUC – area under the curve – which tells us how much of the plot is located under the curve.

The closer AUC is to 1, the better the model.

From our plot we can see the following AUC metrics for each model:

- AUC of logistic regression model: **0.7902**
- AUC of gradient boosted model: **0.9712**

Clearly the gradient boosted model does a better job of classifying the data into categories compared to the logistic regression model.

WEEK-7

AIM:

Demonstrate ZeroR technique on Iris dataset (by using necessary preprocessing technique(s)) andshare your observations

What is Exploratory Data Analysis?

Exploratory Data Analysis (EDA) is a technique to analyze data using some visual Techniques. With this technique, we can get detailed information about the statistical summary of the data. We will also be able to deal with the duplicates values, outliers, and also see some trends or patterns present in the dataset.
Now let's see a brief about the Iris dataset.

Iris Dataset
If you are from a data science background you all must be familiar with the Iris Dataset. If you are not then don't worry we will discuss this here.

Iris Dataset is considered as the Hello World for data science. It contains five columns namely – Petal Length, Petal Width, Sepal Length, Sepal Width, and Species Type. Iris is a flowering plant, the researchers have measured various features of the different iris flowers and recorded them digitally.

Note: This dataset can be downloaded from here.
You can download the Iris.csv file from the above link. Now we will use the Pandas library to load this CSV file, and we will convert it into the dataframe. read_csv() method is used to read CSV files.
Example:

```
import pandas as pd

# Reading the CSV file

df = pd.read_csv("Iris.csv")

# Printing top 5 rows

df.head()
```

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Getting Information about the Dataset

We will use the shape parameter to get the shape of the dataset.

Example:
df.shape

Output:
(150, 6)

We can see that the dataframe contains 6 columns and 150 rows.

Now, let's also the columns and their data types. For this, we will use the info() method.
Example:
df.info()

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

We can see that only one column has categorical data and all the other columns are of the numeric type with non-Null entries.

Let's get a quick statistical summary of the dataset using the **describe()** method. The describe() function applies basic statistical computations on the dataset like extreme values, count of data points standard deviation, etc. Any missing value or NaN value is automatically skipped. describe() function gives a good picture of the distribution of data.
Example:
df.describe()

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

We can see the count of each column along with their mean value, standard deviation, minimum and maximum values.

Checking Missing Values

We will check if our data contains any missing values or not. Missing values can occur when no information is provided for one or more items or for a whole unit. We will use the isnull() method.

Example:

df.isnull().sum()

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

WEEK-8

AIM:

Write a java program to prepare a simulated data set with unique instances.

Description:
The number will be **unique** if it is positive integer and there are no repeated digits in the number. In other words, a number is said to be unique if and only if the digits are not duplicate. For example, 20, 56, 9863, 145, etc. are the unique numbers while 33, 121, 900, 1010, etc. are not unique numbers. In this section, we will create Java programs
To check whether the number is unique or not.

There are the following ways to check the number is unique or not:

- By Comparing Each Digit Manually

There are the following steps to check number is unique or not:

1. Read a number from the user.
2. Find the last digit o the number.
3. Compare all digits of the number with the last digit.
4. If the digit found more than one time, the number is not unique.
5. Else, eliminate the last digit of the number.
6. Repeat steps 2 to 5 until the number becomes zero.

UniqueNumberExample1.java

```
import java.util.Scanner;

public class UniqueNumberExample1
{
public static void main(String args[])
{
int r1, r2, number, num1, num2, count = 0;
Scanner sc = new Scanner(System.in);
System.out.print("Enter the number you want to check: ");
//reading a number from the user
number = sc.nextInt();
//num1 and num2 are temporary variable
num1 = number;
num2 = number;
```

```
//iterate over all digits of the number
while (num1 > 0)
{
//detrmins the last digit of the number
r1 = num1 % 10;
while (num2 > 0)
{
//finds the last digit
r2 = num2 % 10;
//comparing the last digit
if (r1 == r2)
{
//increments the count variable by 1
count++;
}
//removes the last digit from the number
num2 = num2 / 10;
}
//removes the last digit from the number
num1 = num1 / 10;
}
if (count == 1)
{
System.out.println("The number is unique.");
}
else
{
System.out.println("The number is not unique.");
}
}
}
```

Output

```
Enter the number you want to check: 13895
The number is unique.
```

WEEK-9

AIM: Write a Python program to generate frequent item sets / association rules using Apriori algorithm

Description:

Apriori Algorithm is a Machine Learning algorithm which is used to gain insight into the structured relationships between different items involved. The most prominent practical application of the algorithm is to recommend products based on the products already present in the user's cart. **Walmart** especially has made great use of the algorithm in suggesting products to it's users.
Dataset : Groceries data

Implementation of algorithm in Python:
Step 1: Importing the required libraries

```
import numpy as np

import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules
```

Step 2: Loading and exploring the data# Changing the working location to the location of the file

cd C:\Users\Dev\Desktop\Kaggle\Apriori Algorithm

 # Loading the Data

data = pd.read_excel('Online_Retail.xlsx')

data.head()

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

Exploring the columns of the data

data.columns

```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate'
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
```

Exploring the different regions of transactions

data.Country.unique()

```
array(['United Kingdom', 'France', 'Australia', 'Netherlands', 'Germany',
       'Norway', 'EIRE', 'Switzerland', 'Spain', 'Poland', 'Portugal',
       'Italy', 'Belgium', 'Lithuania', 'Japan', 'Iceland',
       'Channel Islands', 'Denmark', 'Cyprus', 'Sweden', 'Austria',
       'Israel', 'Finland', 'Bahrain', 'Greece', 'Hong Kong', 'Singapore',
       'Lebanon', 'United Arab Emirates', 'Saudi Arabia',
       'Czech Republic', 'Canada', 'Unspecified', 'Brazil', 'USA',
       'European Community', 'Malta', 'RSA'], dtype=object)
```

Step 3: Cleaning the Data

Stripping extra spaces in the description

data['Description'] = data['Description'].str.strip()

Dropping the rows without any invoice number

data.dropna(axis = 0, subset =['InvoiceNo'], inplace = True)

data['InvoiceNo'] = data['InvoiceNo'].astype('str')

Dropping all transactions which were done on credit

data = data[~data['InvoiceNo'].str.contains('C')]

Step 4: Splitting the data according to the region of transaction

Transactions done in France

basket_France = (data[data['Country'] =="France"]

.groupby(['InvoiceNo', 'Description'])['Quantity']

.sum().unstack().reset_index().fillna(0)

.set_index('InvoiceNo'))

Transactions done in the United Kingdom

basket_UK = (data[data['Country'] =="United Kingdom"]

.groupby(['InvoiceNo', 'Description'])['Quantity']

.sum().unstack().reset_index().fillna(0)

.set_index('InvoiceNo'))

```
# Transactions done in Portugal
basket_Por = (data[data['Country'] =="Portugal"]
                .groupby(['InvoiceNo', 'Description'])['Quantity']
                .sum().unstack().reset_index().fillna(0)
                .set_index('InvoiceNo'))

basket_Sweden = (data[data['Country'] =="Sweden"]
                .groupby(['InvoiceNo', 'Description'])['Quantity']
                .sum().unstack().reset_index().fillna(0)
                .set_index('InvoiceNo'))
```

Step 5: Hot encoding the Data

```
# Defining the hot encoding function to make the data suitable
# for the concerned libraries
def hot_encode(x):
        if(x<= 0):
                return 0
        if(x>= 1):
                return 1

# Encoding the datasets
basket_encoded = basket_France.applymap(hot_encode)
basket_France = basket_encoded

basket_encoded = basket_UK.applymap(hot_encode)
basket_UK = basket_encoded

basket_encoded = basket_Por.applymap(hot_encode)
basket_Por = basket_encoded
```

```
basket_encoded = basket_Sweden.applymap(hot_encode)

basket_Sweden = basket_encoded
```

Step 6: Building the models and analyzing the results

a) **France:**

```
# Building the model

frq_items = apriori(basket_France, min_support = 0.05, use_colnames = True)


# Collecting the inferred rules in a dataframe

rules = association_rules(frq_items, metric ="lift", min_threshold = 1)

rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])

print(rules.head())
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
44	(JUMBO BAG WOODLAND ANIMALS)	(POSTAGE)	0.076531	0.765306	0.076531	1.000	1.306667	0.017961	inf
258	(PLASTERS IN TIN CIRCUS PARADE, RED TOADSTOOL ...	(POSTAGE)	0.051020	0.765306	0.051020	1.000	1.306667	0.011974	inf
270	(PLASTERS IN TIN WOODLAND ANIMALS, RED TOADSTO...	(POSTAGE)	0.053571	0.765306	0.053571	1.000	1.306667	0.012573	inf
301	(SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO...	(SET/6 RED SPOTTY PAPER PLATES)	0.102041	0.127551	0.099490	0.975	7.644000	0.086474	34.897959
302	(SET/6 RED SPOTTY PAPER PLATES, SET/20 RED RET...	(SET/6 RED SPOTTY PAPER CUPS)	0.102041	0.137755	0.099490	0.975	7.077778	0.085433	34.489796

From the above output, it can be seen that paper cups and paper and plates are bought together in France. This is because the French have a culture of having a get-together with their friends and family atleast once a week. Also, since the French government has banned the use of plastic in the country, the people have to purchase the paper-based alternatives.

```
frq_items = apriori(basket_UK, min_support = 0.01, use_colnames = True)

rules = association_rules(frq_items, metric ="lift", min_threshold = 1)

rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
```

print(rules.head())

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
116	(BEADED CRYSTAL HEART PINK ON STICK)	(DOTCOM POSTAGE)	0.011036	0.037928	0.010768	0.975728	25.725872	0.010349	39.637371
2019	(SUKI SHOULDER BAG, JAM MAKING SET PRINTED)	(DOTCOM POSTAGE)	0.011625	0.037928	0.011196	0.963134	25.393807	0.010755	26.096206
2296	(HERB MARKER THYME, HERB MARKER MINT)	(HERB MARKER ROSEMARY)	0.010714	0.012375	0.010232	0.955000	77.173095	0.010099	21.947227
2302	(HERB MARKER PARSLEY, HERB MARKER ROSEMARY)	(HERB MARKER THYME)	0.011089	0.012321	0.010553	0.951691	77.240055	0.010417	20.444951
2300	(HERB MARKER THYME, HERB MARKER PARSLEY)	(HERB MARKER ROSEMARY)	0.011089	0.012375	0.010553	0.951691	76.905652	0.010416	20.443842

On analyzing the association rules for Portuguese transactions, it is observed that Tiffin sets (Knick Knack Tins) and color pencils. These two products typically belong to a primary school going kid. These two products are required by children in school to carry their lunch and for creative work respectively and hence are logically make sense to be paired together.
d) **Sweden:**

frq_items = apriori(basket_Sweden, min_support = 0.05, use_colnames = True)

rules = association_rules(frq_items, metric ="lift", min_threshold = 1)

rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])

print(rules.head())

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(12 PENCILS SMALL TUBE SKULL)	(PACK OF 72 SKULL CAKE CASES)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf
1	(PACK OF 72 SKULL CAKE CASES)	(12 PENCILS SMALL TUBE SKULL)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf
4	(36 DOILIES DOLLY GIRL)	(ASSORTED BOTTLE TOP MAGNETS)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf
5	(ASSORTED BOTTLE TOP MAGNETS)	(36 DOILIES DOLLY GIRL)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf
180	(CHILDRENS CUTLERY DOLLY GIRL)	(CHILDRENS CUTLERY CIRCUS PARADE)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf

On analyzing the above rules, it is found that boys' and girls' cutlery are paired together. This makes practical sense because when a parent goes shopping for cutlery for his/her children, he/she would want the product to be a little customized according to the kid's wishes.

WEEK-10

AIM: Write a program to calculate chi-square value using Python. Report your observation

Description:

The **Pearson's Chi-Square** statistical hypothesis is a test for independence between categorical variables. In this article, we will perform the test using a mathematical approach and then using Python's **SciPy** module.
First, let us see the mathematical approach :
The Contingency Table :
A Contingency table (also called crosstab) is used in statistics to summarise the relationship between several categorical variables. Here, we take a table that shows the number of men and women buying different types of pets.

	dog	cat	bird	total
men	207	282	241	730
women	234	242	232	708
total	441	524	473	1438

The **aim** of the test is to conclude whether the two variables (gender and choice of pet) are related to each other.
Null hypothesis:
We start by defining the **null** hypothesis (**H0**) which states that there is *no relation* between the variables. An **alternate** hypothesis would state that there is a *significant relation* between the two.
We can verify the hypothesis by these methods:

- Using **p-value**:

We define a **significance factor** to determine whether the relation between the variables is of considerable significance. Generally a significance factor or **alpha value** of **0.05** is chosen. This *alpha value* denotes the probability of erroneously rejecting **H0** when it is true. A lower *alpha value* is chosen in cases where we expect more precision. If the **p-value** for the test comes out to be strictly greater than the alpha value, then H0 holds true.

- Using **chi-square** value:

If our calculated value of chi-square is less or equal to the tabular(also called **critical**) value of chi-square, then **H0** holds true.
Expected Values

	dog	cat	bird	total
men	223.87343533	266.00834492	240.11821975	730
women	217.12656467	257.99165508	232.88178025	708
total	441	524	473	1438

Chi-Square Table :
We prepare this table by calculating for each item the following:

Table:
Next, we prepare a similar table of calculated(or expected) values. To do this we need to calculate each item in the new table as :

The expected values table :

Chi-Square Table :

We prepare this table by calculating for each item the following:

$$\frac{(Observed\_value - Calculated\_value)^2}{Calculated\_value}$$

The chi-square table:

	observed (o)	calculated (c)	(o-c)^2 / c
	207	223.87343533	1.2717579435607573
	282	266.00834492	0.9613722161954465
	241	240.11821975	0.003238139990850831
	234	217.12656467	1.3112758457617977
	242	257.99165508	0.991245364156322
	232	232.88178025	0.0033387601600580606
Total			**4.542228269825232**

From this table, we obtain the total of the last column, which gives us the calculated value of chi-square. Hence the calculated value of chi-square is **4.542228269825232**
Now, we need to find the **critical** value of chi-square. We can obtain this from a table. To use this table, we need to know the **degrees of freedom** for the dataset. The degrees of freedom is defined as : **(no. of rows – 1) * (no. of columns – 1).**
Hence, the degrees of freedom is **(2-1) * (3-1) = 2**

Now, let us look at the table and find the value corresponding to **2** degrees of freedom and **0.05** significance factor :

Critical values of the Chi-square distribution with *d* degrees of freedom

	Probability of exceeding the critical value						
d	0.05	0.01	0.001	*d*	0.05	0.01	0.001
1	3.841	6.635	10.828	11	19.675	24.725	31.264
2	5.991	9.210	13.816	12	21.026	26.217	32.910
3	7.815	11.345	16.266	13	22.362	27.688	34.528
4	9.488	13.277	18.467	14	23.685	29.141	36.123
5	11.070	15.086	20.515	15	24.996	30.578	37.697
6	12.592	16.812	22.458	16	26.296	32.000	39.252
7	14.067	18.475	24.322	17	27.587	33.409	40.790
8	15.507	20.090	26.125	18	28.869	34.805	42.312
9	16.919	21.666	27.877	19	30.144	36.191	43.820
10	18.307	23.209	29.588	20	31.410	37.566	45.315

The tabular or critical value of chi-square here is **5.991**
Hence,

$$critical\ value\ of\ \chi^2\ >=\ calculated\ value\ of\ \chi^2$$

Therefore, **H0** is **accepted**, that is, the variables ***do not*** have a significant relation.
Next, let us see how to perform the test in Python.

Performing the test using Python (scipy.stats) :

SciPy is an Open Source Python library, which is used in mathematics, engineering, scientific and technical computing.

Installation:

```
pip install scipy
```

The **chi2_contingency()** function of **scipy.stats** module takes as input, the contingency table in 2d array format. It returns a tuple containing *test statistics*, the ***p-value***, *degrees of freedom* and *expected table*(the one we created from the calculated values) in that order.

Hence, we need to compare the obtained **p-value** with **alpha** value of 0.05.

```
from scipy.stats import chi2_contingency

# defining the table
data = [[207, 282, 241], [234, 242, 232]]
stat, p, dof, expected = chi2_contingency(data)

# interpret p-value
alpha = 0.05
print("p value is " + str(p))
if p <= alpha:
    print('Dependent (reject H0)')
else:
    print('Independent (H0 holds true)')
```

Output :

```
p value is 0.1031971404730939
Independent (H0 holds true)
```

Since,

$$p\text{-}value > alpha$$

Therefore, we **accept H0,** that is, the variables ***do not*** have a significant relation.

WEEK-11

AIM: Write a program of Naive Bayesian classification using Python programming language.

Description:

Introduction to Naive Bayes

Naive Bayes is among one of the very simple and powerful algorithms for classification based on **Bayes Theorem** with an assumption of independence among the predictors. The Naive Bayes classifier assumes that the presence of a feature in a class is not related to any other feature. Naive Bayes is a classification algorithm for binary and multi-class classification problems.

Bayes Theorem

- Based on prior knowledge of conditions that may be related to an event, Bayes theorem describes the probability of the event
- conditional probability can be found this way
- Assume we have a Hypothesis(*H*) and evidence(*E*), According to Bayes theorem, the relationship between the probability of Hypothesis before getting the evidence represented as *P(H)* and the probability of the hypothesis after getting the evidence represented as *P(H|E)* is:

*P(H|E) = P(E|H)*P(H)/P(E)*

- **Prior probability** = *P(H)* is the probability before getting the evidence **Posterior probability** = *P(H|E)* is the probability after getting evidence
- In general,

*P(class|data) = (P(data|class) * P(class)) / P(data)*

Bayes Theorem Example

Assume we have to find the probability of the randomly picked card to be king given that it is a face card.
There are *4* Kings in a Deck of Cards which implies that *P(King) = 4/52*
as all the Kings are face Cards so *P(Face|King) = 1*
there are *3* Face Cards in a Suit of *13 cards* and there are *4 Suits* in total so *P(Face) = 12/52*
Therefore,
*P(King|face) = P(face|king)*P(king)/P(face) = 1/3*

```
# Importing library

import math

import random

import csv
```

```
# the categorical class names are changed to numberic data
# eg: yes and no encoded to 1 and 0
def encode_class(mydata):
        classes = []
        for i in range(len(mydata)):
                if mydata[i][-1] not in classes:
                        classes.append(mydata[i][-1])
        for i in range(len(classes)):
                for j in range(len(mydata)):
                        if mydata[j][-1] == classes[i]:
                                mydata[j][-1] = i
        return mydata


# Splitting the data
def splitting(mydata, ratio):
        train_num = int(len(mydata) * ratio)
        train = []
        # initially testset will have all the dataset
        test = list(mydata)
        while len(train) < train_num:
                # index generated randomly from range 0
                # to length of testset
                index = random.randrange(len(test))
                # from testset, pop data rows and put it in train
```

```
            train.append(test.pop(index))
        return train, test


# Group the data rows under each class yes or
# no in dictionary eg: dict[yes] and dict[no]
def groupUnderClass(mydata):
        dict = {}
        for i in range(len(mydata)):
            if (mydata[i][-1] not in dict):
                dict[mydata[i][-1]] = []
            dict[mydata[i][-1]].append(mydata[i])
        return dict


# Calculating Mean
def mean(numbers):
        return sum(numbers) / float(len(numbers))

# Calculating Standard Deviation
def std_dev(numbers):
        avg = mean(numbers)
        variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)
        return math.sqrt(variance)
```

```
def MeanAndStdDev(mydata):
        info = [(mean(attribute), std_dev(attribute)) for attribute in zip(*mydata)]
        # eg: list = [ [a, b, c], [m, n, o], [x, y, z]]
        # here mean of 1st attribute =(a + m+x), mean of 2nd attribute = (b + n+y)/3
        # delete summaries of last class
        del info[-1]
        return info

# find Mean and Standard Deviation under each class
def MeanAndStdDevForClass(mydata):
        info = {}
        dict = groupUnderClass(mydata)
        for classValue, instances in dict.items():
                info[classValue] = MeanAndStdDev(instances)
        return info


# Calculate Gaussian Probability Density Function
def calculateGaussianProbability(x, mean, stdev):
        expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
        return (1 / (math.sqrt(2 * math.pi) * stdev)) * expo


# Calculate Class Probabilities
def calculateClassProbabilities(info, test):
```

```
    probabilities = {}
    for classValue, classSummaries in info.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, std_dev = classSummaries[i]
            x = test[i]
            probabilities[classValue] *= calculateGaussianProbability(x, mean, std_dev)
    return probabilities


# Make prediction - highest probability is the prediction
def predict(info, test):
    probabilities = calculateClassProbabilities(info, test)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel


# returns predictions for a set of examples
def getPredictions(info, test):
    predictions = []
```

```
        for i in range(len(test)):
                result = predict(info, test[i])
                predictions.append(result)
        return predictions

# Accuracy score
def accuracy_rate(test, predictions):
        correct = 0
        for i in range(len(test)):
                if test[i][-1] == predictions[i]:
                        correct += 1
        return (correct / float(len(test))) * 100.0



# driver code

# add the data path in your system
filename = r'E:\user\MACHINE LEARNING\machine learning algos\Naive bayes\filedata.csv'



# load the file and store it in mydata list
mydata = csv.reader(open(filename, "rt"))
mydata = list(mydata)
mydata = encode_class(mydata)
for i in range(len(mydata)):
```

```
        mydata[i] = [float(x) for x in mydata[i]]


# split ratio = 0.7
# 70% of data is training data and 30% is test data used for testing
ratio = 0.7
train_data, test_data = splitting(mydata, ratio)
print('Total number of examples are: ', len(mydata))
print('Out of these, training examples are: ', len(train_data))
print("Test examples are: ", len(test_data))

# prepare model
info = MeanAndStdDevForClass(train_data)

# test model
predictions = getPredictions(info, test_data)
accuracy = accuracy_rate(test_data, predictions)
print("Accuracy of your model is: ", accuracy)
```

Output:

```
Total number of examples are: 200
Out of these, training examples are: 140
Test examples are: 60
Accuracy of your model is: 71.2376788
```

WEEK-12

AIM: Implement a Java program to perform Apriori algorithm

Description:

```
/**
 *
 * @author Nathan Magnus, under the supervision of Howard Hamilton
 * Copyright: University of Regina, Nathan Magnus and Su Yibin, June 2009.
 * No reproduction in whole or part without maintaining this copyright notice
 * and imposing this condition on any subsequent users.
 *
 * File:
 * Input files needed:
 *      1. config.txt - three lines, each line is an integer
 *          line 1 - number of items per transaction
 *          line 2 - number of transactions
 *          line 3 - minsup
 *      2. transa.txt - transaction file, each line is a transaction, items are separated by a space
 */

package apriori;
import java.io.*;
import java.util.*;

public class Apriori {

    public static void main(String[] args) {
        AprioriCalculation ap = new AprioriCalculation();

        ap.aprioriProcess();
    }
}
/**********************************************************************************
****
 * Class Name   : AprioriCalculation
 * Purpose      : generate Apriori itemsets

**********************************************************************************
**/
class AprioriCalculation
{
    Vector<String> candidates=new Vector<String>(); //the current candidates
    String configFile="config.txt"; //configuration file
    String transaFile="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 value per column that will be treated as a '1'
    String itemSep = " "; //the separator value for items in the database

    /*************************************************************************
     * Method Name  : aprioriProcess
     * Purpose      : Generate the apriori itemsets
     * Parameters   : None
     * Return       : None

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

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

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

        //while not complete
        do
        {
            //increase the itemset that is being looked at
            itemsetNumber++;

            //generate the candidates
            generateCandidates(itemsetNumber);

            //determine and display frequent itemsets
            calculateFrequentItemsets(itemsetNumber);
            if(candidates.size()!=0)
            {
                System.out.println("Frequent " + itemsetNumber + "-itemsets");
                System.out.println(candidates);
            }
        //if there are <=1 frequent items, then its the end. This prevents reading through the
database again. When there is only one frequent itemset.
        }while(candidates.size()>1);

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

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

   /**********************************************************************
    * Method Name  : getInput
    * Purpose      : get user input from System.in
    * Parameters   : None
    * Return       : String value of the users input

**********************************************************************/
   public static String getInput()
   {
      String input="";
      //read from System.in
      BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

      //try to get users input, if there is an error print the message
      try
      {
         input = reader.readLine();
      }
      catch (Exception e)
      {
         System.out.println(e);
      }
      return input;
   }

   /**********************************************************************
    * Method Name  : getConfig
    * Purpose      : get the configuration information (config filename, transaction filename)
    *              : configFile and transaFile will be change appropriately
    * Parameters   : None
    * Return       : None

**********************************************************************/
   private void getConfig()
   {
      FileWriter fw;
      BufferedWriter file_out;

      String input="";
      //ask if want to change the config
      System.out.println("Default Configuration: ");
      System.out.println("\tRegular transaction file with '" + itemSep + "' item separator.");
      System.out.println("\tConfig File: " + configFile);
      System.out.println("\tTransa File: " + transaFile);
      System.out.println("\tOutput File: " + outputFile);
```

```
        System.out.println("\nPress 'C' to change the item separator, configuration file and
transaction files");
        System.out.print("or any other key to continue.  ");
        input=getInput();

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

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

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

           System.out.println("Filenames changed");

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


        }

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

            //number of transactions
            numTransactions=Integer.valueOf(data_in.readLine()).intValue();

            //minsup
            minSup=(Double.valueOf(data_in.readLine()).doubleValue());

            //output config info to the user
            System.out.print("\nInput configuration: "+numItems+" items, "+numTransactions+"
transactions, ");
            System.out.println("minsup = "+minSup+"%");
```

```
            System.out.println();
            minSup/=100.0;

           oneVal = new String[numItems];
           System.out.print("Enter 'y' to change the value each row recognizes as a '1':");
           if(getInput().compareToIgnoreCase("y")==0)
           {
              for(int i=0; i<oneVal.length; i++)
              {
                 System.out.print("Enter value for column #" + (i+1) + ": ");
                 oneVal[i] = getInput();
              }
           }
           else
              for(int i=0; i<oneVal.length; i++)
                 oneVal[i]="1";

           //create the output file
           fw= new FileWriter(outputFile);
           file_out = new BufferedWriter(fw);
           //put the number of transactions into the output file
           file_out.write(numTransactions + "\n");
           file_out.write(numItems + "\n******\n");
           file_out.close();
        }
        //if there is an error, print the message
        catch(IOException e)
        {
           System.out.println(e);
        }
     }

     /**************************************************************************
      * Method Name  : generateCandidates
      * Purpose      : Generate all possible candidates for the n-th itemsets
      *              : these candidates are stored in the candidates class vector
      * Parameters   : n - integer value representing the current itemsets to be created
      * Return       : None

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

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

```
      for(int i=1; i<=numItems; i++)
      {
         tempCandidates.add(Integer.toString(i));
      }
   }
   else if(n==2) //second itemset is just all combinations of itemset 1
   {
      //add each itemset from the previous frequent itemsets together
      for(int i=0; i<candidates.size(); i++)
      {
         st1 = new StringTokenizer(candidates.get(i));
         str1 = st1.nextToken();
         for(int j=i+1; j<candidates.size(); j++)
         {
            st2 = new StringTokenizer(candidates.elementAt(j));
            str2 = st2.nextToken();
            tempCandidates.add(str1 + " " + str2);
         }
      }
   }
   else
   {
      //for each itemset
      for(int i=0; i<candidates.size(); i++)
      {
         //compare to the next itemset
         for(int j=i+1; j<candidates.size(); j++)
         {
            //create the strigns
            str1 = new String();
            str2 = new String();
            //create the tokenizers
            st1 = new StringTokenizer(candidates.get(i));
            st2 = new StringTokenizer(candidates.get(j));

            //make a string of the first n-2 tokens of the strings
            for(int s=0; s<n-2; s++)
            {
               str1 = str1 + " " + st1.nextToken();
               str2 = str2 + " " + st2.nextToken();
            }

            //if they have the same n-2 tokens, add them together
            if(str2.compareToIgnoreCase(str1)==0)
               tempCandidates.add((str1 + " " + st1.nextToken() + " " +
st2.nextToken()).trim());
         }
      }
   }
   //clear the old candidates
```

```
      candidates.clear();
      //set the new ones
      candidates = new Vector<String>(tempCandidates);
      tempCandidates.clear();
   }

   /**************************************************************************
    * Method Name  : calculateFrequentItemsets
    * Purpose      : Determine which candidates are frequent in the n-th itemsets
    *              : from all possible candidates
    * Parameters   : n - iteger representing the current itemsets being evaluated
    * Return       : None

**************************************************************************/
   private void calculateFrequentItemsets(int n)
   {
      Vector<String> frequentCandidates = new Vector<String>(); //the frequent candidates for the current itemset
      FileInputStream file_in; //file input stream
      BufferedReader data_in; //data input stream
      FileWriter fw;
      BufferedWriter file_out;

      StringTokenizer st, stFile; //tokenizer for candidate and transaction
      boolean match; //whether the transaction has all the items in an itemset
      boolean trans[] = new boolean[numItems]; //array to hold a transaction so that can be checked
      int count[] = new int[candidates.size()]; //the number of successful matches

      try
      {
            //output file
            fw= new FileWriter(outputFile, true);
            file_out = new BufferedWriter(fw);
            //load the transaction file
            file_in = new FileInputStream(transaFile);
            data_in = new BufferedReader(new InputStreamReader(file_in));

            //for each transaction
            for(int i=0; i<numTransactions; i++)
            {
               //System.out.println("Got here " + i + " times"); //useful to debug files that you are unsure of the number of line
               stFile = new StringTokenizer(data_in.readLine(), itemSep); //read a line from the file to the tokenizer
               //put the contents of that line into the transaction array
               for(int j=0; j<numItems; j++)
               {
                  trans[j]=(stFile.nextToken().compareToIgnoreCase(oneVal[j])==0); //if it is not a 0, assign the value to true
```

```
                }

                //check each candidate
                for(int c=0; c<candidates.size(); c++)
                {
                   match = false; //reset match to false
                   //tokenize the candidate so that we know what items need to be present for a
match
                   st = new StringTokenizer(candidates.get(c));
                   //check each item in the itemset to see if it is present in the transaction
                   while(st.hasMoreTokens())
                   {
                      match = (trans[Integer.valueOf(st.nextToken())-1]);
                      if(!match) //if it is not present in the transaction stop checking
                         break;
                   }
                   if(match) //if at this point it is a match, increase the count
                      count[c]++;
                }

             }
             for(int i=0; i<candidates.size(); i++)
             {
                //  System.out.println("Candidate: " + candidates.get(c) + " with count: " + count
+ " % is: " + (count/(double)numItems));
                //if the count% is larger than the minSup%, add to the candidate to the frequent
candidates
                if((count[i]/(double)numTransactions)>=minSup)
                {
                   frequentCandidates.add(candidates.get(i));
                   //put the frequent itemset into the output file
                   file_out.write(candidates.get(i) + "," + count[i]/(double)numTransactions +
"\n");
                }
             }
             file_out.write("-\n");
             file_out.close();
       }
       //if error at all in this process, catch it and print the error messate
       catch(IOException e)
       {
          System.out.println(e);
       }
       //clear old candidates
       candidates.clear();
       //new candidates are the old frequent candidates
       candidates = new Vector<String>(frequentCandidates);
       frequentCandidates.clear();
    }
}
```

Output:

```
hercules[71]% java apriori
Algorithm apriori starting now.....
Press 'C' to change the default configuration and transaction files
or any other key to continue.
Input configuration: 5 items, 5 transactions, minsup = 40%
Frequent 1-itemsets:
[1, 2, 3, 4, 5]
Frequent 2-itemsets:
[1 2, 1 3, 1 4, 1 5, 2 3, 2 4, 3 4, 3 5, 4 5]
Frequent 3-itemsets:
[1 2 3, 1 2 4, 1 3 4, 1 3 5, 1 4 5, 2 3 4, 3 4 5]
Frequent 4-itemsets:
[1 2 3 4, 1 3 4 5]
Execution time is: 0 seconds.
hercules[72]%
```

WEEK-13

AIM:

Write a program to cluster your choice of data using simple k-means algorithm using JDK.

Clustering is a popular data mining task. It involves the division of data items into a number of meaningful groups or clusters. Clustering helps us to identify any underlying relationships between data items, which is good for decision making.

Although there are many clustering algorithms, K-Means is the most popular and simplest clustering algorithm. It works by dividing a data set into k groups. Similar items are put in the same group while dissimilar items are put into different groups. In this article, we will be discussing how to implement the K-Means algorithm using Java and GridDB.

What is K-Means Algorithm?

K-Means is a clustering algorithm that assigns data items to their nearest clusters. It requires you to specify one property, the number of clusters, usually denoted as k.

The following pseudocode describes how the algorithm works:

```
Step 1: Select `k` initial centroids.

REPEAT:

    Step 2: Create `k` clusters by assigning each data point to the nearest cluster centroid.

    Step 3: Recompute the new centroids for each cluster.

Until the centroids don't change.
```

The user is required to specify the value of k, which is the number of clusters to be formed. The algorithm then selects k observations from the dataset to be the centroids for the clusters.

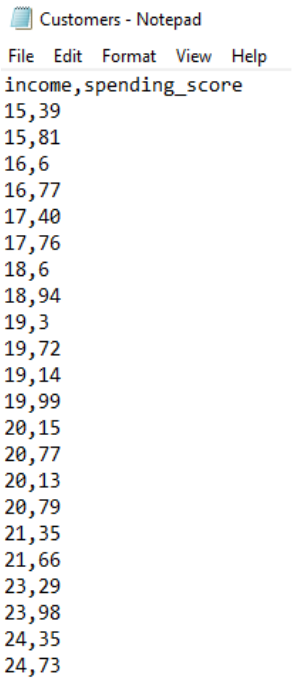
The next step involes going through all the data items and assorting them clusters, while ensuring that every observation is assigned to its closest centroid.

The new centroids for the clusters are then recomputed. This is done by getting the mean of all the observations in a cluster, and the result of the operation becomes the new centroid. The above steps are then repeated until the centroids don't change any more.

Next, I will be showing you how to implement the K-Means algorithm in Java. We will be using GridDB as the storage database.

Implementing K-Means in Java

In this section, we will be implementing the K-Means algorithm using Java and GridDB. We will use a dataset that shows the annual income of different individuals and their corresponding spending scores.



We will first write the data into GridDB, and then pull it from there for analysis using the K-Means algorithm.

Import Packages

First, let's import the packages that we will need to use:

```
import java.io.IOException;
import java.util.Collection;
import java.util.Properties;
import java.util.Scanner;

import com.toshiba.mwcloud.gs.Collection;
import com.toshiba.mwcloud.gs.GSException;
import com.toshiba.mwcloud.gs.GridStore;
import com.toshiba.mwcloud.gs.GridStoreFactory;
import com.toshiba.mwcloud.gs.Query;
```

```
import com.toshiba.mwcloud.gs.RowKey;
import com.toshiba.mwcloud.gs.RowSet;
public static class Customers{

        @RowKey String income;
    String spending_score;

   }
Properties props = new Properties();
       props.setProperty("notificationAddress", "239.0.0.1");
       props.setProperty("notificationPort", "31999");
       props.setProperty("clusterName", "defaultCluster");
       props.setProperty("user", "admin");
       props.setProperty("password", "admin");
       GridStore store = GridStoreFactory.getInstance().getGridStore(props);
Collection<String, Customers> coll = store.putCollection("col01", Customers.class);
File file1 = new File("Customers.csv");
              Scanner sc = new Scanner(file1);
              String data = sc.next();

              while (sc.hasNext()){
                    String scData = sc.next();
                    String dataList[] = scData.split(",");
                    String income = dataList[0];
                    String spending_score = dataList[1];


                    Customers customers = new Customers();

                    customers.income = income;
                    customers.spending_score = spending_score;
                    coll.append(customers);
               }
Query<customers> query = coll.query("select *");
              RowSet</customers><customers> rs = query.fetch(false);
          RowSet res = query.fetch();</customers>
int x,j,k=2;
       int cluster1[][] = new int[18][10];
       int cluster2[][] = new int[20][80];
       float mean1[][] = new float[1][2];
       float mean2[][] = new float[1][2];
       float temp1[][] = new float[1][2], temp2[][] = new float[1][2];
       int sum11 = 0, sum12 = 0, sum21 = 0, sum22 = 0;
       double dist1, dist2;
       int i1 = 0, i2 = 0, itr = 0;


       System.out.println("\nNumber of clusters: "+k);

       // Set random means
```

```
mean1[0][0] = 18;
mean1[0][1] = 10;
mean2[0][0] = 20;
mean2[0][1] = 80;

// Loop untill the new mean and previous mean are the same
while(!Arrays.deepEquals(mean1, temp1) || !Arrays.deepEquals(mean2, temp2)) {

   //Empty the partitions
   for(x=0;x<10;x++) {
      cluster1[x][0] = 0;
      cluster1[x][1] = 0;
      cluster2[x][0] = 0;
      cluster2[x][1] = 0;
   }

   i1 = 0; i2 = 0;

   //Find the distance between mean and the data point and store it in its corresponding
partition
   for(x=0;x<10;x++) {
      dist1 = Math.sqrt(Math.pow(res[x][0] - mean1[0][0],2) + Math.pow(res[x][1] -
mean1[0][1],2));
      dist2 = Math.sqrt(Math.pow(res[x][0] - mean2[0][0],2) + Math.pow(res[x][1] -
mean2[0][1],2));

      if(dist1 < dist2) {
         cluster1[i1][0] = res[x][0];
         cluster1[i1][1] = res[x][1];

         i1++;
      }
      else {
         cluster2[i2][0] = res[x][0];
         cluster2[i2][1] = res[x][1];

         i2++;
      }
   }

   //Store the previous mean
   temp1[0][0] = mean1[0][0];
   temp1[0][1] = mean1[0][1];
   temp2[0][0] = mean2[0][0];
   temp2[0][1] = mean2[0][1];

   //Find the new mean for new partitions
   sum11 = 0; sum12 = 0; sum21 = 0; sum22 = 0;

   for(x=0;x<i1;x++) {
```

```
            sum11 += cluster1[x][0];
            sum12 += cluster1[x][1];
        }
        for(x=0;x<i2;x++) {
            sum21 += cluster2[x][0];
            sum22 += cluster2[x][1];
        }
        mean1[0][0] = (float)sum11/i1;
        mean1[0][1] = (float)sum12/i1;
        mean2[0][0] = (float)sum21/i2;
        mean2[0][1] = (float)sum22/i2;

        itr++;
    }

    System.out.println("Cluster1:");
    for(x=0;x<i1;x++) {
        System.out.println(cluster1[x][0]+" "+cluster1[x][1]);
    }
    System.out.println("\nCluster2:");
    for(x=0;x<i2;x++) {
        System.out.println(cluster2[x][0]+" "+cluster2[x][1]);
    }
    System.out.println("\nFinal Mean: ");
    System.out.println("Mean1 : "+mean1[0][0]+" "+mean1[0][1]);
    System.out.println("Mean2 : "+mean2[0][0]+" "+mean2[0][1]);
    System.out.println("\nTotal Iterations: "+itr);


}

}
```

OUTPUT

```
Number of clusters: 2

Cluster1:

15 39

16 6

17 40
```

```
18 6

19 3



Cluster2:

15 89

16 77

17 76

18 94

19 72



Final Mean:

Mean1 : 17.0 18.8

Mean2 : 17.0 81.6



Total Iterations: 2
```

WEEK-14

AIM:

Write a program of cluster analysis using simple k-means algorithm Python programming language

Description:

Description:: **K-Means Clustering: Concepts and Implementation in R for DataScience**

Clustering is a technique in machine learning that attempts to find *clusters* of observations within a dataset.

The goal is to find clusters such that the observations within each cluster are quite similar to each other, whileobservations in different clusters are quite different from each other.

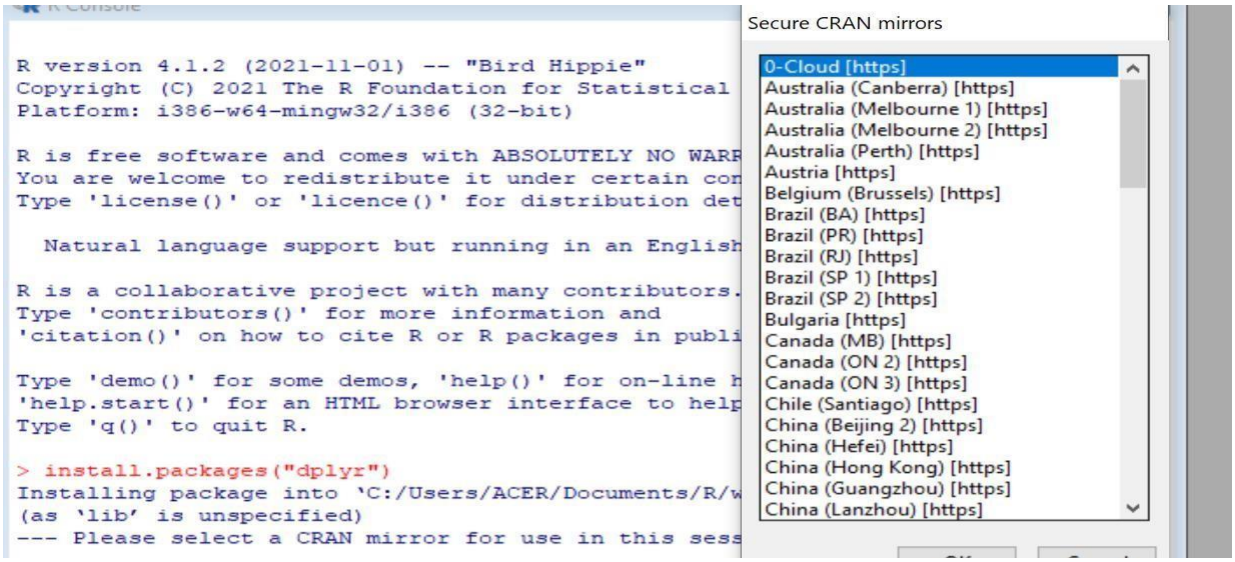
Clustering is a form of unsupervised learning because we're simply attempting to find structure within a dataset ratherthan predicting the value of some response variable.

Clustering is often used in marketing when companies have access to information like:

- Household income
- Household size
- Head of household Occupation
- Distance from nearest urban area

When this information is available, clustering can be used to identify households that are similar and may be more likely to purchase certain products or respond better to a certain type of advertising.

One of the most common forms of clustering is known as **k-means clustering**.



```
> install.packages("ggplot2")
Installing package into 'C:/Users/ACER/Documents/R/win-library/4.1'
(as 'lib' is unspecified)
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.1/ggplot2_3.3.5.z$
Content type 'application/zip' length 4130482 bytes (3.9 MB)
downloaded 3.9 MB

package 'ggplot2' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
```

```
> library("ggplot2")
Need help getting started? Try the R Graphics Cookbook:
https://r-graphics.org
> library("dplyr")

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> library("ggfortify")
>
```

```
> install.packages("ggfortify")
Installing package into 'C:/Users/ACER/Documents/R/win-library/4.1'
(as 'lib' is unspecified)
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.1/ggfortify_0.4.1$
Content type 'application/zip' length 2369968 bytes (2.3 MB)
downloaded 2.3 MB

package 'ggfortify' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\ACER\AppData\Local\Temp\RtmpecgYDt\downloaded_packages
```

The end goal is to have *K* clusters in which the observations within each cluster are quite similar to each other while the observations in different clusters are quite different from each other.

In practice, we use the following steps to perform K-means clustering:

1. **Choose a value for *K*.**

- First, we must decide how many clusters we'd like to identify in the data. Often we have to simply test several different values for *K* and analyze the results to see which number of clusters seems to make themost sense for a given problem.

2. **Randomly assign each observation to an initial cluster, from 1 to *K*.**

3. **Perform the following procedure until the cluster assignments stop changing.**

- For each of the *K* clusters, compute the cluster *centroid*. This is simply the vector of the *p* feature means forthe observations in the *k*th cluster.
- Assign each observation to the cluster whose centroid is closest. Here, *closest* is defined using Euclideandistance.

K-Means Clustering in R

The following tutorial provides a step-by-step example of how to perform k-means clustering in R.

Step 1: Load the Necessary Packages

First, we'll load two packages that contain several useful functions for k-means clustering in R

```
> library(factoextra)
```

Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa

```
> library(cluster)
```

Step 2: Load and Prep the Data

For this example we'll use the *USArrests* dataset built into R, which contains the number of arrests per 100,000 residents in each U.S. state in 1973 for *Murder*, *Assault*, and *Rape* along with the percentage of the population ineach state living in urban areas, *UrbanPop*.

The following code shows how to do the following:

- Load the *USArrests* dataset
- Remove any rows with missing values
- Scale each variable in the dataset to have a mean of 0 and a standard deviation of 1The following code shows how to do the following:

- Load the *USArrests* dataset
- Remove any rows with missing values
- Scale each variable in the dataset to have a mean of 0 and a standard deviation of 1


```
> library(factoextra)
Welcome! Want to learn more? See two factoextra-related books at https://goo.gl$
> library(cluster)
> df <- USArrests
> df <- na.omit(df)
> df <- scale(df)
> df <- scale(df)
> head(df)
               Murder   Assault   UrbanPop         Rape
Alabama    1.24256408 0.7828393 -0.5209066 -0.003416473
Alaska     0.50786248 1.1068225 -1.2117642  2.484202941
Arizona    0.07163341 1.4788032  0.9989801  1.042878388
Arkansas   0.23234938 0.2308680 -1.0735927 -0.184916602
California 0.27826823 1.2628144  1.7589234  2.067820292
Colorado   0.02571456 0.3988593  0.8608085  1.864967207
> |
```

Step 3: Find the Optimal Number of Clusters

To perform k-means clustering in R we can use the built-in **kmeans()** function, which uses the following syntax:

kmeans(data, centers, nstart)

where:

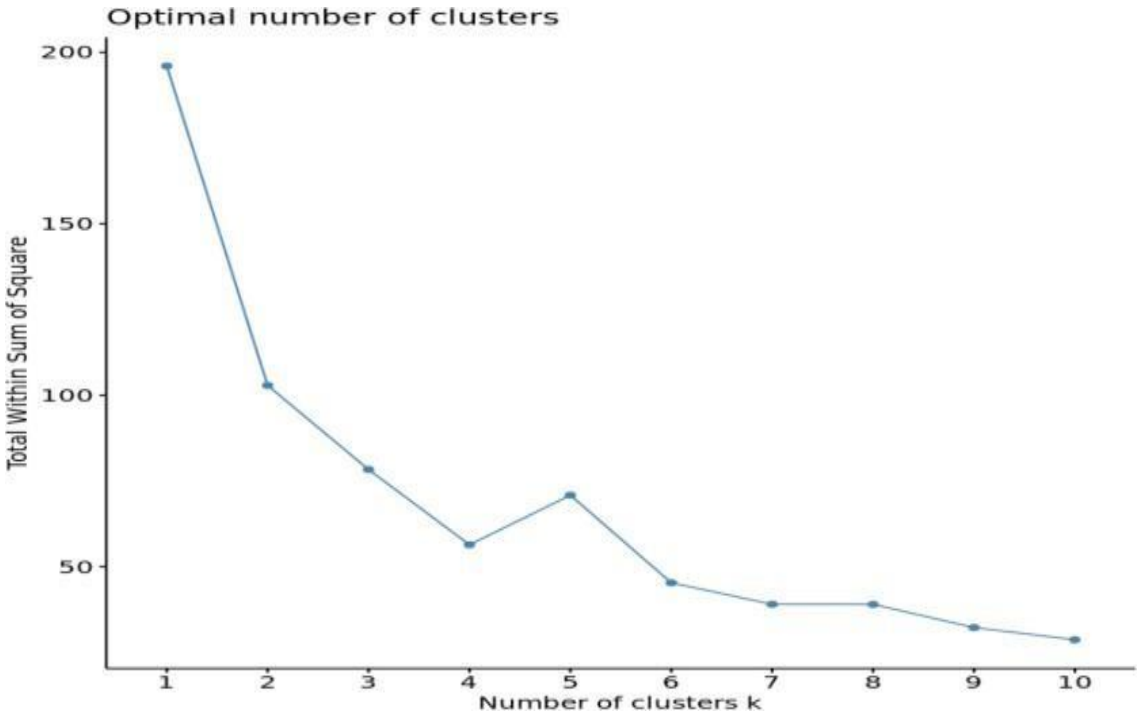
- **data:** Name of the dataset.
- **centers:** The number of clusters, denoted *k*.
- **nstart:** The number of initial configurations. Because it's possible that different initial starting clusters can lead to different results, it's recommended to use several different initial configurations. The k-means algorithm will find the initial configurations that lead to the smallest within-cluster variation.

Since we don't know beforehand how many clusters is optimal, we'll create two different plots that can help usdecide:

1. Number of Clusters vs. the Total Within Sum of Squares

First, we'll use the **fviz_nbclust()** function to create a plot of the number of clusters vs. the total within sum ofsquares:

```
> fviz_nbclust(df, kmeans, method = "wss")
```



Typically when we create this type of plot we look for an “elbow” where the sum of squares begins to “bend” or leveloff. This is typically the optimal number of clusters.

For this plot it appear that there is a bit of an elbow or “bend” at k = 4 clusters.

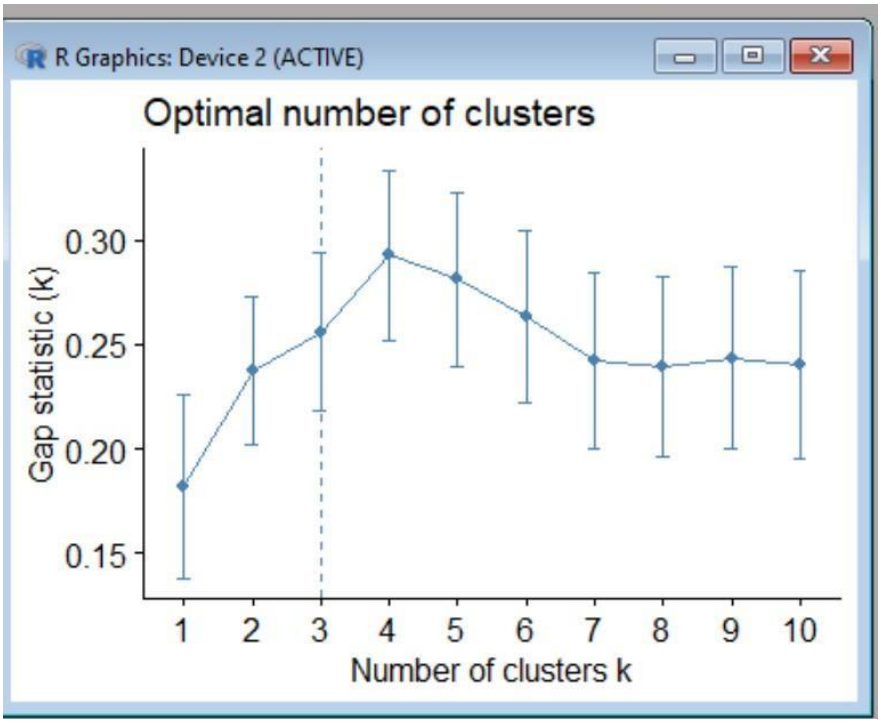
2. Number of Clusters vs. Gap Statistic

Another way to determine the optimal number of clusters is to use a metric known as the gap statistic, which compares the total intra-cluster variation for different values of k with their expected values for a distribution with noclustering.

We can calculate the gap statistic for each number of clusters using the **clusGap()** function from the *cluster* package along with a plot of clusters vs. gap statistic using the **fviz_gap_stat()** function

#calculate gap statistic based on number of clusters

```
> gap_stat <- clusGap(df,
+                     FUN = kmeans,
+                     nstart = 25,
+                     K.max = 10,
+                     B = 50)
Clustering k = 1,2,..., K.max (= 10): .. done
Bootstrapping, b = 1,2,..., B (= 50)  [one "." per sample]:
.................................................. 50
> fviz_gap_stat(gap_stat)
>
```



From the plot we can see that gap statistic is highest at k = 4 clusters, which matches the elbow method we usedearlier.

Step 4: Perform K-Means Clustering with Optimal *K*

Lastly, we can perform k-means clustering on the dataset using the optimal value for *k* of 4:

```
> set.seed(1)
> km <- kmeans(df, centers = 4, nstart = 25)
>
> km
```

```
#make this example reproducibleset.seed(1)

#perform k-means clustering with k = 4 clusterskm <- kmeans(df, centers = 4, nstart = 25)
#view resultskm

K-means clustering with 4 clusters of sizes 16, 13, 13, 8

Cluster means:
      Murder    Assault   UrbanPop        Rape
1 -0.4894375 -0.3826001  0.5758298 -0.26165379
2 -0.9615407 -1.1066010 -0.9301069 -0.96676331
3  0.6950701  1.0394414  0.7226370  1.27693964
4  1.4118898  0.8743346 -0.8145211  0.01927104

Clustering vector:
```

Alabama	Alaska	Arizona	Arkansas	California	Colorado

4	**3**	**3**	**4**	**3**	**3**
Connecti	**Delawa**	**Florida**	**Georgia**	**Hawaii**	**Idaho**

cut	re				
1	1	3	4	1	2
Illinois	Indiana	Iowa	Kansas	Kentucky	Louisiana
3	1	2	1	2	4
Maine	Maryland	Massachusetts	Michigan	Minnesota	Mississippi
2	3	1	3	2	4
Missouri	Montana	Nebraska	Nevada	New Hampshire	New Jersey
3	2	2	3	2	1
New Mexico	New York	North Carolina	North Dakota	Ohio	Oklahoma

WEEK-15
AIM:

Write a program to compute/display dissimilarity matrix (for your own dataset containing at least fourinstances with two attributes) using Python.

Description:

Dissimilarity and distance matrices (**skbio.core.distance**)

This module provides functionality for serializing, deserializing, and manipulating dissimilarity and distance matrices in memory. There are two matrix classes available, DissimilarityMatrix and DistanceMatrix. Both classes can store measures of difference/distinction between objects. A dissimilarity/distance matrix includes both a matrix of dissimilarities/distances (floats) between objects, as well as unique IDs (object labels; strings) identifying each object in the matrix.

DissimilarityMatrix can be used to store measures of dissimilarity between objects, and does not require that the dissimilarities are symmetric (e.g., dissimilarities obtained using the *Gain in PD* measure [R1]). DissimilarityMatrix is a more general container to store differences than DistanceMatrix.

DistanceMatrix has the additional requirement that the differences it stores are symmetric (e.g., Euclidean or Hamming distances).

Note

DissimilarityMatrix can be used to store distances, but it is recommended to use DistanceMatrix to store this type of data as it provides an additional check for symmetry. A distance matrix is a dissimilarity matrix; this is modeled in the class design by having DistanceMatrix as a subclass of DissimilarityMatrix.

Classes

DissimilarityMatrix	Store dissimilarities between objects.
DistanceMatrix(data, ids)	Store distances between objects.

Functions

randdm(num_objects[, ids, constructor, ...])	Generate a distance matrix populated with random distances.

References

[R1] Faith, D. P. (1992). “Conservation evaluation and phylogenetic diversity”.

Examples

Assume we have the following delimited text file storing distances between three objects with IDs `a`, `b`, and `c`:

```
\ta\tb\tc
a\t0.0\t0.5\t1.0
b\t0.5\t0.0\t0.75
c\t1.0\t0.75\t0.0
```

Load a distance matrix from the file:

```
>>> from StringIO import StringIO
>>> from skbio.core.distance import DistanceMatrix
>>> dm_f = StringIO("\ta\tb\tc\n"
...                 "a\t0.0\t0.5\t1.0\n"
...                 "b\t0.5\t0.0\t0.75\n"
...                 "c\t1.0\t0.75\t0.0\n")
>>> dm = DistanceMatrix.from_file(dm_f)
>>> print dm
3x3 distance matrix
IDs:
a, b, c
Data:
[[ 0.    0.5   1.  ]
 [ 0.5   0.    0.75]
 [ 1.    0.75  0.  ]]
```

Access the distance (scalar) between objects `'a'` and `'c'`:

```
>>> dm['a', 'c']
1.0
```

Get a row vector of distances between object `'b'` and all other objects:

>>>

```
>>> dm['b']
array([ 0.5 ,  0.  ,  0.75])
```

numpy indexing/slicing also works as expected. Extract the third column:

```
>>> dm[:, 2]
array([ 1.  ,  0.75,  0.  ])
```

Serialize the distance matrix to delimited text file

```
>>> out_f = StringIO()
>>> dm.to_file(out_f)
>>> out_f.getvalue()
'\ta\tb\tc\na\t0.0\t0.5\t1.0\nb\t0.5\t0.0\t0.75\nc\t1.0\t0.75\t0.0\n'
>>> out_f.getvalue() == dm_f.getvalue()
True
```

A distance matrix object can also be created from an existing `numpy.array` (or an array-like object, such as a nested Python list):

```
>>> import numpy as np
>>> data = np.array([[0.0, 0.5, 1.0],
...                  [0.5, 0.0, 0.75],
...                  [1.0, 0.75, 0.0]])
>>> ids = ["a", "b", "c"]
>>> dm_from_np = DistanceMatrix(data, ids)
>>> print dm_from_np
3x3 distance matrix
IDs:
a, b, c
Data:
[[ 0.    0.5   1.  ]
 [ 0.5   0.    0.75]
 [ 1.    0.75  0.  ]]
>>> dm_from_np == dm
True
```

WEEK-16

AIM: Visualize the datasets using matplotlib in python.(Histogram, Box plot, Bar chart, Pie chartetc.,)

Description:

What is matplotlib?

"Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits."

You can draw up all sorts of charts and visualization using matplotlib. I will be exploring the most common plots in the matplotlib Python library in this tutorial. We will first understand the dataset at hand and then start building different plots using matplotlib, including scatterplots and line charts!

matplotlib – The Most Popular Python Library for Data Visualization and Exploration

I love working with matplotlib in Python. It was the first visualization library I learned to master and it has stayed with me ever since. There is a reason why matplotlib is the most popular Python library for data visualization and exploration – the flexibility and agility it offers is unparalleled!

Matplotlib provides an easy but comprehensive visual approach to present our findings. There are a number of visualizations we can choose from to present our results, as we'll soon see in this tutorial.

From histograms to scatterplots, matplotlib lays down an array of colors, themes, palettes, and other options to customize and personalize our plots. matplotlib is useful whether you're performing data exploration for a machine learning project or simply want to create dazzling and eye-catching charts.

Note: If you're new to the world of Python, we highly recommend taking the below popular free courses:

What is matplotlib?

Let's put a formal definition to matplotlib before we dive into the crux of the article. If this is the first time you've heard of matplotlib, here's the official description:

"Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits."

You can draw up all sorts of charts and visualization using matplotlib. I will be exploring the most common plots in the matplotlib Python library in this tutorial. We will first understand the dataset at hand and then start building different plots using matplotlib, including scatterplots and line charts!

Note: If you're looking for a matplotlib alternative or want to explore other Python visualization libraries, check out the below tutorial on Seaborn:

Here are the Visualization We'll Design using matplotlib

- Bar Graph
- Pie Chart
- Box Plot
- Histogram
- Line Chart and Subplots
- Scatter Plot

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn')
df_meal = pd.read_csv('C:\\Users\Dell\\Desktop\\train_food\\meal_info.csv')
df_meal.head()
```

	meal_id	category	cuisine
0	1885	Beverages	Thai
1	1993	Beverages	Thai
2	2539	Beverages	Thai
3	1248	Beverages	Indian
4	2631	Beverages	Indian

```
df_center =
```

```
pd.read_csv('C:\\Users\Dell\\Desktop\\train_food\\fulfilment_center_info.csv')
df_center.head()
```

	center_id	city_code	region_code	center_type	op_area
0	11	679	56	TYPE_A	3.7
1	13	590	56	TYPE_B	6.7
2	124	590	56	TYPE_C	4.0
3	66	648	34	TYPE_A	4.1
4	94	632	34	TYPE_C	3.6

```
df_food = pd.read_csv('C:\\Users\Dell\\Desktop\\train_food\\train_food.csv')
df_food.head()
```

	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders
0	1379560	1	55	1885	136.83	152.29	0	0	177
1	1466964	1	55	1993	136.83	135.83	0	0	270
2	1346989	1	55	2539	134.86	135.86	0	0	189
3	1338232	1	55	2139	339.50	437.53	0	0	54
4	1448490	1	55	2631	243.50	242.50	0	0	40

```
df = pd.merge(df_food,df_center,on='center_id')
df = pd.merge(df,df_meal,on='meal_id')
```

matplotlib – The Most Popular Python Library for Data Visualization and Exploration

I love working with matplotlib in Python. It was the first visualization library I learned to master and it has stayed with me ever since. There is a reason why matplotlib is the most popular Python library for data visualization and exploration – the flexibility and agility it offers is unparalleled!

Matplotlib provides an easy but comprehensive visual approach to present our findings. There are a number of visualizations we can choose from to present our results, as we'll soon see in this tutorial.

From histograms to scatterplots, matplotlib lays down an array of colors, themes, palettes, and other options to customize and personalize our plots. matplotlib is useful whether you're performing data exploration for a machine learning project or simply want to create dazzling and eye-catching charts.

Note: If you're new to the world of Python, we highly recommend taking the below popular free courses:

What is matplotlib?

Let's put a formal definition to matplotlib before we dive into the crux of the article. If this is the first time you've heard of matplotlib, here's the official description:

"Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits."

You can draw up all sorts of charts and visualization using matplotlib. I will be exploring the most common plots in the matplotlib Python library in this tutorial. We will first understand the dataset at hand and then start building different plots using matplotlib, including scatterplots and line charts!

Note: If you're looking for a matplotlib alternative or want to explore other Python visualization libraries, check out the below tutorial on Seaborn:

- *Become a Data Visualization Whiz with this Comprehensive Guide to Seaborn in Python*

Here are the Visualization We'll Design using matplotlib

- Bar Graph
- Pie Chart
- Box Plot
- Histogram
- Line Chart and Subplots
- Scatter Plot

Understanding the Dataset and the Problem Statement

Before we get into the different visualizations and chart types, I want to spend a few minutes understanding the data. This is a critical part of the machine learning pipeline and we should pay full attention to it.

We will be analyzing the Food Demand Forecasting project in this matplotlib tutorial. The aim of this project is to predict the number of food orders that customers will place in the upcoming weeks with the company. We will, of course, only spend time on the exploration stage of the project.

Let us first import the relevant libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn')
```

view rawMatplotlib_1.py hosted with by **GitHub**

*I have used a matplotlib stylesheet to make our plots look neat and pretty. Here, I have used the **'seaborn'** stylesheet. However, there are plenty of other stylesheets in Matplotlib which you can use to best suit your presentation style.*

Our dataset has three dataframes: ***df_meal*** describing the meals, ***df_center*** describing the food centers, and ***df_food*** describing the overall food order. Have a look at them below:

```
df_meal = pd.read_csv('C:\\Users\Dell\\Desktop\\train_food\\meal_info.csv')
df_meal.head()
```

view rawMatplotlib_2.py hosted with by **GitHub**

```
df_center = pd.read_csv('C:\\Users\Dell\\Desktop\\train_food\\fulfilment_center_info.csv')
df_center.head()
```

view rawMatplotlib_3.py hosted with by **GitHub**

```
df_food = pd.read_csv('C:\\Users\Dell\\Desktop\\train_food\\train_food.csv')
df_food.head()
```

view rawMatplotlib_4.py hosted with by **GitHub**

I will first merge all the three dataframes into a single dataframe. This will make it easier to manipulate the data while plotting it:

```
df = pd.merge(df_food,df_center,on='center_id')
df = pd.merge(df,df_meal,on='meal_id')
```

view rawMatplotlib_5.py hosted with by **GitHub**

Right – now let's jump into the different chart types we can create using matplotlib in Python!

1. Bar Graph using matplotlib

First, we want to find the most popular food item that customers have bought from the company.

I will be using the Pandas ***pivot_table*** function to find the total number of orders for each category of the food item:

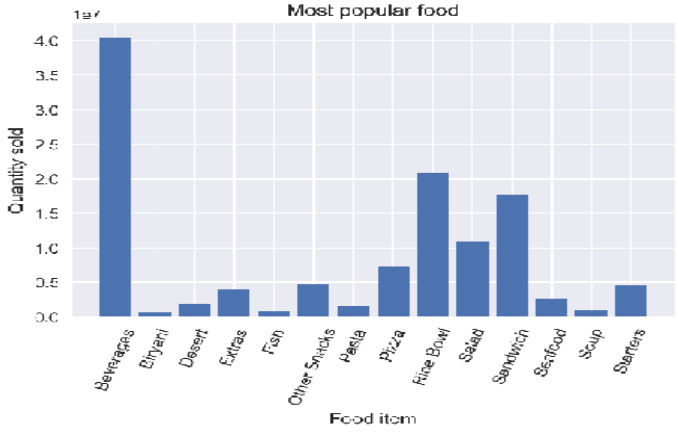
```
table = pd.pivot_table(data=df,index='category',values='num_orders',aggfunc=np.sum)
table
```

category	num_orders
Beverages	40480525
Biryani	631848
Desert	1940754
Extras	3984979
Fish	871959
Other Snacks	4766293
Pasta	1637744
Pizza	7383720
Rice Bowl	20874063
Salad	10944336
Sandwich	17636782
Seafood	2715714
Soup	1039646
Starters	4649122

Next, I will try to visualize this using a bar graph.

Bar graphs are best used when we need to compare the quantity of categorical values within the same category.

Bar graph is generated using ***plt.bar()*** in matplot lib:



\2. Pie Chart using matplotlib

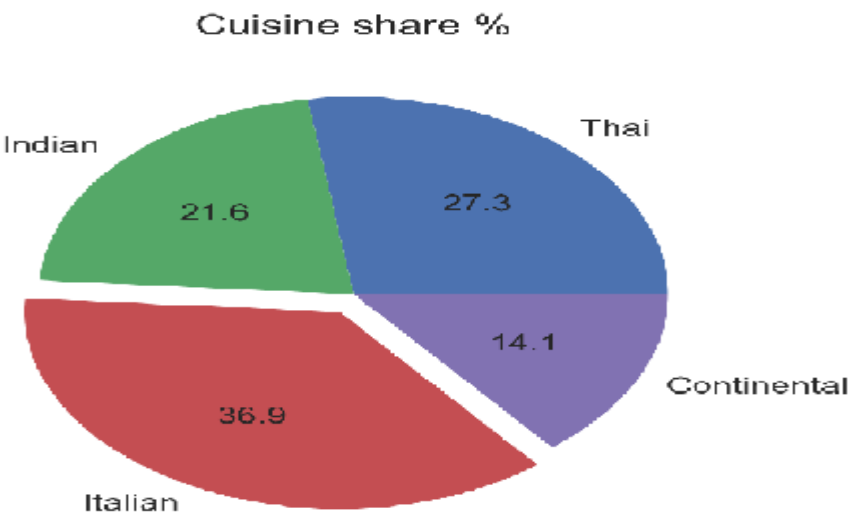
Let us now see the ratio of orders from each cuisine.

A pie chart is suitable to show the proportional distribution of items within the same category

```
#dictionary
for  cuisine
and        its
total orders
                d_cuisine = {}
                #total number of order
                total = df['num_orders'].sum()
                #find ratio of orders per cuisine
                for i in range(df['cuisine'].nunique()):
                #cuisine
                c = df['cuisine'].unique()[i]
                #num of orders for the cuisine
                c_order = df[df['cuisine']==c]['num_orders'].sum()
                d_cuisine[c] = c_order/total
```

Let’s plot the pie chart:

```
#pie plot
plt.pie([x*100       for       x       in       d_cuisine.values()],labels=[x       for       x       in
d_cuisine.keys()],autopct='%0.1f',explode=[0,0,0.1,0])
#label the plot
plt.title('Cuisine share %')
plt.savefig('C:\\Users\\Dell\\Desktop\\AV                                                    Plotting
images\\matplotlib_plotting_8.png',dpi=300,bbox_inches='tight')
plt.show();
```



- I used ***plt.pie()*** to draw the pie chart and adjust its parameters to make it more appealing
- The ***autopct*** parameter was used to print the values within the pie chart up to 1 decimal place
- The ***explode*** parameter was used to offset the Italian wedge to make it stand out from the rest. This makes it instantly clear to the viewer that people love Italian food!

A pie chart is rendered useless when there are a lot of items within a category. This will decrease the size of each slice and there will be no distinction between the items.

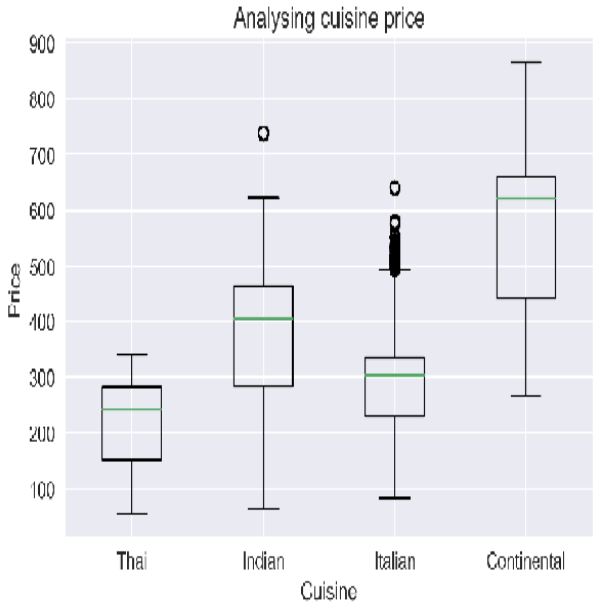
3. Box Plot using matplotlib

Since we are discussing cuisine, let's check out which one is the most expensive cuisine! For this, I will be using a ***Box Plot***.

Box plot gives statistical information about the distribution of numeric data divided into different groups. It is useful for detecting outliers within each group.

- The lower, middle and upper part of the box represents the **25th, 50th,** and **75th percentile** values respectively
- The top whisker represents **Q3+1.5*IQR**
- The bottom whisker represents **Q1-1.5*IQR**
- Outliers are shown as scatter points
- Shows skewness in the data

```
#dictionary for base
price per cuisine
                    c_price = {}
                    for i in df['cuisine'].unique():
                       c_price[i] = df[df['cuisine']==i].base_price
#plotting
boxplot
                plt.boxplot([x for x in c_price.values()],labels=[x for x in c_price.keys()])
                #x and y-axis labels
                plt.xlabel('Cuisine')
                plt.ylabel('Price')
                #plot title
                plt.title('Analysing cuisine price')
                #save and display
                plt.savefig('C:\\Users\\Dell\\Desktop\\AV                    Plotting
                images\\matplotlib_plotting_9.png',dpi=300,bbox_inches='tight')
                plt.show();
```



Continental cuisine was the most expensive cuisine served by the company! Even its median price is higher than the maximum price of all the cuisines.

Box plot does not show the distribution of data points within each group.

. Histogram using matplotlib

On the topic of prices, did we forget to inspect the base price and checkout price? Don't worry, we will do that using a histogram.

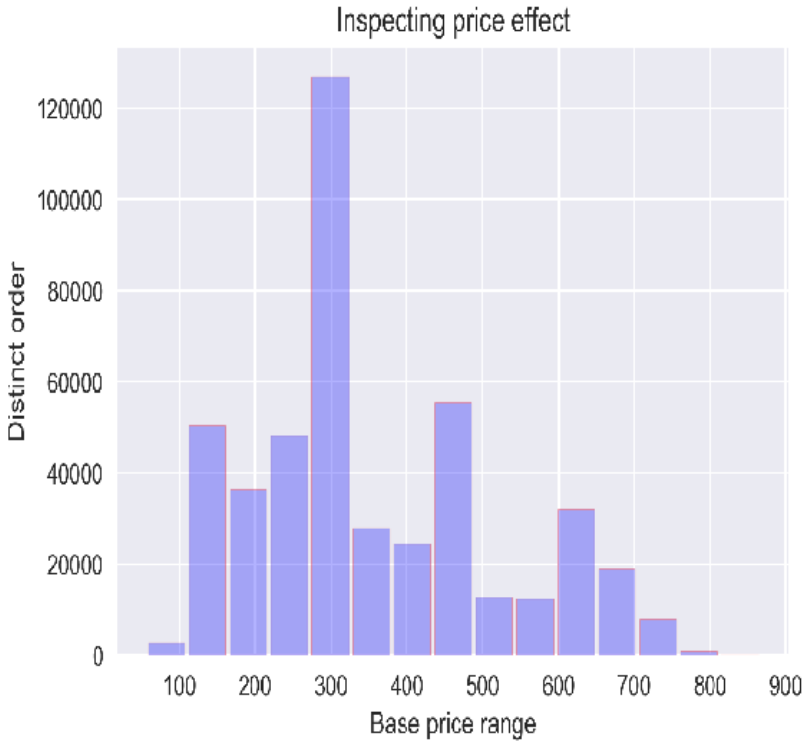
A histogram shows the distribution of numeric data through a continuous interval by segmenting data into different bins. Useful for inspecting skewness in the data.

Since ***base_price*** is a continuous variable, we will inspect its range in different distinct orders using a histogram. We can do this using ***plt.hist()***.

But the confusing part is what should be the number of bins? By default, it is 10. However, there is no correct answer and you can vary it according to your dataset to best visualize it.

```
#plotting histogram
plt.hist(df['base_price'],rwidth=0.9,alpha=0.3,color='blue',bins=15,edgecolor='red')
#x and y-axis labels
plt.xlabel('Base price range')
plt.ylabel('Distinct order')
```

```
#plot title
plt.title('Inspecting price effect')
#save and display the plot
plt.savefig('C:\\Users\\Dell\\Desktop\\AV                    Plotting
images\\matplotlib_plotting_10.png',dpi=300,bbox_inches='tight')
plt.show();
```



5. Line Plot and Subplots using matplotlib

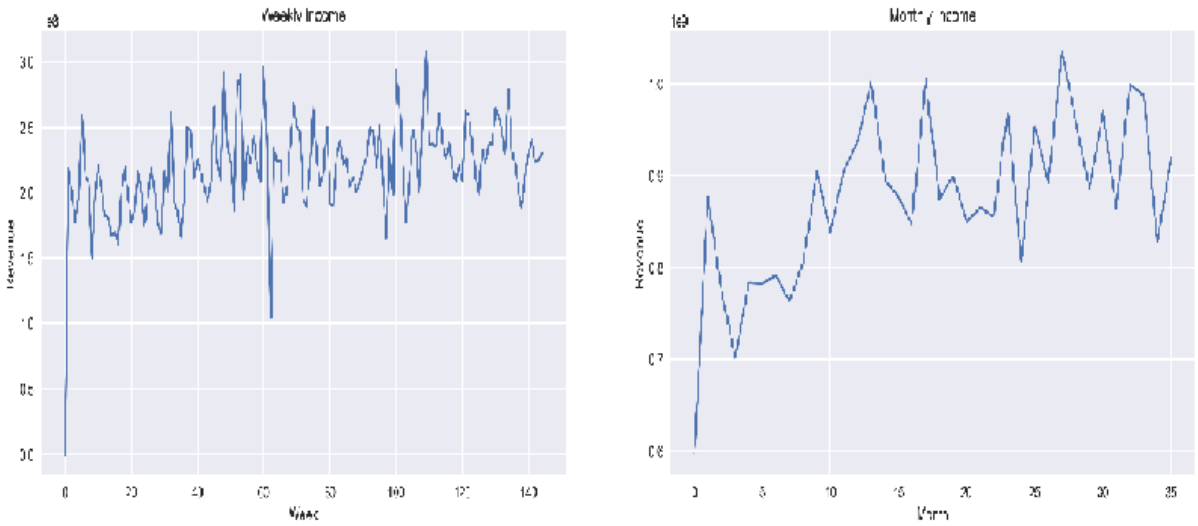
A line plot is useful for visualizing the trend in a numerical value over a continuous time interval.

How are the weekly and monthly sales of the company varying? This is a critical business question that makes or breaks the marketing strategy.

Before exploring that, I will create two lists for storing the week-wise and month-wise revenue of the company:

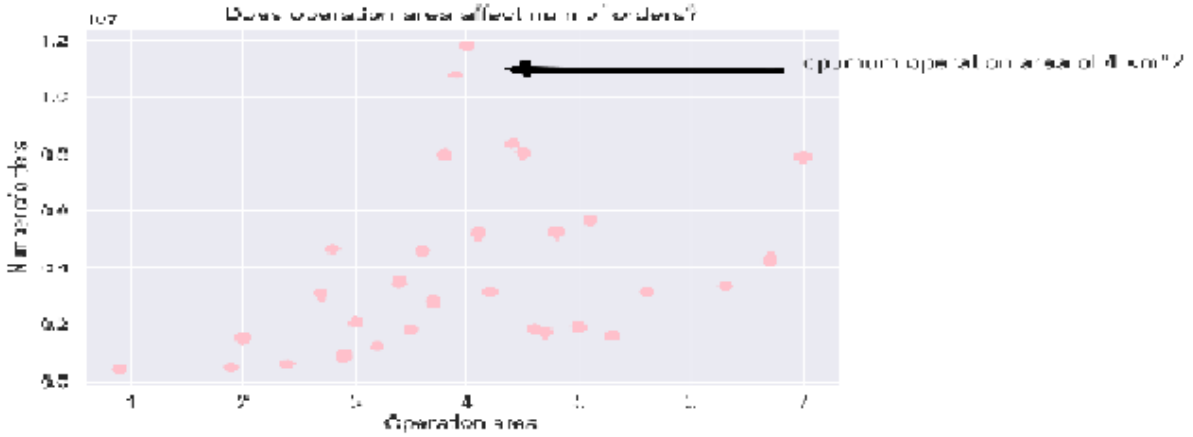
```
#new revenue column
df['revenue']          =          df.apply(lambda          x:
x.checkout_price*x.num_orders,axis=1)
#new month column
df['month'] = df['week'].apply(lambda x: x//4)
#list to store month-wise revenue
```

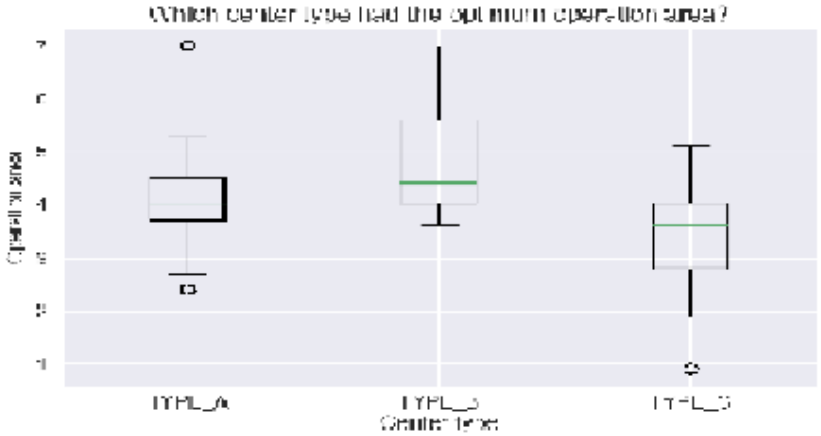
```
month=[]
month_order=[]
for i in range(max(df['month'])):
    month.append(i)
    month_order.append(df[df['month']==i].revenue.sum())

#list to store week-wise revenue
week=[]
week_order=[]
for i in range(max(df['week'])):
    week.append(i)
    week_order.append(df[df['week']==i].revenue.sum())
      fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))
      #manipulating the first Axes
      ax[0].plot(week,week_order)
      ax[0].set_xlabel('Week')
      ax[0].set_ylabel('Revenue')
      ax[0].set_title('Weekly income')
      #manipulating the second Axes
      ax[1].plot(month,month_order)
      ax[1].set_xlabel('Month')
      ax[1].set_ylabel('Revenue')
      ax[1].set_title('Monthly income')
      #save and display the plot
      plt.savefig('C:\\Users\\Dell\\Desktop\\AV                    Plotting
      images\\matplotlib_plotting_11.png',dpi=300,bbox_inches='tight')
      plt.show();
```

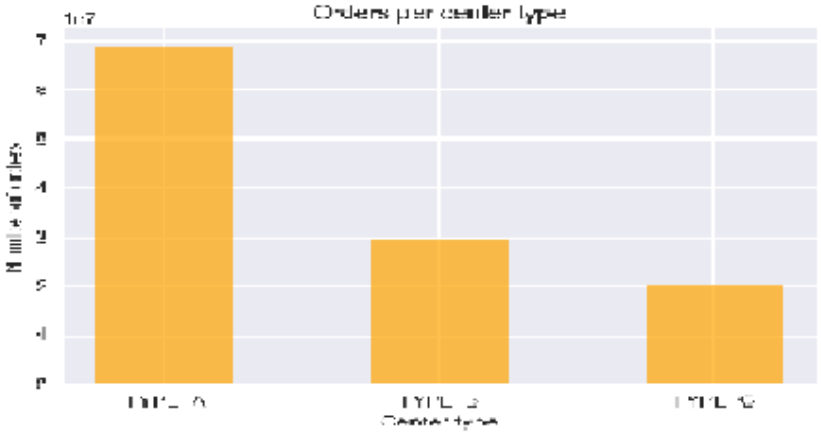


6. Scatter Plot using matplotlib

```
center_type_name =
['TYPE_A','TYPE_B','T
YPE_C']
                    #relation between op area and number of orders
                    op_table=pd.pivot_table(df,index='op_area',values='num_orders',agg
                    func=np.sum)
                    #relation between center type and op area
                    c_type = {}
                    for i in center_type_name:
                        c_type[i] = df[df['center_type']==i].op_area
                    #relation between center type and num of orders
                    center_table=pd.pivot_table(df,index='center_type',values='num_ord
                    ers',aggfunc=np.sum)
                    #subplots
                    fig,ax = plt.subplots(nrows=3,ncols=1,figsize=(8,12))
                    #scatter plots
                    ax[0].scatter(op_table.index,op_table['num_orders'],color='pink')
                    ax[0].set_xlabel('Operation area')
                    ax[0].set_ylabel('Number of orders')
                    ax[0].set_title('Does operation area affect num of orders?')
                    ax[0].annotate('optimum operation area of 4
                    km^2',xy=(4.2,1.1*10**7),xytext=(7,1.1*10**7),arrowprops=dict(fa
                    cecolor='black', shrink=0.05),fontsize=12)
                    #boxplot
                    ax[1].boxplot([x for x in c_type.values()], labels=[x for x in
                    c_type.keys()])
                    ax[1].set_xlabel('Center type')
                    ax[1].set_ylabel('Operation area')
                    ax[1].set_title('Which center type had the optimum operation area?')
                    #bar graph
                    ax[2].bar(center_table.index,center_table['num_orders'],alpha=0.7,c
                    olor='orange',width=0.5)
                    ax[2].set_xlabel('Center type')
                    ax[2].set_ylabel('Number of orders')
                    ax[2].set_title('Orders per center type')
                    #show figure
                    plt.tight_layout()
                    plt.savefig('C:\\Users\\Dell\\Desktop\\AV Plotting
                    images\\matplotlib_plotting_12.png',dpi=300,bbox_inches='tight')
                    plt.show();
```


Does operation area affect number of orders?
Optimum operation area of 4 km^2
Number of orders
Operation area


Which center type had the optimum operation area?
Operation area
TYPE_A
TYPE_B
TYPE_C
Center type


Orders per center type
Number of orders
TYPE_A
TYPE_B
TYPE_C
Center type