## INDEX

<table>
<thead>
<tr>
<th>S.NO</th>
<th>EXPERIMENT NAME</th>
<th>PAGE NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unit-I Build Data Warehouse and Explore WEKA</td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>Unit-II Perform data preprocessing tasks and Demonstrate performing association rule mining on data sets</td>
<td>53</td>
</tr>
<tr>
<td>3</td>
<td>Unit-III Demonstrate performing classification on data sets</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>Unit-IV Demonstrate performing clustering on data sets</td>
<td>88</td>
</tr>
<tr>
<td>5</td>
<td>Unit-V Demonstrate performing Regression on data sets</td>
<td>97</td>
</tr>
<tr>
<td>6</td>
<td>Task 1: Credit Risk Assessment. Sample Programs using German Credit Data</td>
<td>109</td>
</tr>
<tr>
<td>7</td>
<td>Task 2: Sample Programs using Hospital Management System</td>
<td>130</td>
</tr>
<tr>
<td>8</td>
<td>Beyond the Syllabus - Simple Project on Data Transformation</td>
<td>132</td>
</tr>
</tbody>
</table>
Unit-I Build Data Warehouse and Explore WEKA

A. 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.)

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

The data warehouse contains 4 tables:

1. Date dimension: contains every single date from 2006 to 2016.
2. Customer dimension: contains 100 customers. To be simple we’ll make it type 1 so we don’t create a new row for each change.
3. Van dimension: contains 20 vans. To be simple we’ll make it type 1 so we don’t create a new row for each change.
4. Hire fact table: contains 1000 hire transactions since 1\textsuperscript{st} Jan 2011. It is a daily snapshot fact table so that every day we insert 1000 rows into this fact table. So over time we can track the changes of total bill, van charges, satnav income, etc.

Create the source tables and populate them

create 3 tables in HireBase database: Customer, Van, and Hire and populate them.
Customer Dimension table:

<table>
<thead>
<tr>
<th>ID</th>
<th>CusName</th>
<th>Contact Name</th>
<th>PostalCode</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Francis</td>
<td>Flintoff</td>
<td>23452</td>
<td>INDIA</td>
</tr>
<tr>
<td>2</td>
<td>Michael</td>
<td>Flamingo</td>
<td>52132</td>
<td>INDIA</td>
</tr>
<tr>
<td>3</td>
<td>Nicole</td>
<td>Kidman</td>
<td>54121</td>
<td>INDIA</td>
</tr>
<tr>
<td>4</td>
<td>Dean</td>
<td>Konata</td>
<td>45454</td>
<td>INDIA</td>
</tr>
</tbody>
</table>

Van Dimension table:
And here is the script in mysql to create and populate them in pentaho:

**DATABASE CONNNECTION:**

1) Mysql Command prompt  
2) Enter password: root(****)  
3) Show databases;  

**Output:**

Database  
Information_schema

Abc  
**CREATE CUSTOMER TABLE:**

**STEP 1:**  
create database HireBase;  

**Output:**  
Query ok, 1 row affected

**STEP 2:**  
use HireBase;  

**output:**  
database changed

**STEP 3:**  
To check in which database we are in:  
Select database();  

**Output:**  
Database()  
Hirebase()

**STEP 4:**  
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));  

**Output:**  
Query ok, 0 rows affected

**STEP 5:**  
Insert into customer values (1,’CSE’,’14-jul-17’, 1234, 12,’lecturer’);  

**Output:**  
Query ok, 1 rows affected;
POPULATE CUSTOMER TABLE IN PENTAHO:

STEP 1:
- a) Goto Pentaho
- b) Data integration
- c) Spoon.bat (Double click on spoon.bat)
- d) pentaho opens
- e) There will be two options
  - i) View
  - ii) Design
- f) In view goto Transformation. In that click on Transformation 1 ➔ database connection
- g) In database connection ➔ goto MYSQL ➔ Settings

   - HostName: Local Host
   - Database name: Hirebase
   - Port number: 3306
   - Username: root
   - Password: root(****)

STEP 2:
1) In design ➔ Table input ➔ Dialogue box opens

   Stepname: Tableinput
   Connection name: Hirebase

2) Write the query in the Text space

   Select * from customer;

3) Click on preview button to populate the data

Similarly We need to create vantable, hiretable;

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

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

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)
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.

The tables which are created are displayed as in the fig:
<table>
<thead>
<tr>
<th>No.</th>
<th>Address</th>
<th>Category</th>
<th>Type</th>
<th>Grade</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123</td>
<td>123</td>
<td>123</td>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>2</td>
<td>234</td>
<td>234</td>
<td>234</td>
<td>234</td>
<td>234</td>
</tr>
<tr>
<td>3</td>
<td>345</td>
<td>345</td>
<td>345</td>
<td>345</td>
<td>345</td>
</tr>
<tr>
<td>4</td>
<td>456</td>
<td>456</td>
<td>456</td>
<td>456</td>
<td>456</td>
</tr>
<tr>
<td>5</td>
<td>567</td>
<td>567</td>
<td>567</td>
<td>567</td>
<td>567</td>
</tr>
</tbody>
</table>
Below is the script to create and populate those dim and fact tables:

```sql
-- 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
```
-- 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


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.
A 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:

```sql
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:

```sql
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),
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 (  
    SELECT par_id AS old_par_id,
            par_id AS party_id,
            CASE par_domain
                   WHEN 'P' THEN 'PE' /*Person*/
                   WHEN 'O' THEN 'BU' /*Business*/
               END AS party_type_code,
            par_first_name AS first_name,
            par_last_name AS surname,
            par_dob AS date_of_birth,
            par_business_name AS business_name,
    )
    ON (dest.party_id = old_par_id)
    WHEN NOT MATCHED
    THEN INSERT;
```
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 ,

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.
Here Slice is performed for the dimension "time" using the criterion time = "Q1".

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

Dice

Dice selects two or more dimensions from a given cube and provides a new sub-cube. Consider the following diagram that shows the dice operation.
The dice operation on the cube based on the following selection criteria involves three dimensions.

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

**Pivot**

The pivot operation is also known as rotation. It rotates the data axes in view in order to provide an alternative presentation of data. Consider the following diagram that shows the pivot operation.
A. (v). Explore visualization features of the tool for analysis like identifying trends etc.

Ans:

Visualization Features:

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

Access To Visualization From The Classifier, Cluster And Attribute Selection Panel Is Available From A Popup Menu. Click The Right Mouse Button Over An Entry In The Result List To Bring Up The Menu. You Will Be Presented With Options For Viewing Or
Saving The Text Output And --- Depending On The Scheme --- Further Options For Visualizing Errors, Clusters, Trees Etc.
To open Visualization screen, click ‘Visualize’ tab.

Select a square that corresponds to the attributes you would like to visualize. For example, let’s choose ‘outlook’ for X – axis and ‘play’ for Y – axis. Click anywhere inside the square that corresponds to ‘play on the left and ‘outlook’ at the to
Changing the View:

In the visualization window, beneath the X-axis selector there is a drop-down list, ‘Colour’, for choosing the color scheme. This allows you to choose the color of points based on the attribute selected. Below the plot area, there is a legend that describes what values the colors correspond to. In your example, red represents ‘no’, while blue represents ‘yes’. For better visibility you should change the color of label ‘yes’. Left-click on ‘yes’ in the ‘Class colour’ box and select lighter color from the color palette.

To the right of the plot area there are series of horizontal strips. Each strip represents an attribute, and the dots within it show the distribution values of the attribute. You can choose what axes are used in the main graph by clicking on these strips (left-click changes X-axis, right-click changes Y-axis).

The software sets X-axis to ‘Outlook’ attribute and Y-axis to ‘Play’. The instances are spread out in the plot area and concentration points are not visible. Keep sliding ‘Jitter’, a random displacement given to all points in the plot, to the right, until you can spot concentration points.

The results are shown below. But on this screen we changed ‘Colour’ to temperature. Besides ‘outlook’ and ‘play’, this allows you to see the ‘temperature’ corresponding to the ‘outlook’. It will affect your result because if you see ‘outlook = ‘sunny’ and ‘play’ = ‘no’ to explain the result, you need to see the ‘temperature’ – if it is too hot, you do not want to play. Change ‘Colour’ to ‘windy’, you can see that if it is windy, you do not want to play as well.

Selecting Instances

Sometimes it is helpful to select a subset of the data using visualization tool. A special case is the ‘UserClassifier’, which lets you to build your own classifier by interactively selecting instances. Below the Y-axis there is a drop-down list that allows you to choose a selection method. A group of points on the graph can be selected in four ways [2]
1. **Select Instance.** Click on an individual data point. It brings up a window listing attributes of the point. If more than one point will appear at the same location, more than one set of attributes will be shown.

2. **Rectangle.** You can create a rectangle by dragging it around the point.
3. **Polygon.** You can select several points by building a free-form polygon. Left-click on the graph to add vertices to the polygon and right-click to complete it.

4. **Polyline.** To distinguish the points on one side from the ones on another, you can build a polyline. Left-click on the graph to add vertices to the polyline and right-click to finish.
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 Luc
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.

![Weka GUI Chooser](image)

The buttons can be used to start the following applications:

- **Explorer** An environment for exploring data with WEKA (the rest of this Documentation deals with this application in more detail).
- **Experimenter** An environment for performing experiments and conducting statistical tests between learning schemes.
- **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.
- **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 interfac
1. Explorer

The Graphical user interface

1.1 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:

- Simple
- 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.
3. 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 are available in the KnowledgeFlow 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 incrementally requires a classifier that ca
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:

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

This results in the following output:

**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/Linux: /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
4.4,2.9,1.4,0.2,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 attribute 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 of that attribute and its 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 attribute's 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:

- numeric
- integer is treated as numeric
- real is treated as numeric
- <nominal-specification>
- string
- 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 this (”...” 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
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.
5. Select and Click on data option button.
Sample Weka Data Sets

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
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.
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.
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:

B. (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 filesystem.

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 information. 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., .* idselects 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 (see below). 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 clipboard 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.

قم بإتباع الخطوات التالية للعمل مع جدول البيانات:

1. افتح ملف WEKA.
2. افتح WEKA Explorer.
3. افتح panel Preprocessing.
4. افتح panel Preprocess.
5. افتح panel Preprocess.
6. افتح panel Preprocess.
7. افتح panel Preprocess.
8. افتح panel Preprocess.
9. افتح panel Preprocess.

**Dataset labor.arff**
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 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.
8. Click on Associate tab and Choose Aprior algorithm
9. Click on start button.

**Output**: Run information

```
== Run information ==
Scheme: weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation: weather.symbolic
Instances: 14
Attributes: outlook
temperature
humidity
windy
play
== Associator model (full training set) ==
Apriori
======
```

Minimum support: 0.15 (2 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17
Generated sets of large itemsets:

Size of set of large itemsets L(1): 12

Size of set of large itemsets L(2): 47

Size of set of large itemsets L(3): 39

Size of set of large itemsets L(4): 6

Best rules found:

1. outlook=overcast 4 ==> play=yes 4  conf:(1)
2. temperature=cool 4 ==> humidity=normal 4  conf:(1)
3. humidity=normal windy=FALSE 4 ==> play=yes 4  conf:(1)
4. outlook=sunny play=no 3 ==> humidity=high 3  conf:(1)
5. outlook=sunny humidity=high 3 ==> play=no 3  conf:(1)
6. outlook=rainy play=yes 3 ==> windy=FALSE 3  conf:(1)
7. outlook=rainy windy=FALSE 3 ==> play=yes 3  conf:(1)
8. temperature=cool play=yes 3 ==> humidity=normal 3  conf:(1)
9. outlook=sunny temperature=hot 2 ==> humidity=high 2  conf:(1)
10. temperature=hot play=no 2 ==> outlook=sunny 2  conf:(1)
**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 criteria support 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,**

  \[
  \text{support} = \frac{(X \cup Y).count}{n}
  \]

  \[
  \text{confidence} = \frac{(X \cup Y).count}{X . count}
  \]

  support = support($\{A U C\}$)

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

**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.
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.
Output: === Run information ===

Scheme: weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S 1.0 -c 1
Relation: weather.symbolic
Instances: 14
Attributes: 5
outlook
temperature
humidity
windy
play
=== Associator model (full training set) ===
Apriori
========
Minimum support: 0.15 (2 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

Size of set of large itemsets L(1): 12

Size of set of large itemsets L(2): 47
Size of set of large itemsets L(3): 39
Size of set of large itemsets L(4): 6

Best rules found:

1. outlook=overcast 4 ==> play=yes 4  conf:(1)
2. temperature=cool 4 ==> humidity=normal 4  conf:(1)
3. humidity=normal windy=FALSE 4 ==> play=yes 4  conf:(1)
4. outlook=sunny play=no 3 ==> humidity=high 3  conf:(1)
5. outlook=sunny humidity=high 3 ==> play=no 3  conf:(1)
6. outlook=rainy play=yes 3 ==> windy=FALSE 3  conf:(1)
7. outlook=rainy windy=FALSE 3 ==> play=yes 3  conf:(1)
Unit – III Demonstrate performing classification on data sets.

**Classification Tab**

**Selecting a Classifier**

At the top of the classify section is the Classifier box. This box has a text field that gives the name of the currently selected classifier, and its options. Clicking on the text box with the left mouse button brings up a GenericObjectEditor dialog box, just the same as for filters, that you can use to configure the options of the current classifier. With a right click (or Alt+Shift+left click) you can once again copy the setup string to the clipboard or display the properties in a GenericObjectEditor dialog box. The Choose button allows you to choose one of the classifiers that are available in WEKA.

**Test Options**

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

1. **Use training set.** The classifier is evaluated on how well it predicts the class of the instances it was trained on.

2. **Supplied test set.** The classifier is evaluated on how well it predicts the class of a set of instances loaded from a file. Clicking the Set... button brings up a dialog allowing you to choose the file to test on.

3. **Cross-validation.** The classifier is evaluated by cross-validation, using the number of folds that are entered in the Folds text field.

4. **Percentage split.** The classifier is evaluated on how well it predicts a certain percentage of the data which is held out for testing. The amount of data held out depends on the value entered in the % field.

**Classifier Evaluation Options:**

1. **Output model.** The classification model on the full training set is output so that it can be viewed, visualized, etc. This option is selected by default.
2. **Output per-class stats.** The precision/recall and true/false statistics for each class are output. This option is also selected by default.

3. **Output entropy evaluation measures.** Entropy evaluation measures are included in the output. This option is not selected by default.

4. **Output confusion matrix.** The confusion matrix of the classifier’s predictions is included in the output. This option is selected by default.

5. **Store predictions for visualization.** The classifier’s predictions are remembered so that they can be visualized. This option is selected by default.

6. **Output predictions.** The predictions on the evaluation data are output.

**Note** that in the case of a cross-validation the instance numbers do not correspond to the location in the data!

7. **Output additional attributes.** If additional attributes need to be output alongside the predictions, e.g., an ID attribute for tracking misclassifications, then the index of this attribute can be specified here. The usual **Weka ranges are supported, “first” and “last” are therefore valid indices as well (example: “first-3,6,8,12-last”).**

8. **Cost-sensitive evaluation.** The errors is evaluated with respect to a cost matrix. The Set... button allows you to specify the cost matrix used.

9. **Random seed for xval / % Split.** This specifies the random seed used when randomizing the data before it is divided up for evaluation purposes.

10. **Preserve order for % Split.** This suppresses the randomization of the data before splitting into train and test set.

11. **Output source code.** If the classifier can output the built model as Java source code, you can specify the class name here. The code will be printed in the “Classifier output” area.

**The Class Attribute**

The classifiers in **WEKA are designed to be trained to predict a single ‘class’**
attribute, which is the target for prediction. Some classifiers can only learn nominal classes; others can only learn numeric classes (regression problems) still others can learn both.

By default, the class is taken to be the last attribute in the data. If you want to train a classifier to predict a different attribute, click on the box below the Test options box to bring up a drop-down list of attributes to choose from.

Training a Classifier

Once the classifier, test options and class have all been set, the learning process is started by clicking on the Start button. While the classifier is busy being trained, the little bird moves around. You can stop the training process at any time by clicking on the Stop button. When training is complete, several things happen. The Classifier output area to the right of the display is filled with text describing the results of training and testing. A new entry appears in the Result list box. We look at the result list below; but first we investigate the text that has been output.

A. Load each dataset into Weka and run id3, j48 classification algorithm, study the classifier output. Compute entropy values, Kappa statistic.

Ans:

Steps for run ID3 and J48 Classification algorithms 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. Click on classify tab and Choose J48 algorithm and select use training set test option.
   9. Click on start button.
   10. Click on classify tab and Choose ID3 algorithm and select use training set test option.
   11. Click on start button.
Output:

=== Run information ===

Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2
Relation: iris
Instances: 150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth class

Test mode: evaluate on training data

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

J48 pruned tree

-------------

petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth > 0.6
| petalwidth <= 1.7
| | petallength <= 4.9: Iris-versicolor (48.0/1.0)
| | petallength > 4.9
| | | petalwidth <= 1.5: Iris-virginica (3.0)
| | | petalwidth > 1.5: Iris-versicolor (3.0/1.0)
| | petalwidth > 1.7: Iris-virginica (46.0/1.0)

Number of Leaves: 5

Size of the tree: 9

Time taken to build model: 0 seconds

=== Evaluation on training set ===

=== Summary ===
Correctly Classified Instances: 147 (98\%)  
Incorrectly Classified Instances: 3 (2\%)  

**Kappa statistic** 0.97  
K&B Relative Info Score: 14376.1925\%  
K&B Information Score: 227.8573 bits, 1.519 bits/instance  
Class complexity | order 0: 237.7444 bits, 1.585 bits/instance  
Class complexity | scheme: 16.7179 bits, 0.1115 bits/instance  
Complexity improvement (Sf): 221.0265 bits, 1.4735 bits/instance  
Mean absolute error: 0.0233  
Root mean squared error: 0.108  
Relative absolute error: 5.2482\%  
Root relative squared error: 22.9089\%  
Total Number of Instances: 150  

=== Detailed Accuracy By Class ===  

<table>
<thead>
<tr>
<th>Class</th>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris-setosa</td>
<td>0.98</td>
<td>0.02</td>
<td>0.961</td>
<td>0.98</td>
<td>0.97</td>
<td>0.99</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>Iris-versicolor</td>
<td>0.96</td>
<td>0.01</td>
<td>0.98</td>
<td>0.96</td>
<td>0.97</td>
<td>0.99</td>
<td>Iris-versicolor</td>
</tr>
<tr>
<td>Iris-virginica</td>
<td>0.98</td>
<td>0.01</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
<td>0.993</td>
</tr>
</tbody>
</table>

Weighted Avg. 0.98 0.01 0.98 0.98 0.98 Iris-virginica

=== Confusion Matrix ===

a b c <-- classified as  
50 0 0 | a = Iris-setosa  
0 49 1 | b = Iris-versicolor  
0 2 48 | c = Iris-virginica
The Classifier Output Text

The text in the Classifier output area has scroll bars allowing you to browse the results. Clicking with the left mouse button into the text area, while holding Alt and Shift, brings up a dialog that enables you to save the displayed output in a variety of formats (currently, BMP, EPS, JPEG and PNG). Of course, you can also resize the Explorer window to get a larger display area.

The output is

Split into several sections:

1. Run information. A list of information giving the learning scheme options, relation name, instances, attributes and test mode that were involved in the process.
2. Classifier model (full training set). A textual representation of the classification model that was produced on the full training data.

3. The results of the chosen test mode are broken down thus.

4. Summary. A list of statistics summarizing how accurately the classifier was able to predict the true class of the instances under the chosen test mode.

5. Detailed Accuracy By Class. A more detailed per-class break down **of the classifier's** prediction accuracy.

6. Confusion Matrix. Shows how many instances have been assigned to each class. Elements show the number of test examples whose actual class is the row and whose predicted class is the column.

7. Source code (optional). This section lists the Java source code if one **chose “Output source code” in the “More options” dialog.**

**B. Extract if-then rules from decision tree generated by classifier. Observe the 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.**

**Ans:**

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

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

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

**IF-THEN Rules:**

Rule-based classifier makes use of a set of IF-THEN rules for classification. We can express a rule in the following form –

**IF** condition **THEN** conclusion

Let us consider a rule R1,

| R1: IF age=youth AND student=yes | THEN buy_computer=yes |

**Points to remember** –

- The **IF** part of the rule is called **rule antecedent** or **precondition**.
- The **THEN** part of the rule is called **rule consequent**.
- The antecedent part the condition consist of one or more attribute tests and these tests are logically ANDed.
- The consequent part consists of class prediction.
Note – We can also write rule R1 as follows:

R1: (age = youth) ^ (student = yes))(buys computer = yes)

If the condition holds true for a given tuple, then the antecedent is satisfied.

Rule Extraction

Here we will learn how to build a rule-based classifier by extracting IF-THEN rules from a decision tree.

Points to remember –

- One rule is created for each path from the root to the leaf node.
- To form a rule antecedent, each splitting criterion is logically ANDed.
- The leaf node holds the class prediction, forming the rule consequent.

Rule Induction Using Sequential Covering Algorithm

Sequential Covering Algorithm can be used to extract IF-THEN rules form the training data. We do not require to generate a decision tree first. In this algorithm, each rule for a given class covers many of the tuples of that class.

Some of the sequential Covering Algorithms are AQ, CN2, and RIPPER. As per the general strategy the rules are learned one at a time. For each time rules are learned, a tuple covered by the rule is removed and the process continues for the rest of the tuples. This is because the path to each leaf in a decision tree corresponds to a rule.

Note – The Decision tree induction can be considered as learning a set of rules simultaneously.

The Following is the sequential learning Algorithm where rules are learned for one class at a time. When learning a rule from a class Ci, we want the rule to cover all the tuples from class C only and no tuple form any other class.

Algorithm: Sequential Covering

Input:
D, a data set class-labeled tuples,
Att_vals, the set of all attributes and their possible values.
Output: A Set of IF-THEN rules.
Method:
Rule_set={ }; // initial set of rules learned is empty

for each class c do

repeat
Rule = Learn_One_Rule(D, Att_valls, c); remove tuples covered by Rule form D; until termination condition;

Rule_set=Rule_set+Rule; // add a new rule to rule-set
end for
return Rule_Set;

Rule Pruning
The rule is pruned is due to the following reason –

- The Assessment of quality is made on the original set of training data. The rule may perform well on training data but less well on subsequent data. That's why the rule pruning is required.

- The rule is pruned by removing conjunct. The rule R is pruned, if pruned version of R has greater quality than what was assessed on an independent set of tuples.

FOIL is one of the simple and effective method for rule pruning. For a given rule R,

FOIL_Prune = pos - neg / pos + neg

where pos and neg is the number of positive tuples covered by R, respectively.

Note – This value will increase with the accuracy of R on the pruning set. Hence, if the FOIL_Prune value is higher for the pruned version of R, then we prune R.

_steps for run decision tree algorithms 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. Click on classify tab and Choose decision table algorithm and select cross-validation folds value-10 test option.
9. Click on start button.

**Output:**

```plaintext
=== Run information ===
Scheme:weka.classifiers.rules.DecisionTable -X 1 -S "weka.attributeSelection.BestFirst -D 1 -N 5"
Relation: iris
Instances: 150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth class

Test mode:10-fold cross-validation

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

Decision Table:

Number of training instances: 150
Number of Rules : 3
Non matches covered by Majority class.
Best first.
Start set: no attributes
Search direction: forward
Stale search after 5 node expansions
Total number of subsets evaluated: 12
Merit of best subset found: 96
Evaluation (for feature selection): CV (leave one out)
Feature set: 4,5
```
Time taken to build model: 0.02 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances 139 92.6667 %
Incorrectly Classified Instances 11 7.3333 %
Kappa statistic 0.89
Mean absolute error 0.092
Root mean squared error 0.2087
Relative absolute error 20.6978 %
Root relative squared error 44.2707 %
Total Number of Instances 150

=== Detailed Accuracy By Class ===

<table>
<thead>
<tr>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>0.88</td>
<td>0.05</td>
<td>0.898</td>
<td>0.88</td>
<td>0.889</td>
<td>0.946</td>
<td>Iris-versicolor</td>
</tr>
<tr>
<td>0.9</td>
<td>0.06</td>
<td>0.882</td>
<td>0.9</td>
<td>0.891</td>
<td>0.947</td>
<td>Iris-virginica</td>
</tr>
</tbody>
</table>

Weighted Avg. 0.927 0.037 0.927 0.927 0.927 0.964

=== Confusion Matrix ===

a  b  c  <-- classified as
50  0  0 | a = Iris-setosa
0  44  6 | b = Iris-versicolor
0  5  45 | c = Iris-virginica
C. Load each dataset into Weka and perform Naïve-bayes classification and k-Nearest Neighbor classification, Interpret the results obtained.

Ans:

9 Steps for run Naïve-bayes and k-nearest neighbor Classification algorithms 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. Click on classify tab and Choose Naïve-bayes algorithm and select use training set test option.
9. Click on start button.
10. Click on classify tab and Choose k-nearest neighbor and select use training set test option.
11. Click on start button.
Output: Naïve Bayes

=== Run information ===

Scheme: weka.classifiers.bayes.NaiveBayes
Relation: iris
Instances: 150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth class

Test mode: evaluate on training data

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

Naive Bayes Classifier

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Iris-setosa</th>
<th>Iris-versicolor</th>
<th>Iris-virginica</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(0.33)</td>
<td>(0.33)</td>
<td>(0.33)</td>
</tr>
</tbody>
</table>

sepal length

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>std. dev.</th>
<th>weight sum</th>
<th>precision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4.9913</td>
<td>0.355</td>
<td>50</td>
<td>0.1059</td>
</tr>
</tbody>
</table>

sepal width

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>std. dev.</th>
<th>weight sum</th>
<th>precision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.4015</td>
<td>0.3925</td>
<td>50</td>
<td>0.1091</td>
</tr>
</tbody>
</table>
mean 1.4694 4.2452 5.5516
std. dev. 0.1782 0.4712 0.5529
weight sum 50 50 50
precision 0.1405 0.1405 0.1405

petalwidth
mean 0.2743 1.3097 2.0343
std. dev. 0.1096 0.1915 0.2646
weight sum 50 50 50
precision 0.1143 0.1143 0.1143

Time taken to build model: 0 seconds

=== Evaluation on training set ====

=== Summary ===
Correctly Classified Instances 144 96 %
Incorrectly Classified Instances 6 4 %
Kappa statistic 0.94
Mean absolute error 0.0324
Root mean squared error 0.1495
Relative absolute error 7.2883 %
Root relative squared error 31.7089 %
Total Number of Instances 150

=== Detailed Accuracy By Class ===

<table>
<thead>
<tr>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1 Iris-setosa</td>
</tr>
<tr>
<td>0.96</td>
<td>0.04</td>
<td>0.923</td>
<td>0.96</td>
<td>0.941</td>
<td>0.993 Iris-versedcolor</td>
</tr>
<tr>
<td>0.02</td>
<td>0.958</td>
<td>0.92</td>
<td>0.92</td>
<td>0.939</td>
<td>0.993 Iris-virginica</td>
</tr>
<tr>
<td>Weighted Avg. 0.96</td>
<td>0.02</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
</tr>
</tbody>
</table>

=== Confusion Matrix ===
a b c <-- classified as
50 0 0 | a = Iris-setosa
0 48 2 | b = Iris-versicolor
0 4 46 | c = Iris-virginica.

Output: KNN (IBK)

=== Run information ===

Scheme:weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A weka.core.EuclideanDistance -R first-last"
Relation: iris
Instances: 150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth
class
Test mode: evaluate on training data

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

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

=== Evaluation on training set ===

=== Summary ===

Correctly Classified Instances 150 100 %
Incorrectly Classified Instances 0 0 %
Kappa statistic 1
Mean absolute error 0.0085
Root mean squared error 0.0091
Relative absolute error 1.9219 %
Root relative squared error 1.9335 %
Total Number of Instances 150

=== Detailed Accuracy By Class ===

<table>
<thead>
<tr>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Iris-versicolor</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Iris-virginica</td>
</tr>
</tbody>
</table>

Weighted Avg. 1 0 1 1 1 1

=== Confusion Matrix ===

a b c <-- classified as
50 0 0 | a = Iris-setosa
0 50 0 | b = Iris-versicolor
0 0 50 | c = Iris-virginica
D. Plot RoC Curves.

**Ans:** Steps for identify the plot RoC Curves.

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

Ans:

- Steps for run ID3 and J48 Classification algorithms 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. Click on classify tab and Choose J48 algorithm and select use training set test option.
9. Click on start button.
10. Click on classify tab and Choose ID3 algorithm and select use training set test option.
11. Click on start button.
12. Click on classify tab and Choose Naïve-bayes algorithm and select use training set test option.
13. Click on start button.
14. Click on classify tab and Choose k-nearest neighbor and select use training set test option.
15. Click on start button.

**J48:**

=== Run information ===

Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2
Relation: iris
Instances: 150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth
class

Test mode: evaluate on training data

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

J48 pruned tree

------------------
petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth > 0.6
  | petalwidth <= 1.7
  |   | petallength <= 4.9: Iris-versicolor (48.0/1.0)
  |   | petallength > 4.9
  |   | petalwidth <= 1.5: Iris-virginica (3.0)
  |   | petalwidth > 1.5: Iris-versicolor (3.0/1.0)
  | petalwidth > 1.7: Iris-virginica (46.0/1.0)

Number of Leaves: 5
Size of the tree : 9

Time taken to build model: 0 seconds

=== Evaluation on training set ===

Correctly Classified Instances 147 98 %
Incorrectly Classified Instances 3 2 %
Kappa statistic 0.97
Mean absolute error 0.0233
Root mean squared error 0.108
Relative absolute error 5.2482 %
Root relative squared error 22.9089 %
Total Number of Instances 150

=== Detailed Accuracy By Class ===

<table>
<thead>
<tr>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>0.98</td>
<td>0.02</td>
<td>0.961</td>
<td>0.98</td>
<td>0.97</td>
<td>0.99</td>
<td>Iris-versicolor</td>
</tr>
<tr>
<td>0.96</td>
<td>0.01</td>
<td>0.98</td>
<td>0.96</td>
<td>0.97</td>
<td>0.99</td>
<td>Iris-virginica</td>
</tr>
</tbody>
</table>
Weighted Avg. 0.98 0.01 0.98 0.98 0.98 0.993

=== Confusion Matrix ===

a b c <-- classified as
50 0 0  | a = Iris-setosa
0 49 1  | b = Iris-versicolor
0 2 48  | c = Iris-virginica

Naive-bayes:

=== Run information ===

Scheme: weka.classifiers.bayes.NaiveBayes
Relation: iris
Instances: 150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth
class
Test mode: evaluate on training data

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

Naive Bayes Classifier

Class Attribute Iris-setosa Iris-versicolor Iris-virginica
(0.33) (0.33) (0.33)

===
==== Summary ====
Correctly Classified Instances 144 96 %
Incorrectly Classified Instances 6 4 %
Kappa statistic 0.94
Mean absolute error 0.0324
Root mean squared error 0.1495
Relative absolute error 7.2883 %
Root relative squared error 31.7089 %
Total Number of Instances 150

==== Detailed Accuracy By Class====
<table>
<thead>
<tr>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris-setosa</td>
<td>0.96</td>
<td>0.04</td>
<td>0.923</td>
<td>0.96</td>
<td>0.941</td>
</tr>
<tr>
<td>Iris-versicolor</td>
<td>0.92</td>
<td>0.02</td>
<td>0.958</td>
<td>0.92</td>
<td>0.939</td>
</tr>
<tr>
<td>Iris-virginica</td>
<td>0.96</td>
<td>0.02</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
</tr>
</tbody>
</table>

=== Confusion Matrix ====
a b c <-- classified as
50 0 0 | a = Iris-setosa
0 48 2 | b = Iris-versicolor
0 4 46 | c = Iris-virginica

K-Nearest Neighbor (IBK):

=== Run information ====
Scheme:weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\"
Relation: iris
Instances: 150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth
class
Test mode: evaluate on training data

=== Classifier model (full training set) ====
IB1 instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

=== Evaluation on training set ===

=== Summary ===

Correctly Classified Instances 150 100 %
Incorrectly Classified Instances 0 0 %
Kappa statistic 1
Mean absolute error 0.0085
Root mean squared error 0.0091
Relative absolute error 1.9219 %
Root relative squared error 1.9335 %
Total Number of Instances 150

=== Detailed Accuracy By Class ===

<table>
<thead>
<tr>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Iris-versicolor</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Iris-virginica</td>
</tr>
</tbody>
</table>

Weighted Avg. 1 0 1 1 1 1

=== Confusion Matrix ===

a   b   c  <-- classified as
50   0   0 | a = Iris-setosa
0   50   0 | b = Iris-versicolor
0   0   50 | c = Iris-virginica
Unit – IV demonstrate performing clustering on data sets Clustering Tab

Selecting a Clusterer

By now you will be familiar with the process of selecting and configuring objects. Clicking on the clustering scheme listed in the Clusterer box at the top of the window brings up a GenericObjectEditor dialog with which to choose a new clustering scheme.

Cluster Modes

The Cluster mode box is used to choose what to cluster and how to evaluate the results. The first three options are the same as for classification: Use training set, Supplied test set and Percentage split (Section 5.3.1)—except that now the data is assigned to clusters instead of trying to predict a specific class. The fourth mode, Classes to clusters evaluation, compares how well the chosen clusters match up with a pre-assigned class in the data. The drop-down box below this option selects the class, just as in the Classify panel.

An additional option in the Cluster mode box, the Store clusters for visualization tick box, determines whether or not it will be possible to visualize the clusters once training is complete. When dealing with datasets that are so large that memory becomes a problem it may be helpful to disable this option.

Ignoring Attributes

Often, some attributes in the data should be ignored when clustering. The Ignore attributes button brings up a small window that allows you to select which attributes are ignored. Clicking on an attribute in the window highlights it, holding down the SHIFT key selects a range of consecutive attributes, and holding down CTRL toggles individual attributes on and off. To cancel the selection, back out with the Cancel button. To activate it, click the Select button. The next time clustering is invoked, the selected attributes are ignored.

Working with Filters
The Filtered Clusterer meta-clusterer offers the user the possibility to apply filters directly before the clusterer is learned. This approach eliminates the manual application of a filter in the Preprocess panel, since the data gets processed on the fly. Useful if one needs to try out different filter setups.

**Learning Clusters**

The Cluster section, like the Classify section, has Start/Stop buttons, a result text area and a result list. These all behave just like their classification counterparts. Right-clicking an entry in the result list brings up a similar menu, except that it shows only two visualization options: Visualize cluster assignments and Visualize tree. The latter is grayed out when it is not applicable.

**A. 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.**

**Ans:**

- Steps for run K-mean Clustering algorithms 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.
   - 8. Click on cluster tab and Choose k-mean and select use training set test option.
   - 9. Click on start button.

**Output:**

```plaintext
=== Run information ===

Scheme:weka.clusterers.SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10
Relation: iris
```
Instances: 150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth
class

Test mode: evaluate on training data

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

kMeans

======
Number of iterations: 7
Within cluster sum of squared errors: 62.1436882815797
Missing values globally replaced with mean/mode

Cluster centroids:
Cluster#
Attribute      Full Data  0    1
(150)         (100)    (50)

sepallength  5.8433  6.262  5.006
sepalwidth   3.054   2.872  3.418
petallength  3.7587  4.906  1.464
petalwidth   1.1987  1.676  0.244
class   Iris-setosa Iris-versicolor Iris-setosa

Time taken to build model (full training data): 0 seconds

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

Clustered Instances

0  100 ( 67%)
1  50  ( 33%)
B. Explore other clustering techniques available in Weka.

**Ans:** Clustering Algorithms And Techniques in WEKA, They are

![Clustering Methods Diagram]

- **Partitioning Methods**
  - k-means
  - k-medoids

- **Hierarchical Methods**
  - Bottom-up
  - Top-down

- **Density-based Methods**
  - DBSCAN
  - OPTICS
C. Explore visualization features of weka to visualize the clusters. Derive interesting insights and explain.

Ans: **Visualize Features**

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

Access To **Visualization** From The **Classifier, Cluster And Attribute Selection** Panel Is Available From A Popup Menu. Click The Right Mouse Button Over An Entry In The Result List To Bring Up The Menu. You Will Be Presented With Options For Viewing Or Saving The Text Output And --- Depending On The Scheme --- Further Options For Visualizing Errors, Clusters, Trees Etc.

To open Visualization screen, click ‘Visualize’ tab.
Select a square that corresponds to the attributes you would like to visualize. For example, let’s choose ‘outlook’ for X-axis and ‘play’ for Y-axis. Click anywhere inside the square that corresponds to ‘play o

**Changing the View:**

In the visualization window, beneath the X-axis selector there is a drop-down list, ‘Colour’, for choosing the color scheme. This allows you to choose the color of points based on the attribute selected. Below the plot area, there is a legend that describes what values the colors correspond to. In your example, red represents ‘no’, while blue represents ‘yes’. For better visibility you should change the color of label ‘yes’. Left-click on ‘yes’ in the ‘Class colour’ box and select lighter color from the color palette.

In the left and ‘outlook’ at the top.
Selecting Instances

Sometimes it is helpful to select a subset of the data using visualization tool. A special case is the ‘UserClassifier’, which lets you to build your own classifier by interactively selecting instances. Below the Y – axis there is a drop-down list that allows you to choose a selection method. A group of points on the graph can be selected in four ways [2]:

1. **Select Instance.** Click on an individual data point. It brings up a window listing attributes of the point. If more than one point will appear at the same location, more than one set of attributes will be shown.

![Instance Info Window]

2. **Rectangle.** You can create a rectangle by dragging it around the point.
3. **Polygon.** You can select several points by building a free-form polygon. Left-click on the graph to add vertices to the polygon and right-click to complete it.

4. **Polyline.** To distinguish the points on one side from the once on another, you can build a polyline. Left-click on the graph to add vertices to the polyline and right-click to finish.
Unit-V Demonstrate performing regression on data sets.

Regression:

Regression is a data mining function that predicts a number. Age, weight, distance, temperature, income, or sales could all be predicted using regression techniques. For example, a regression model could be used to predict children's height, given their age, weight, and other factors.

A regression task begins with a data set in which the target values are known. For example, a regression model that predicts children's height could be developed based on observed data for many children over a period of time. The data might track age, height, weight, developmental milestones, family history, and so on. Height would be the target, the other attributes would be the predictors, and the data for each child would constitute a case.

In the model build (training) process, a regression algorithm estimates the value of the target as a function of the predictors for each case in the build data. These relationships between predictors and target are summarized in a model, which can then be applied to a different data set in which the target values are unknown.

Regression models are tested by computing various statistics that measure the difference between the predicted values and the expected values. See "Testing a Regression Model".

Common Applications of Regression

Regression modeling has many applications in trend analysis, business planning, marketing, financial forecasting, time series prediction, biomedical and drug response modeling, and environmental modeling.

How Does Regression Work?

You do not need to understand the mathematics used in regression analysis to develop quality regression models for data mining. However, it is helpful to understand a few basic concepts.

The goal of regression analysis is to determine the values of parameters for a function that cause the function to best fit a set of data observations that you provide. The following equation expresses these relationships in symbols. It shows that regression is the process of estimating the
value of a continuous target \(y\) as a function \(F\) of one or more predictors \(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\), a set of parameters \(\theta_1, \theta_2, ..., \theta_h\), and a measure of error \(e\).
$y = F(x, \theta) + e$

The process of training a regression model involves finding the best parameter values for the function that minimize a measure of the error, for example, the sum of squared errors.

There are different families of regression functions and different ways of measuring the error.

**Linear Regression**

The simplest form of regression to visualize is linear regression with a single predictor. A linear regression technique can be used if the relationship between $x$ and $y$ can be approximated with a straight line, as shown in Figure 4-1.

**Figure 4-1 Linear Relationship Between x and y**

In a linear regression scenario with a single predictor ($y = \theta_2 x + \theta_1$), the regression parameters (also called coefficients) are:

- The **slope** of the line ($\theta_2$) — the angle between a data point and the regression line and
- The **y intercept** ($\theta_1$) — the point where $x$ crosses the $y$ axis ($x = 0$)
Nonlinear Regression

Often the relationship between x and y cannot be approximated with a straight line. In this case, a nonlinear regression technique may be used. Alternatively, the data could be preprocessed to make the relationship linear.

In Figure 4-2, x and y have a nonlinear relationship. Oracle Data Mining supports nonlinear regression via the gaussian kernel of SVM. (See "Kernel-Based Learning").

**Figure: Nonlinear Relationship Between x and y**

![Nonlinear Relationship Between x and y](image)

**Description of "Figure:Nonlinear Relationship Between x and y"**

Multivariate Regression

Multivariate regression refers to regression with multiple predictors (x₁, x₂, ..., xₙ). For purposes of illustration, Figure 4-1 and Figure 4-2 show regression with a single predictor. Multivariate regression is also referred to as multiple regression.

Regression Algorithms

Oracle Data Mining provides the following algorithms for regression:
Generalized Linear Models

Generalized Linear Models (GLM) is a popular statistical technique for linear modeling. Oracle Data Mining implements GLM for regression and classification. See Chapter 12, "Generalized Linear Models".

Support Vector Machines

Support Vector Machines (SVM) is a powerful, state-of-the-art algorithm for linear and nonlinear regression. Oracle Data Mining implements SVM for regression and other mining functions. See Chapter 18, "Support Vector Machines".

Note:
Both GLM and SVM, as implemented by Oracle Data Mining, are particularly suited for mining data that includes many predictors (wide data).

Testing a Regression Model

The Root Mean Squared Error and the Mean Absolute Error are statistics for evaluating the overall quality of a regression model. Different statistics may also be available depending on the regression methods used by the algorithm.

Root Mean Squared Error

The Root Mean Squared Error (RMSE) is the square root of the average squared distance of a data point from the fitted line. Figure 4-3 shows the formula for the RMSE.

Figure 4-3 Root Mean Squared Error

![RMSE formula](image)

Description of "Figure 4-3 Root Mean Squared Error"

This SQL expression calculates the RMSE.

\[
\text{SQRT}(\text{AVG}((\text{predicted\_value} - \text{actual\_value}) \times (\text{predicted\_value} - \text{actual\_value})))
\]
Mean Absolute Error

The Mean Absolute Error (MAE) is the average of the absolute value of the residuals. The MAE is very similar to the RMSE but is less sensitive to large errors. Figure 4-4 shows the formula for the MAE.

Figure: Mean Absolute Error

\[
MAE = \frac{1}{n} \sum_{j=1}^{n} |y_j - \hat{y}_j|
\]

A. Load each dataset into Weka and build Linear Regression model. Study the cluster formed. Use training set option. Interpret the regression model and derive patterns and conclusions from the regression results.

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 labor data set and open file.
8. Click on Classify tab and Click the Choose button then expand the functions branch.
9. Select the LinearRegression leaf ans select use training set test option.
10. Click on start button.

Output:

=== Run information ===

Scheme: weka.classifiers.functions.LinearRegression -S 0 -R 1.0E-8
Relation: labor-neg-data
Instances: 57
Attributes: 17
duration
wage-increase-first-year
wage-increase-second-year
wage-increase-third-year
cost-of-living-adjustment
working-hours
pension standby-pay
shift-differential
education-allowance
statutory-holidays
vacation
longterm-disability-assistance
contribution-to-dental-plan
bereavement-assistance
contribution-to-health-plan
class
Test mode: 10-fold cross-validation
=== Classifier model (full training set) ===
Linear Regression Model
duration =
0.4689 * cost-of-living-adjustment=tc.tcf +
0.6523 * pension=none.empl_contr +
1.0321 * bereavement-assistance=yes +
0.3904 * contribution-to-health-plan=full +
0.2765

Time taken to build model: 0 seconds

=== Cross-validation ===

=== Summary ===

Correlation coefficient  0.1967
Mean absolute error       0.6499
Root mean squared error   0.777
Relative absolute error   111.6598 %
Root relative squared error 108.8152 %
Total Number of Instances  56
Ignored Class Unknown Instances  1
B. Use options cross-validation and percentage split and repeat running the Linear Regression Model. Observe the results and derive meaningful results.

**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 labor data set and open file.
8. Click on Classify tab and Click the Choose button then expand the functions branch.
9. Select the LinearRegression leaf and select test options cross-validation.
10. Click on start button.
11. Select the LinearRegression leaf and select test options percentage split.
12. Click on start button.

**Output: cross-validation**

```plaintext
=== Run information ===
Scheme: weka.classifiers.functions.LinearRegression -S 0 -R 1.0E-8
Relation: labor-neg-data
Instances: 57
Attributes: 17
duration
wage-increase-first-year
wage-increase-second-year
wage-increase-third-year
cost-of-living-adjustment
working-hours
pension standby-pay
shift-differential
education-allowance
statutory-holidays
```
vacation longterm-disability-
assistance contribution-to-
dental-plan bereavement-
assistance contribution-to-
health-plan class

Test mode: 10-fold cross-validation

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

Linear Regression Model

duration =

0.4689 * cost-of-living-adjustment=tc.tcf +
0.6523 * pension=none,empl_contri +
1.0321 * bereavement-assistance=yes +
0.3904 * contribution-to-health-plan=full
+ 0.2765

Time taken to build model: 0.02 seconds

=== Cross-validation ===

=== Summary ===

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation coefficient</td>
<td>0.1967</td>
</tr>
<tr>
<td>Mean absolute error</td>
<td>0.6499</td>
</tr>
<tr>
<td>Root mean squared error</td>
<td>0.777</td>
</tr>
<tr>
<td>Relative absolute error</td>
<td>111.6598 %</td>
</tr>
<tr>
<td>Root relative squared error</td>
<td>108.8152 %</td>
</tr>
<tr>
<td>Total Number of Instances</td>
<td>56</td>
</tr>
<tr>
<td>Ignored Class Unknown Instances</td>
<td>1</td>
</tr>
</tbody>
</table>
Output: percentage split

=== Run information ===

Scheme:  weka.classifiers.functions.LinearRegression -S 0 -R 1.0E-8
Relation:  labor-neg-data
Instances:  57
Attributes:  17
duration wage-increase-
first-year wage-increase-
second-year wage-
increase-third-year cost-of-
living-adjustment
working-hours
pension standby-pay
shift-differential
education-
allowance statutory-
holidays vacation
longterm-disability-assistance
contribution-to-dental-plan
bereavement-assistance
contribution-to-health-plan
class
Test mode: split 66.0% train, remainder test

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

Linear Regression Model
duration =
0.4689 * cost-of-living-adjustment=tc.tcf +
0.6523 * pension=none,empl_contr +
1.0321 * bereavement-assistance=yes +
0.3904 * contribution-to-health-plan=full
+ 0.2765
Time taken to build model: 0.02 seconds

=== Evaluation on test split ===

=== Summary ===

Correlation coefficient 0.243
Mean absolute error 0.783
Root mean squared error 0.9496
Relative absolute error 106.8823 %
Root relative squared error 114.13 %
Total Number of Instances 19
C. Explore Simple linear regression techniques that only looks at one variable.

**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 labor data set and open file.
8. Click on Classify tab and Click the Choose button then expand the functions branch.
9. Select the Simple Linear Regression leaf and select test options cross-validation.
10. Click on start button.
6. Sample Programs using German Credit Data.

Task 1: Credit Risk Assessment

Description: The business of banks is making loans. Assessing the credit worthiness of an applicant is of crucial importance. You have to develop a system to help a loan officer decide **whether the credit of a customer is good. Or bad. A bank’s business rules regarding loans must** consider two opposing factors. On the one hand, a bank wants to make as many loans as possible.

Interest on these loans is the bank’s profit source. On the other hand, a bank can not afford to make too many bad loans. Too many bad loans could lead to the collapse of the bank. The bank’s loan policy must involved a compromise. Not too strict and not too lenient.

To do the assignment, you first and foremost need some knowledge about the world of credit. You can acquire such knowledge in a number of ways.

1. Knowledge engineering: Find a loan officer who is willing to talk. Interview her and try to represent her knowledge in a number of ways.

2. Books: Find some training manuals for loan officers or perhaps a suitable textbook on finance. Translate this knowledge from text from to production rule form.

3. Common sense: Imagine yourself as a loan officer and make up reasonable rules which can be used to judge the credit worthiness of a loan applicant.

4. Case histories: Find records of actual cases where competent loan officers correctly judged when and not to. Approve a loan application.

The German Credit Data

Actual historical credit data is not always easy to come by because of confidentiality rules. Here is one such data set. Consisting of 1000 actual cases collected in Germany.

In spite of the fact that the data is German, you should probably make use of it for this assignment (Unless you really can consult a real loan officer!)
There are 20 attributes used in judging a loan applicant (i.e., 7 Numerical attributes and 13 Categoric or Nominal attributes). The goal is to classify the applicant into one of two categories: Good or Bad.

The total number of attributes present in German credit data are:

1. Checking_Status
2. Duration
3. Credit_history
4. Purpose
5. Credit_amount
6. Savings_status
7. Employment
8. Installment_Commitment
9. Personal_status
10. Other_parties
11. Residence_since
12. Property_Magnitude
13. Age
14. Other_payment_plans
15. Housing
16. Existing_credits
17. Job
18. Num_dependents
19. Own_telephone
20. Foreign_worker
21. Class
Tasks (Turn in your answers to the following tasks)

1. List all the categorical (or nominal) attributes and the real valued attributes separately.

Ans) Steps for identifying categorical attributes

1. Double click on credit-g.arff file.
2. Select all categorical attributes.
3. Click on invert.
4. Then we get all real valued attributes selected
5. Click on remove
6. Click on visualize all.

Steps for identifying real valued attributes

1. Double click on credit-g.arff file.
2. Select all real valued attributes.
3. Click on invert.
4. Then we get all categorical attributes selected
5. Click on remove
6. Click on visualize all.

The following are the Categorical (or Nominal) attributes)

1. Checking_Status
2. Credit_history
3. Purpose
4. Savings_status
5. Employment
6. Personal_status
7. Other_parties
8. Property_Magnitude
9. Other_payment_plans
10. Housing
11. Job
12. Own_telephone
13. Foreign_worker

The following are the Numerical attributes:

1. Duration
2. Credit_amount
3. Installment_Commitment
4. Residence_since
5. Age
6. Existing_credits
7. Num_dependents

2. What attributes do you think might be crucial in making the credit assessment? Come up with some simple rules in plain English using your selected attributes.

Ans) The following are the attributes may be crucial in making the credit assessment.
1. Credit_amount
2. Age
3. Job
4. Savings_status
5. Existing_credits
6. Installment_commitment
7. Property_magnitude

3. One type of model that you can create is a Decision tree. Train a Decision tree using the complete data set as the training data. Report the model obtained after training.

Ans) Steps to model decision tree.
1. Double click on credit-g.arff file.
2. Consider all the 21 attributes for making decision tree.
3. Click on classify tab.
4. Click on choose button.
5. Expand tree folder and select J48
6. Click on use training set in test options.
7. Click on start button.
8. Right click on result list and choose the visualize tree to get decision tree.
We created a decision tree by using J48 Technique for the complete dataset as the training data.

The following model obtained after training.

**Output:**

```plaintext
=== Run information ===

Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2
Relation: german_credit
Instances: 1000
Attributes: 21

Checking_status duration credit_history purpose credit_amount savings_status employment installment_commitment personal_status other_parties residence_since property_magnitude age other_payment_plans housing existing_credits job num_dependents own_telephone foreign_worker class

Test mode: evaluate on training data

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

J48 pruned tree

-------------

Number of Leaves : 103

Size of the tree : 140

Time taken to build model: 0.08 seconds

=== Evaluation on training set ===

=== Summary ===

Correctly Classified Instances 855 85.5 %
Incorrectly Classified Instances 145 14.5 %
```
Kappa statistic 0.6251
Mean absolute error 0.2312
Root mean squared error 0.34
Relative absolute error 55.0377 %
Root relative squared error 74.2015 %
Coverage of cases (0.95 level) 100 %
Mean rel. region size (0.95 level) 93.3 %
Total Number of Instances 1000

=== Detailed Accuracy By Class ===

<table>
<thead>
<tr>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.956</td>
<td>0.38</td>
<td>0.854</td>
<td>0.956</td>
<td>0.902</td>
<td>0.857</td>
</tr>
<tr>
<td>0.62</td>
<td>0.044</td>
<td>0.857</td>
<td>0.62</td>
<td>0.72</td>
<td>0.857</td>
</tr>
</tbody>
</table>

Weighted Avg 0.855

=== Confusion Matrix ===

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>&lt;classified as</th>
</tr>
</thead>
<tbody>
<tr>
<td>669</td>
<td>a</td>
<td>good</td>
</tr>
<tr>
<td>114</td>
<td>186</td>
<td>b = bad</td>
</tr>
</tbody>
</table>

4. Suppose you use your above model trained on the complete dataset, and classify credit good/bad for each of the examples in the dataset. What % of examples can you classify correctly? (This is also called testing on the training set) why do you think can not get 100% training accuracy?

Ans) Steps followed are:

1. Double click on credit-g.arff file.
2. Click on classify tab.
3. Click on choose button.
4. Expand tree folder and select J48
5. Click on use training set in test options.
6. Click on start button.
7. On right side we find confusion matrix
8. Note the correctly classified instances.
Output:
If we used our above model trained on the complete dataset and classified credit as good/bad for each of the examples in that dataset. We can not get 100% training accuracy only 85.5% of examples, we can classify correctly.

5. Is testing on the training set as you did above a good idea? Why or why not?
Ans) It is not a good idea by using 100% training data set.

6. One approach for solving the problem encountered in the previous question is using cross-validation? Describe what is cross validation briefly. Train a decision tree again using cross validation and report your results. Does accuracy increase/decrease? Why?

Ans) steps followed are:
1. Double click on credit-g.arff file.
2. Click on classify tab.
3. Click on choose button.
4. Expand tree folder and select J48
5. Click on cross validations in test options.
6. Select folds as 10
7. Click on start
8. Change the folds to 5
9. Again click on start
10. Change the folds with 2
11. Click on start.
12. Right click on blue bar under result list and go to visualize tree

Output:
Cross-Validation Definition: The classifier is evaluated by cross validation using the number of folds that are entered in the folds text field.
In Classify Tab, Select cross-validation option and folds size is 2 then Press Start Button, next time change as folds size is 5 then press start, and next time change as folds size is 10 then press start.

i) Fold Size-10
Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances    705       70.5 %
Incorrectly Classified Instances  295       29.5 %
Kappa statistic                  0.2467
Mean absolute error              0.3467
Root mean squared error          0.4796
Relative absolute error          82.5233 %
Root relative squared error      104.6565 %
Coverage of cases (0.95 level)   92.8 %
Mean rel. region size (0.95 level) 91.7 %
Total Number of Instances       1000

=== Detailed Accuracy By Class ===

<table>
<thead>
<tr>
<th>Class</th>
<th>TP Rate</th>
<th>FP Rate Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>good</td>
<td>0.84</td>
<td>0.61</td>
<td>0.763</td>
<td>0.84</td>
<td>0.799</td>
</tr>
<tr>
<td>bad</td>
<td>0.39</td>
<td>0.16</td>
<td>0.511</td>
<td>0.39</td>
<td>0.442</td>
</tr>
<tr>
<td>Weighted Avg.</td>
<td>0.705</td>
<td>0.475</td>
<td>0.687</td>
<td>0.705</td>
<td>0.692</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>0.639</th>
<th>good</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.639</td>
<td>bad</td>
<td></td>
</tr>
</tbody>
</table>

--- Confusion Matrix ---

a b <-- classified as
588 112 | a = good
183 117 | b = bad

ii) Fold Size-5
Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances      733      73.3 %
Incorrectly Classified Instances    267      26.7 %
Kappa statistic                     0.3264
Mean absolute error                 0.3293
Root mean squared error             0.4579
Relative absolute error             78.3705 %
Root relative squared error         99.914 %
Coverage of cases (0.95 level)      94.7 %
Mean rel. region size (0.95 level)  93 %
Total Number of Instances           1000
=== Detailed Accuracy By Class ===

<table>
<thead>
<tr>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.851</td>
<td>0.543</td>
<td>0.785</td>
<td>0.851</td>
<td>0.817</td>
<td>0.685</td>
<td>good</td>
</tr>
<tr>
<td>0.457</td>
<td>0.149</td>
<td>0.568</td>
<td>0.457</td>
<td>0.506</td>
<td>0.685</td>
<td>bad</td>
</tr>
</tbody>
</table>

Weighted Avg. 0.733 0.425 0.72 0.733 0.724 0.685

=== Confusion Matrix ===

a b <-- classified as
596 104 | a = good
163 137 | b = bad

iii) Fold Size-2
Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances  721 72.1%
Incorrectly Classified Instances 279 27.9%
Kappa statistic 0.2443
Mean absolute error 0.3407
Root mean squared error 0.4669
Relative absolute error 81.0491%
Root relative squared error 101.8806%
Coverage of cases (0.95 level) 92.8%
Mean rel. region size (0.95 level) 91.3%
Total Number of Instances 1000

=== Detailed Accuracy By Class ===

<table>
<thead>
<tr>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.891</td>
<td>0.677</td>
<td>0.755</td>
<td>0.891</td>
<td>0.817</td>
<td>0.662</td>
<td>good</td>
</tr>
<tr>
<td>0.323</td>
<td>0.109</td>
<td>0.561</td>
<td>0.323</td>
<td>0.41</td>
<td>0.662</td>
<td>bad</td>
</tr>
</tbody>
</table>

Weighted Avg. 0.721 0.506 0.696 0.721 0.695 0.662

=== Confusion Matrix ===

a b <-- classified as
624  76 | a = good
203  97 | b = bad

**Note:** With this observation, we have seen accuracy is increased when we have folds size is 5 and accuracy is decreased when we have 10 folds.

7. Check to see if the data shows a bias against “foreign workers” or “personal-status”.

One way to do this is to remove these attributes from the data set and see if the decision tree created in those cases is significantly different from the full dataset case which you have already done. Did removing these attributes have any significantly effect? Discuss.

Ans) steps followed are:
1. Double click on credit-g.arff file.
2. Click on classify tab.
3. Click on choose button.
4. Expand tree folder and select J48
5. Click on cross validations in test options.
6. Select folds as 10
7. Click on start
8. Click on visualization
9. Now click on preprocessor tab
10. Select 9th and 20th attribute
11. Click on remove button
12. Goto classify tab
13. Choose J48 tree
14. Select cross validation with 10 folds
15. Click on start button
16. Right click on blue bar under the result list and go to visualize tree.

**Output:**

We use the **Preprocess Tab in Weka GUI Explorer to remove an attribute “Foreign-workers” & “Personal_status” one by one.** In Classify Tab, Select Use Training set option then

Press Start Button, If these attributes removed from the dataset, we can see change in the accuracy compare to full data set when we removed.
i) If Foreign_worker is removed

Evaluation on training set ===

=== Summary ===

Correctly Classified Instances 859 85.9 %
Incorrectly Classified Instances 141 14.1 %
Kappa statistic 0.6377
Mean absolute error 0.2233
Root mean squared error 0.3341
Relative absolute error 53.1347 %
Root relative squared error 72.9074 %
Coverage of cases (0.95 level) 100 %
Mean rel. region size (0.95 level) 91.9 %
Total Number of Instances 1000

=== Detailed Accuracy By Class ===

<table>
<thead>
<tr>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.954</td>
<td>0.363</td>
<td>0.86</td>
<td>0.954</td>
<td>0.905</td>
</tr>
<tr>
<td>0.637</td>
<td>0.046</td>
<td>0.857</td>
<td>0.637</td>
<td>0.73</td>
</tr>
</tbody>
</table>

Weighted Avg 0.859 0.268 0.857 0.859 0.852

0.867 good
0.867 bad

=== Confusion Matrix ===

a b <-- classified as
668 32 | a = good
109 191 | b = bad

i) If Personal_status is removed

Evaluation on training set ===

=== Summary ===

Correctly Classified Instances 866 86.6 %
Incorrectly Classified Instances 134 13.4 %
Kappa statistic 0.6582
Mean absolute error 0.2162
Root mean squared error 0.3288
Relative absolute error 51.4483 %
Root relative squared error 71.7411 %
Coverage of cases (0.95 level) 100 %
Mean rel. region size (0.95 level) 91.7 %
Total Number of Instances 1000

=== Detailed Accuracy By Class ===

<table>
<thead>
<tr>
<th>Class</th>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>good</td>
<td>0.954</td>
<td>0.34</td>
<td>0.868</td>
<td>0.954</td>
<td>0.909</td>
<td>0.866</td>
</tr>
<tr>
<td>bad</td>
<td>0.66</td>
<td>0.046</td>
<td>0.861</td>
<td>0.66</td>
<td>0.747</td>
<td>0.868</td>
</tr>
</tbody>
</table>

Weighted Avg. 0.866

=== Confusion Matrix ===

a b <-- classified as
668 32 | a = good 102
198 | b = bad

Note: With this observation we have seen, when “Foreign_worker “attribute is removed from the Dataset, the accuracy is decreased. So this attribute is important for classification.

8. Another question might be, do you really need to input so many attributes to get good results? May be only a few would do. For example, you could try just having attributes 2,3,5,7,10,17 and 21. Try out some combinations.(You had removed two attributes in problem 7. Remember to reload the arff data file to get all the attributes initially before you start selecting the ones you want.)

Ans) steps followed are:
1. Double click on credit-g.arff file.
2. Select 2,3,5,7,10,17,21 and tick the check boxes.
3. Click on invert
4. Click on remove
5. Click on classify tab
6. Choose trace and then algorithm as J48
7. Select cross validation folds as 2
8. Click on start.
OUTPUT:
We use the **Preprocess Tab** in Weka GUI Explorer to remove 2\textsuperscript{nd} attribute (Duration). In Classify Tab, Select Use Training set option then Press Start Button, If these attributes removed from the dataset, we can see change in the accuracy compare to full data set when we removed.

```
=== Evaluation on training set ===
=== Summary ===

| Correctly Classified Instances | 841 | 84.1 | % |
| Incorrectly Classified Instances | 159 | 15.9 | % |
```

Remember to reload the previous removed attribute, press Undo option in Preprocess tab. We use the **Preprocess Tab** in Weka GUI Explorer to remove 3\textsuperscript{rd} attribute (Credit_history). In Classify Tab, Select Use Training set option then Press Start Button, If these attributes removed from the dataset, we can see change in the accuracy compare to full data set when we removed.

```
=== Evaluation on training set ===
=== Summary ===

| Correctly Classified Instances | 839 | 83.9 | % |
| Incorrectly Classified Instances | 161 | 16.1 | % |
```

Remember to reload the previous removed attribute, press Undo option in Preprocess tab. We use the **Preprocess Tab** in Weka GUI Explorer to remove 5\textsuperscript{th} attribute (Credit_amount). In Classify Tab, Select Use Training set option then Press Start Button, If these attributes removed from the dataset, we can see change in the accuracy compare to full data set when we removed.

```
=== Evaluation on training set ===
=== Summary ===

| Correctly Classified Instances | 839 | 83.9 | % |
| Incorrectly Classified Instances | 161 | 16.1 | % |
```
== Evaluation on training set ==
== Summary ==

Correctly Classified Instances  864  86.4  
Incorrectly Classified Instances 136  13.6  

= Confusion Matrix ==

a b  <-- classified as
675  25 | a = good
111  189 | b = bad

Remember to reload the previous removed attribute, press Undo option in Preprocess tab. We use the Preprocess Tab in Weka GUI Explorer to remove 7th attribute (Employment). In Classify Tab, Select Use Training set option then Press Start Button, If these attributes removed from the dataset, we can see change in the accuracy compare to full data set when we removed.

== Evaluation on training set ==
== Summary ==

Correctly Classified Instances  858  85.8  
Incorrectly Classified Instances 142  14.2  

== Confusion Matrix ==

a b  <-- classified as
670  30 | a = good
112  188 | b = bad

Remember to reload the previous removed attribute, press Undo option in Preprocess tab. We use the Preprocess Tab in Weka GUI Explorer to remove 10th attribute (Other_parties). In Classify Tab, Select Use Training set option then Press Start Button, If these attributes removed from the dataset, we can see change in the accuracy compare to full data set when we removed.

Time taken to build model: 0.05 seconds

== Evaluation on training set ==
== Summary ==

Correctly Classified Instances  845  84.5  

129  COMPUTER SCIENCE & ENGINEERING
**Confusion Matrix ===**

<table>
<thead>
<tr>
<th></th>
<th>a = good</th>
<th>b = bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>663</td>
<td>118</td>
</tr>
<tr>
<td>b</td>
<td>37</td>
<td>182</td>
</tr>
</tbody>
</table>

Remember to reload the previous removed attribute, press Undo option in Preprocess tab. We use the **Preprocess Tab** in Weka GUI Explorer to remove 17th attribute (Job). In Classify Tab, Select Use Training set option then Press Start Button. If these attributes removed from the dataset, we can see change in the accuracy compare to full data set when we removed.

--- Evaluation on training set ===
--- Summary ===

<table>
<thead>
<tr>
<th>Correctly Classified Instances</th>
<th>859</th>
<th>85.9 %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incorrectly Classified Instances</td>
<td>141</td>
<td>14.1 %</td>
</tr>
</tbody>
</table>

--- Confusion Matrix ===

<table>
<thead>
<tr>
<th></th>
<th>a = good</th>
<th>b = bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>675</td>
<td>116</td>
</tr>
<tr>
<td>b</td>
<td>25</td>
<td>184</td>
</tr>
</tbody>
</table>

Remember to reload the previous removed attribute, press Undo option in Preprocess tab. We use the **Preprocess Tab** in Weka GUI Explorer to remove 21st attribute (Class). In Classify Tab, Select Use Training set option then Press Start Button. If these attributes removed from the dataset, we can see change in the accuracy compare to full data set when we removed.

--- Evaluation on training set ===
--- Summary ===

<table>
<thead>
<tr>
<th>Correctly Classified Instances</th>
<th>963</th>
<th>96.3 %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incorrectly Classified Instances</td>
<td>37</td>
<td>3.7 %</td>
</tr>
</tbody>
</table>

--- Confusion Matrix ===

<table>
<thead>
<tr>
<th></th>
<th>a = good</th>
<th>b = bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>963</td>
<td>37</td>
</tr>
<tr>
<td>b</td>
<td>37</td>
<td>963</td>
</tr>
</tbody>
</table>
963 0 | a = yes
37 0 | b = no

**Note:** With this observation we have seen, when 3\textsuperscript{rd} attribute is removed from the Dataset, the accuracy (83\%) is decreased. So this attribute is important for classification. When 2\textsuperscript{nd} and 10\textsuperscript{th} attributes are removed from the Dataset, the accuracy (84\%) is same. So we can remove any one among them. When 7\textsuperscript{th} and 17\textsuperscript{th} attributes are removed from the Dataset, the accuracy (85\%) is same. So we can remove any one among them. If we remove 5\textsuperscript{th} and 21\textsuperscript{st} attributes the accuracy is increased, so these attributes may not be needed for the classification.

9. Sometimes, the cost of rejecting an applicant who actually has good credit might be higher than accepting an applicant who has bad credit. Instead of counting the misclassification equally in both cases, give a higher cost to the first case (say cost 5) and lower cost to the second case. By using a cost matrix in weak. Train your decision tree and report the Decision Tree and cross validation results. Are they significantly different from results obtained in problem 6.

Ans) steps followed are:
1. Double click on credit-g.arff file.
2. Click on classify tab.
3. Click on choose button.
4. Expand tree folder and select J48
5. Click on start
6. Note down the accuracy values
7. Now click on credit arff file
8. Click on attributes 2,3,5,7,10,17,21
9. Click on invert
10. Click on classify tab
11. Choose J48 algorithm
12. Select Cross validation fold as 2
13. Click on start and note down the accuracy values.
14. Again make cross validation folds as 10 and note down the accuracy values.
15. Again make cross validation folds as 20 and note down the accuracy values.
OUTPUT:

In Weka GUI Explorer, Select Classify Tab, In that Select **Use Training set**option . In Classify Tab then press **Choose** button in that select J48 as Decision Tree Technique. In Classify Tab then press **More options** button then we get classifier evaluation options window in that select cost sensitive evaluation the press set option Button then we get Cost Matrix Editor. In that change classes as 2 then press Resize button. Then we get 2X2 Cost matrix. In Cost Matrix (0,1) location value change as 5, then we get modified cost matrix is as follows.

\[
\begin{array}{cc}
0.0 & 5.0 \\
1.0 & 0.0
\end{array}
\]

Then close the cost matrix editor, then press ok button. Then press start button.

```java
=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances 855  85.5  
Incorrectly Classified Instances 145  14.5  

=== Confusion Matrix ===

a b -- classified as
669 31 | a = good 114
186  | b = bad
114

Note: With this observation we have seen that ,total 700 customers in that 669 classified as good customers and 31 misclassified as bad customers. In total 300 cusotmers, 186 classified as bad customers and 114 misclassified as good customers.

10. Do you think it is a good idea to prefect simple decision trees instead of having long complex decision tress? How does the complexity of a Decision Tree relate to the bias of the model?

Ans)

steps followed are:-
1)click on credit arff file
2)Select all attributes
3)click on classify tab
4)click on choose and select J48 algorithm
5)select cross validation folds with 2
6)click on start
7) write down the time complexity value

It is a good idea to prefer simple Decision trees, instead of having complex Decision tree.

11. You can make your Decision Trees simpler by pruning the nodes. One approach is to use Reduced Error Pruning. Explain this idea briefly. Try reduced error pruning for training your Decision Trees using cross validation and report the Decision Trees you obtain? Also, report your accuracy using the pruned model. Does your accuracy increase?

Ans)

Steps followed are:
1) click on credit arff file
2) Select all attributes
3) click on classify tab
4) click on choose and select REP algorithm
5) select cross validation 2
6) click on start
7) Note down the results

We can make our decision tree simpler by pruning the nodes. For that In Weka GUI Explorer, Select Classify Tab, In that Select Use Training set option. In Classify Tab then press Choose button in that select J48 as Decision Tree Technique. Beside Choose Button Press on J48 –c 0.25–M2 text we get Generic Object Editor. In that select Reduced Error pruning Property as True then press ok. Then press start button.

=== Evaluation on training set ===
=== Summary ===
Correctly Classified Instances    786    78.6 %
Incorrectly Classified Instances  214    21.4 %

== Confusion Matrix ==

a b <-- classified as
662 38 | a = good 176
124 | b = bad

By using pruned model, the accuracy decreased. Therefore by pruning the nodes we can make our decision tree simpler.
12) How can you convert a Decision Tree into “if-then-else rules”. Make up your own small

Decision Tree consisting 2-3 levels and convert into a set of rules. There also exist
different classifiers that output the model in the form of rules. One such classifier in
weka is rules. PART, train this model and report the set of rules obtained. Sometimes just
one attribute can be good enough in making the decision, yes, just one ! Can you predict
what attribute that might be in this data set? OneR classifier uses a single attribute to
make decisions(it chooses the attribute based on minimum error). Report the rule
obtained by training a one R classifier. Rank the performance of j48, PART, oneR.

Ans)

Steps For Analyze Decision
   Tree: 1) click on credit arff file
   2) Select all attributes
   3) click on classify tab
   4) click on choose and select J48 algorithm
   5) select cross validation folds with 2
   6) click on start
   7) note down the accuracy value
   8) again goto choose tab and select PART
   9) select cross validation folds with 2
   10) click on start
   11) note down accuracy value
   12) again goto choose tab and select One R
   13) select cross validation folds with 2
   14) click on start
   15) note down the accuracy value.
Sample Decision Tree of 2-3 levels.

Converting Decision tree into a set of rules is as follows.

Rule1: If age = youth AND student=yes THEN buys_computer=yes
Rule2: If age = youth AND student=no THEN buys_computer=no
Rule3: If age = middle_aged THEN buys_computer=yes
Rule4: If age = senior AND credit_rating=excellent THEN buys_computer=yes
Rule5: If age = senior AND credit_rating=fair THEN buys_computer=no

In Weka GUI Explorer, Select Classify Tab, In that Select Use Training set option. There also exist different classifiers that output the model in the form of Rules. Such classifiers in weka are “PART” and “OneR”. Then go to Choose and select Rules in that select PART and press start Button.

== Evaluation on training set ===
=== Summary ===
Correctly Classified Instances 897  89.7  
Incorrectly Classified Instances 103  10.3  

== Confusion Matrix ==

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>653</td>
<td>47</td>
</tr>
<tr>
<td>b</td>
<td>56</td>
<td>244</td>
</tr>
</tbody>
</table>

classified as
653 47 | a = good
56 244 | b = bad

Then go to Choose and select Rules in that select OneR and press start Button.

== Evaluation on training set ==

=== Summary ===
Correctly Classified Instances 742  74.2  
Incorrectly Classified Instances 258  25.8  

== Confusion Matrix ==

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>642</td>
<td>58</td>
</tr>
<tr>
<td>b</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

classified as
642 58 | a = good 200
100     | b = bad

Then go to Choose and select Trees in that select J48 and press start Button.

=== Evaluation on training set ==

=== Summary ===
Correctly Classified Instances 855  85.5  
Incorrectly Classified Instances 145  14.5  

=== Confusion Matrix ===

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>669</td>
<td>31</td>
</tr>
<tr>
<td>b</td>
<td>186</td>
<td></td>
</tr>
</tbody>
</table>

classified as
669 31 | a = good 114
186     | b = bad

Note: With this observation we have seen the performance of classifier and Rank is as follows
1. PART
2. J48 3. OneR
7. Task 2: Hospital Management System

Data warehouse consists dimension table and fact table.

REMEMBER the following

Dimension

The dimension object(dimension);

_name

_attributes(levels), with primary key

_hierarchies

One time dimension is must.

About levels and hierarchies

Dimensions objects(dimension) consists of set of levels and set of hierarchies defined over those levels. The levels represent levels of aggregation. Hierarchies describe-child relationships among a set of levels.

For example, a typical calendar dimension could contain five levels. Two hierarchies can be defined on these levels.

H1: YearL>QuarterL>MonthL>DayL

H2: YearL>WeekL>DayL

The hierarchies are describes from parent to child, so that year is the parent of Quarter, quarter are parent of month, and so forth.

About Unique key constraints

When you create a definition for a hierarchy, warehouse builder creates an identifier key for each level of the hierarchy and unique key constraint on the lowest level (base level).

Design a hospital management system data warehouse(TARGET) consists of dimensions patient, medicine, supplier, time. Where measure are ‘NO UNITS’, UNIT PRICE.

Assume the relational database(SOURCE) table schemas as follows TIME(day, month, year)

PATIENT(patient_name, age, address, etc)
MEDICINE(Medicine_brand_name, Drug_name, supplier, no_units, units_price, etc...)

SUPPLIER:( Supplier_name, medicine_brand_name, address, etc...)

If each dimension has 6 levels, decide the levels and hierarchies, assumes the level names suitably.

Design the hospital management system data warehousing using all schemas. Give the example 4-D cube with assumption names.
8. Simple Project on Data Preprocessing

Data Preprocessing

Objective: Understanding the purpose of unsupervised attribute-instance filters for preprocessing the input data.

Follow the steps mentioned below to configure and apply a filter.

The preprocess section allows filters to be defined that transform the data in various ways. The Filter box is used to set up 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 brings up a GenericObjectEditor dialog box, which lets you configure a filter. Once you are happy with the settings you have chosen, click OK to return to the main Explorer window.

Now you can apply it to the data by pressing the Apply button at the right end of the Filter panel. The Preprocess panel will then show the transformed data. The change can be undone using the Undo button. Use the Edit button to view your transformed data in the dataset editor.

Try each of the following Unsupervised Attribute Filters. (Choose -> weka -> filters -> unsupervised -> attribute)

- Use ReplaceMissingValues to replace missing values in the given dataset.
- Use the filter Add to add the attribute Average.
- Use the filter AddExpression and add an attribute which is the average of attributes M1 and M2. Name this attribute as AVG.
- Understand the purpose of the attribute filter Copy.
- Use the attribute filters Discretize and PKIDiscretize to discretize the M1 and M2 attributes into five bins. (NOTE: Open the file afresh to apply the second filter.
since there would be no numeric attribute to discretize after you have applied the first filter.)

- Perform **Normalize** and **Standardize** on the dataset and identify the difference between these operations.

- Use the attribute filter **FirstOrder** to convert the M1 and M2 attributes into a single attribute representing the first differences between them.

- Add a nominal attribute Grade and use the filter **MakeIndicator** to convert the attribute into a Boolean attribute.

- Try if you can accomplish the task in the previous step using the filter **MergeTwoValues**.

- Try the following transformation functions and identify the purpose of each:
  - **NumericTransform**
  - **NominalToBinary**
  - **NumericToBinary**
  - **Remove**
  - **RemoveType**
  - **RemoveUseless**
  - **ReplaceMissingValues**
  - **SwapValues**

Try the following **Unsupervised Instance Filters**.

(Choose -> weka -> filters -> unsupervised -> instance)

- Perform **Randomize** on the given dataset and try to correlate the resultant sequence with the given one.

- Use **RemoveRange** filter to remove the last two instances.

- Use **RemovePercent** to remove 10 percent of the dataset.

- Apply the filter **RemoveWithValues** to a nominal and a numeric attribute.
Topic Beyond the Syllabus

Creating Transformations

The Data Integration perspective of Spoon allows you to create two basic document types: transformations and jobs. Transformations are used to describe the data flows for ETL such as reading from a source, transforming data and loading it into a target location. Jobs are used to coordinate ETL activities such as defining the flow and dependencies for what order transformations should be run, or prepare for execution by checking conditions such as, "Is my source file available?" or "Does a table exist in my database?"

This exercise will step you through building your first transformation with Pentaho Data Integration introducing common concepts along the way. The exercise scenario includes a flat file (CSV) of sales data that you will load into a database so that mailing lists can be generated. Several of the customer records are missing postal codes (zip codes) that must be resolved before loading into the database. The logic looks like this:

Retrieving Data

1. Click New in the upper left corner of the Spoon graphical interface.
2. Select Transformation from the list.
3. Under the Design tab, expand the Input node; then, select and drag a Text File Input step onto the canvas on the right.

Retrieving Data from a Flat File (Text File Input Step)

Follow the instructions below to retrieve data from a flat file.

1. Click New in the upper left corner of the Spoon graphical interface.
2. Select Transformation from the list.
3. Under the Design tab, expand the Input node; then, select and drag a Text File Input step onto the canvas on the right.
4. Double-click on the Text File input step. The edit properties dialog box associated with the Text File input step appears. In this dialog box, you specify the properties related to a particular step.

![Text File Input Step](image)

5. In the Step Name field, type Read Sales Data.
   You are renaming the Text File Input step to Read Sales Data.

6. Click Browse to locate the source file, sales_data.csv, available at ..\design-tools\data-integration\samples\transformations\files. The path to the source file appears in the File or directory field.
7. Click **Add**. The path to the file appears under **Selected Files**. You can look at the contents of the file by clicking the **Show file content** to determine things such as how the input file is delimited, what enclosure character is used, and whether or not a header row is present. In the example, the input file is comma (,) delimited, the enclosure character being a quotation mark ("), and it contains a single header row containing field names.

8. Click the **Content** tab.

The fields under the **Content** tab allow you to define how your data is formatted.

9. Make sure that the **Separator** is set to comma (,) and that the **Enclosure** is set to quotation mark ("). Enable **Header** because there is one line of header rows in the file.

10. Click the **Fields** tab and click **Get Fields** to retrieve the input fields from your source file.

A dialog box appears requesting that you to specify the number of lines to scan, allowing you to determine default settings for the fields such as their format, length, and precision. Type 0 (zero) in the **Number of Sample Lines** textbox to scan all lines. By scanning all lines, you ensure that Pentaho Data Integration has read the entire content of the file and you reduce the possibility of errors that may cause a transformation not to run. Click **OK** and the summary of the scan results appears. Once you are done examining the scan results, click **Close** to return to the step properties editor.

11. Click **Preview Rows** to verify that your file is being read correctly. You can change the number of rows to preview. Click **OK** to exit the step properties dialog box.
12. Save your transformation.

Saving Your Transformation
Follow the instructions below to save your transformation.

Note: You can save your transformation at any point in this walk through. Saving allows you to start and stop the exercises at your convenience.

1. In the Spoon designer, click **File -> Save As**.

The Transformation Properties dialog box appears.

2. In the **Transformation Name** field, type **Getting Started Transformation**

3. In the **Directory** field, click (folder icon) to select a repository folder where you will save your transformation.
4. Expand the Home directory and double-click the joe folder.
   Your transformation will be stored in the joe folder in the Enterprise Repository.
5. Click OK to exit the Transformation Properties dialog box.
   The Enter Comment dialog box appears.
6. Click in the Enter Comment dialog box and press <Delete> to remove the default text string.
   Type a meaningful comment about your transformation. The comment and your transformation
   are tracked for version control purposes in the Enterprise Repository.
7. Click OK to exit the Enter Comment dialog box.

Filter Records with Missing Postal Codes (Filter Rows Step)
The source file contains several records that are missing postal codes. You will now use the
Filter Rows transformation step to separate out those records so that you can resolve them in a
later exercise.
1. Add a Filter Rows step to your transformation. Under the Design tab, go to Flow -> Filter Rows.
2. Create a "hop" between the Read Sales Data (Text File Input) step and the Filter Rows step.
   Hops are used to describe the flow of data in your transformation. To create the hop, click the
   Read Sales Data (Text File Input) step, then press the <SHIFT> key down and draw a line to
   the Filter Rows step.

3. Double-click the Filter Rows step. The Filter Rows edit properties dialog box appears.
4. In the Step Name field type, Filter Missing Zips.
5. Under The condition, click <field>. A dialog box that contains the fields you can use to create
   your condition appears.
6. In the **Fields:** dialog box select **POSTALCODE** and click **OK**.

7. Click on the comparison operator (set to = by default) and select the **IS NOT NULL** function and click **OK**. Click **OK** to exit the Filter Rows properties dialog box.

8. Save your transformation.

**Loading Your Data into a Relational Database (Table Output Step)**

In this exercise you will take all records exiting the Filter rows step where the POSTALCODE was not null (the **true** condition) and load them into a database table.

1. Under the Design tab, expand the contents of the **Output** node.
2. Click and drag a **Table Output** step into your transformation; create a hop between the **Filter Missing Zips** (Filter Rows) and **Table Output** steps. Select **Result is TRUE**.
3. Double-click the **Table Output** step to open its edit properties dialog box.
4. Rename your Table Output Step to **Write to Database**.
5. Click **New** next to the **Connection** field. You must create a connection to the database.

6. Provide the settings for connecting to the database as shown in the table below.

<table>
<thead>
<tr>
<th>Connection Name</th>
<th>Type, <strong>Sample Data</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Choose, H2</td>
</tr>
<tr>
<td>Host Name</td>
<td>localhost</td>
</tr>
<tr>
<td>Database Name</td>
<td>Type <strong>sampledata</strong></td>
</tr>
<tr>
<td>Port Number</td>
<td>9092</td>
</tr>
</tbody>
</table>
7. Click **Test** to make sure your entries are correct. A success message appears. Click **OK**

8. Click **OK**, to exit the Database Connections dialog box.

Type **SALES_DATA** in the **Target Table** text field.

This table does not exist in the target database. In the next steps you will generate the Data Definition Language (DDL) to create the table and execute it. DDL are the SQL commands that define the different structures in a database such as CREATE TABLE.

10. In the **Write to Database** edit properties dialog box, enable the **Truncate Table** property.

11. Click **SQL** to generate the DDL for creating your target table.

12. Click **Execute** to run the SQL.

A results dialog box appears indicating that one SQL statement was executed. Click **OK** to close the execution dialog box. Click **Close** to close the Simple SQL editor dialog box. Click **OK** to close the Table Output edit properties dialog box.

13. Save your transformation.

---

**Retrieving Data from your Lookup File (Text File Input Step)**

You have been provided a second text file containing a list of cities, states, and postal codes that you will now use to look up the postal codes for all of the records where they were missing (the ‘false’ branch of your Filter rows step). First, you will use a Text file input step to read from the source file, next you will use a Stream lookup step to bring the resolved Postal Codes into the stream.

1. Add a new **Text File Input** step to your transformation. In this step you will retrieve the records from your lookup file.

2. Rename your **Text File input** step to, **Read Postal Codes**.

3. Click **Browse** to locate the source file, **Zipssortedbycitystate.csv**, located at ...\design-tools\dataintegration\samples\transformations\files.

4. Click **Add**. The path to the file appears under **Selected Files**.

5. Under the **Content** tab, enable the **Header** option. Change the separator character to a comma (,), and confirm that the enclosure setting is correct.

6. Under the **Fields** tab, click **Get Fields** to retrieve the data from your .csv file.

7. Click **Preview Rows** to make sure your entries are correct and click **OK** to exit the Text File input properties dialog box.
8. Save your transformation.

**Resolving Missing Zip Code Information (Stream Lookup Step)**

In this exercise, you will begin to resolve the missing zip codes.

1. Add a Stream Lookup step to your transformation. Under the Design tab, expand the Lookup folder and choose Stream Lookup.
2. Draw a hop between the Filter Missing Zips (Filter rows) and Stream Lookup steps. Select the Result is FALSE.
3. Create a hop from the Read Postal Codes step (Text File input) to the Stream lookup step.
4. Double-click on the Stream lookup step to open its edit properties dialog box.
5. Rename Stream Lookup to Lookup Missing Zips.

6. Select the Read Postal Codes (Text File input) as the Lookup step.
7. Define the CITY and STATE fields in the key(s) to look up the value(s) table. Click the drop down in the Field column and select CITY. Then, click in the LookupField column and select CITY. Perform the same actions to define the second key based on the STATE fields coming in on the source and lookup streams.
8. Click **Get Lookup Fields**. **POSTALCODE** is the only field you want to retrieve. (To delete the extra **CITY** and **STATE** lines, right-click in the line and select **Delete Selected Line**.) Give **POSTALCODE** a new name of **ZIP_RESOLVED** and make sure the **Type** is set to **String**. Click **OK** to close the **Stream Lookup** edit properties dialog box.

9. Save your transformation. You can now select the **Lookup Missing Zips** step (Stream lookup) in the graphical workspace. Right-click and select **Preview** to display the preview/debugger dialog box. Click **Quick Launch** to preview the data flowing through this step. Notice that the new field, **ZIP_RESOLVED**, has been added to the stream containing your resolved postal codes.

**Completing your Transformation (Select Values Step)**

The last task is to clean up the field layout on your lookup stream so that it matches the format and layout of your other stream going to the **Write to Database** (Table output) step. You will create a **Select values** step. This is a very useful step for renaming fields on the stream, removing unnecessary fields, and more.

1. Add a **Select Values** step to your transformation. Expand the **Transform** folder and choose **Select Values**.
2. Create a hop between the **Lookup Missing Zips** and **Select Values** steps.
3. Double-click the **Select Values** step to open its properties dialog box.
4. Rename the **Select Values** step to, **Prepare Field Layout**.
5. Click **Get fields to select** to retrieve all fields and begin modifying the stream layout.
6. Select the **ZIP_RESOLVED** field in the **Fields** list and use <CTRL><UP> to move it just below the **POSTALCODE** field (the one that still contains null values).
7. Select the old **POSTALCODE** field in the list (line 20) and delete it.
8. The original POSTALCODE field was formatted as an 9-character string. You must modify your new field to match the form. Click the Meta-Data tab.
9. In the first row of the Fields to alter table, click in the Fieldname column and select ZIP_RESOLVED.
10. Type POSTALCODE in the Rename to column; select String in the Type column, and type 9 in the Length column. Click OK to exit the edit properties dialog box.
11. Draw a hop from the Prepare Field Layout (Select values) step to the Write to Database (Table output) step.
12. Save your transformation.

Running Your Transformation
1. In the Spoon graphical interface, click Run This Transformation. The Execute a Transformation dialog box appears. You can run a transformation locally, remotely, or in a clustered environment. For the purposes of this exercise, keep the default Local Execution.
2. Click Launch. The transformation executes. Upon running the transformation, the Execution Results panel opens below the graphical workspace.
The **Step Metrics** tab provides statistics for each step in your transformation including how many records were read, written, caused an error, processing speed (rows per second) and more. If any of the steps caused the transformation to fail, they would be highlighted in red as shown below.

The **Logging** tab displays the logging details for the most recent execution of the transformation. Error lines are highlighted in red.
You can see that in this case the **Lookup Missing Zips** step caused an error because it attempted to lookup values on a field called POSTALCODE2, which did not exist in the lookup stream.

The **Execution History** tab provides you access to the Step Metrics and log information from previous executions of the transformation. This feature works only if you have configured your transformation to log to a database through the Logging tab of the Transformation Settings dialog. For more information on configuring logging or viewing the execution history, see the *Pentaho Data Integration User Guide* found in the Pentaho InfoCenter. The **Performance Graph** allows you to analyze the performance of steps based on a variety of metrics including how many records were read, written, caused an error, processing speed (rows per second) and more.