DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road, Delhi 110042

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CO-405 : Data Warehouse and Data Mining Lab File

Submitted To:
Mr Ankur
Faculty
Department of Computer Science and Engineering

Submitted By:
Kunal Sinha
B.Tech Computer Science
8th Semester
2K17/CO/164

INDEX

S.No.	Topic	Date	Signature
1	Briefly explain various tools used for Data Warehousing and Data Mining.	12-01-2021	
2	List out various open-source data mining tools and techniques and explain them.	19-01-2021	
3	Generate Histogram and Boxplot for a sample program in WEKA.	02-02-2021	
4	a. Identify two data sets in which baseline classification is better then j48. Show the results using WEKA. b. Calculate the sample mean and SD for any data set(at least 1500 instant and 6 classes) by setting the random number seed from 1 to 10 using WEKA. c. Show the results for handout(10%) over 10-cross validation using WEKA.	09-02-2021	
5	a. To perform decision tree learning for classification in WEKA. b. WAP to implement the decision tree in python.	23-02-2021	
6	a. To perform a simple k mean clustering algorithm on any data set(at least 1500 instant and 6 classes). b. WAP to implement k mean clustering algorithm in python.	02-03-2021	
7	To perform Association rule mining on any sample dataset using any two algorithms.	09-03-2021	
8	WAP to implement the Apriori algorithm in C++.	16-03-2021	
9	To perform ROC analysis, Forecast analysis and Survival analysis on any sample dataset.	23-03-2021	
10	WAP to implement the KNN and model evaluation in python.	06-04-2021	

Experiment 1

Aim: Briefly explain various tools used for Data Warehouse and Data Mining.

Theory:

Data Warehouse

In computing, a data warehouse, also known as an enterprise data warehouse, is a system used for reporting and data analysis, and is considered a core component of business intelligence. DWs are central repositories of integrated data from one or more disparate sources. Some of the most popular open-source tools used for Data Warehouse are:

1. **QuerySurge:** QuerySurge is an ETL testing solution developed by RTTS. It is built specifically to automate the testing of Data Warehouses & Big Data. It ensures that the data extracted from data sources remains intact in the target systems as well. Some of its features:
   - Improve data quality, data governance and accelerate your data delivery cycles
   - It helps to automate the manual testing effort and also provides up to 100% data coverage
   - Provide testing across different platforms like Oracle, Teradata, IBM, Amazon etc.
   - It integrates an out-of-the-box DevOps solution for most Build, ETL & QA software
   - Deliver shareable, automated email reports and data health dashboards

2. **Oracle:** Oracle data warehouse software is a collection of data which is treated as a unit. The purpose of this database is to store and retrieve related information. It helps the server to reliably manage huge amounts of data so that multiple users can access the same data.
   - Distributes data in the same way across disks to offer uniform performance
   - Works for single-instance and real application clusters
   - Offers real application testing
   - Common architecture between any Private Cloud and Oracle's public cloud
   - Hi-Speed Connection to move large data
   - Works seamlessly with UNIX/Linux and Windows platforms
   - It provides support for virtualization
   - Allows connecting to the remote database, table, or view

3. **Amazon Redshift:** Amazon Redshift is easy to manage, simple, and cost-effective data warehouse tool. It can analyze almost every type of data using standard SQL.
   - No Up-Front Costs for its installation
   - It allows automating administrative tasks to monitor, manage, and scale your DW
   - Possible to change the number or type of nodes.
   - Helps to enhance the reliability of the data warehouse cluster
   - Every data centre is fully equipped with climate control

- Continuously monitors the health of the cluster. It automatically re-replicates data from failed drives and replaces nodes when needed

4. **Domo:** Domo is a cloud-based Data warehouse management tool that easily integrates various types of data sources, including spreadsheets, databases, social media and almost all cloud-based or on-premise Data warehouse solutions.
   - Help you to build your dream dashboard and Integrates all existing business data
   - Stay connected anywhere you go
   - Helps you to get true insights into your business data
   - Easy Communication & messaging platform
   - It provides support for ad-hoc queries using SQL
   - It can handle most concurrent users for running complex and multiple queries

5. **Teradata Corporation:** The Teradata Database is the only commercially available shared-nothing or Massively Parallel Processing (MPP) data warehousing tool. It is one of the best data warehousing tool for viewing and managing large amounts of data. Features:
   - Simple and Cost Effective solutions with quick and most insightful analytics
   - The tool is best suitable option for an organization of any size
   - Get the same Database on multiple deployment options
   - It allows multiple concurrent users to ask complex questions related to data
   - Offers High performance, diverse queries, and sophisticated workload management

6. **SAP:** SAP is an integrated data management platform, to maps all business processes of an organization. It is an enterprise-level application suite for open client/server systems. It has set new standards for providing the best business information management solutions.
   - It provides highly flexible and most transparent business solutions
   - The application developed using SAP can integrate with any system
   - It follows modular concept for the easy setup and space utilization
   - You can create a Database system that combines analytics and transactions. These next
   - next-generation databases can be deployed on any device
   - Provide support for On-premise or cloud deployment
   - Simplified data warehouse architecture
   - Integration with SAP and non-SAP applications

7. **IBM – DataStage:** IBM data Stage is a business intelligence tool for integrating trusted data across various enterprise systems. It leverages a high-performance parallel framework either in the cloud or on-premise. This data warehousing tool supports extended metadata management and universal business connectivity.
   - Support for Big Data and Hadoop

- Additional storage/ services can be accessed without installing new software or hardware.
- Real time data integration
- Provide trusted ETL data anytime, anywhere and solve complex big data challenges
- Optimize hardware utilization and prioritize mission-critical tasks
- Deploy on-premises or in the cloud

8. **Informatica:** Informatica PowerCenter is Data Integration tool developed by Informatica Corporation. The tool offers the capability to connect & fetch data from different sources.
   - It has a centralized error logging system which facilitates logging errors and rejecting data into relational tables
   - Build in Intelligence to improve performance
   - Ability to Scale up Data Integration with enforced best practices on code development.
   - Foundation for Data Architecture Modernization
   - Code integration with external Software Configuration tools
   - Synchronization amongst geographically distributed team members

Data Mining

Data mining is a process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems. Some of the most popular open-source tools used for Data Mining are:

1. **Rapid Miner:** RapidMiner is one of the best predictive analysis systems developed by the company with the same name as the Rapid Miner. It is written in JAVA programming language. It provides an integrated environment for deep learning, text mining, machine learning & predictive analysis. The tool can be used for over a vast range of applications including for business applications, commercial applications, training, education, research, application development, machine learning.
   Rapid Miner offers the server as both on premise & in public/private cloud infrastructures. It has a client/server model as its base. Rapid Miner comes with template based frameworks that enable speedy delivery with reduced number of errors (which are quite commonly expected in manual code writing process). Rapid Miner constitutes of three modules, namely
   - RapidMiner Studio- This module is for workflow design, prototyping, validation etc.
   - RapidMiner Server- To operate predictive data models created in studio
   - RapidMiner Radoop- Executes processes directly in the Hadoop cluster to simplify predictive analysis.

2. **Orange:** Orange is a perfect software suite for machine learning & data mining. It best aids the data visualization and is a component based software. It has been written in Python computing language. As it is a component-based software, the components of orange are

called ‘widgets’. These widgets range from data visualization & pre-processing to an evaluation of algorithms and predictive modeling. Widgets offer major functionalities like

- Showing data table and allowing to select features
- Reading the data and training predictors and to compare learning algorithms
- Visualizing data elements etc.

Additionally, Orange brings a more interactive and fun vibe to the dull analytic tools. It is quite interesting to operate. Data coming to Orange gets quickly formatted to the desired pattern and it can be easily moved where needed by simply moving/flipping the widgets. Users are quite fascinated by Orange. Orange allows users to make smarter decisions in a short time by quickly comparing & analyzing the data.

3. **Weka:** Also known as Waikato Environment is a machine learning software developed at the University of Waikato in New Zealand. It is best suited for data analysis and predictive modeling. It contains algorithms and visualization tools that support machine learning. Weka has a GUI that facilitates easy access to all its features. It is written in JAVA programming language. Weka supports major data mining tasks including data mining, processing, visualization, regression etc. It works on the assumption that data is available in the form of a flat file. Weka can provide access to SQL Databases through database connectivity and can further process the data/results returned by the query.

4. **KNIME:** KNIME is the best integration platform for data analytics and reporting developed by KNIME.com AG. It operates on the concept of the modular data pipeline. KNIME constitutes of various machine learning and data mining components embedded together. KNIME has been used widely for pharmaceutical research. In addition, it performs excellently for customer data analysis, financial data analysis, and business intelligence. KNIME has some brilliant features like quick deployment and scaling efficiency. Users get familiar with KNIME in quite lesser time and it has made predictive analysis accessible to even naive users. KNIME utilizes the assembly of nodes to pre-process the data for analytics and visualization.

5. **Apache Mahout:** Apache Mahout is a project developed by Apache Foundation that serves the primary purpose of creating machine learning algorithms. It focuses mainly on data clustering, classification, and collaborative filtering. Mahout is written in JAVA and includes JAVA libraries to perform mathematical operations like linear algebra and statistics. Mahout is growing continuously as the algorithms implemented inside Apache Mahout are continuously growing. The algorithms of Mahout have implemented a level above Hadoop through mapping/reducing templates. To key up, Mahout has following major features
   - Extensible programming environment
   - Pre-made algorithms and math experimentation environment

- GPU computes for performance improvement.

6. **DataMelt:** DataMelt, also known as DMelt is a computation and visualization environment that provides an interactive framework to do data analysis and visualization. It is designed mainly for engineers, scientists & students. DMelt is written in JAVA and it is a multi-platform utility. It can run on any operating system which is compatible with JVM(Java Virtual Machine). It contains Scientific & mathematical libraries. Scientific libraries: To draw 2D/3D plots. Mathematical libraries: To generate random numbers, curve fitting, algorithms etc.DataMelt can be used for analysis of large data volumes, data mining, and stat analysis. It is widely used in the analysis of financial markets, natural sciences & engineering.

Findings and Learnings :

We learned about data warehousing and data mining and their respective significance. We also learned about the open source tools which can be used to assist us in the practice of data warehousing and data mining. Finally we learned about the features of the tools used for data warehouse and data mining.

Experiment 2

Aim: List out various open-source data mining tools and techniques and explain them.

Theory:

Data Mining

Data mining is a process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems.
Some of the most popular open-source tools used for Data Mining are:

1. **Weka:** Also known as Waikato Environment is a machine learning software developed at the University of Waikato in New Zealand. It is best suited for data analysis and predictive modelling. It contains algorithms and visualization tools that support machine learning. Weka has a GUI that facilitates easy access to all its features. It is written in the JAVA programming language. Weka supports major data mining tasks including data mining, processing, visualization, regression etc. It works on the assumption that data is available in the form of a flat-file. Weka can provide access to SQL Databases through database connectivity and can further process the data/results returned by the query.

2. **Rapid Miner:** RapidMiner is one of the best predictive analysis systems developed by the company with the same name as Rapid Miner. It is written in the JAVA programming language. It provides an integrated environment for deep learning, text mining, machine learning & predictive analysis. The tool can be used for a vast range of applications including business applications, commercial applications, training, education, research, application development, machine learning.
   Rapid Miner offers the server as both on-premise & in public/private cloud infrastructures. It has a client/server model as its base. Rapid Miner comes with template-based frameworks that enable speedy delivery with a reduced number of errors (which are quite commonly expected in the manual code writing process). Rapid Miner constitutes of three modules, namely
   - RapidMiner Studio- This module is for workflow design, prototyping, validation etc.
   - RapidMiner Server- To operate predictive data models created in the studio
   - RapidMiner Radoop- Executes processes directly in the Hadoop cluster to simplify predictive analysis.

3. **Orange:** Orange is a perfect software suite for machine learning & data mining. It best aids data visualization and is a component-based software. It has been written in Python computing language. As it is a component-based software, the components of orange are

called ‘widgets’. These widgets range from data visualization & pre-processing to an evaluation of algorithms and predictive modelling. Widgets offer major functionalities like

- Showing data table and allowing to select features
- Reading the data and training predictors and to compare learning algorithms
- Visualizing data elements etc.

Additionally, Orange brings a more interactive and fun vibe to the dull analytic tools. It is quite interesting to operate. Data coming to Orange gets quickly formatted to the desired pattern and it can be easily moved where needed by simply moving/flipping the widgets. Users are quite fascinated by Orange. Orange allows users to make smarter decisions in a short time by quickly comparing & analyzing the data.

4. **KNIME:** KNIME is the best integration platform for data analytics and reporting developed by KNIME.com AG. It operates on the concept of the modular data pipeline. KNIME constitutes various machine learning and data mining components embedded together. KNIME has been used widely for pharmaceutical research. In addition, it performs excellently for customer data analysis, financial data analysis, and business intelligence. KNIME has some brilliant features like quick deployment and scaling efficiency. Users get familiar with KNIME in quite a lesser time and it has made predictive analysis accessible to even naive users. KNIME utilizes the assembly of nodes to pre-process the data for analytics and visualization.

5. **Apache Mahout:** Apache Mahout is a project developed by the Apache Foundation that serves the primary purpose of creating machine learning algorithms. It focuses mainly on data clustering, classification, and collaborative filtering. Mahout is written in JAVA and includes JAVA libraries to perform mathematical operations like linear algebra and statistics. Mahout is growing continuously as the algorithms implemented inside Apache Mahout are continuously growing. The algorithms of Mahout have implemented a level above Hadoop through mapping/reducing templates. To key up, Mahout has the following major features
   - Extensible programming environment
   - Pre-made algorithms and math experimentation environment
   - GPU computes for performance improvement.

6. **DataMelt:** DataMelt, also known as DMelt is a computation and visualization environment that provides an interactive framework to do data analysis and visualization. It is designed mainly for engineers, scientists & students. DMelt is written in JAVA and is a multi-platform utility. It can run on any operating system which is compatible with JVM. It contains Scientific & mathematical libraries. Scientific libraries: To draw 2D/3D plots. Mathematical libraries: To generate random numbers, curve fitting, algorithms etc.DataMelt can be used for

the analysis of large data volumes, data mining, and stat analysis. It is widely used in the analysis of financial markets, natural sciences & engineering.

There are several major data mining techniques that have been developing and using in data mining projects recently including association, classification, clustering, prediction, sequential patterns and decision tree.

1. **Association:** Association is one of the best-known data mining technique. In association, a pattern is discovered based on a relationship between items in the same transaction. The association technique is used in market basket analysis to identify a set of products that customers frequently purchase together. Retailers are using the association technique to research customer's buying habits. Based on historical sale data, retailers might find out that customers always buy crisps when they buy beers, and, therefore, they can put beers and crisps next to each other to save time for the customer and increase sales.

2. **Classification:** Classification is a classic data mining technique based on machine learning. Basically, classification is used to classify each item in a set of data into one of a predefined set of classes or groups. The classification method makes use of mathematical techniques such as decision trees, linear programming, neural network, and statistics. In classification, we develop software that can learn how to classify the data items into groups. For example, we can apply classification in the application that "given all records of employees who left the company, predict who will probably leave the company in a future period." In this case, we divide the records of employees into two groups named "leave" and "stay". And then we can ask our data mining software to classify the employees into separate groups.

3. **Regression:** Regression, used primarily as a form of planning and modelling, is used to identify the likelihood of a certain variable, given the presence of other variables. For example, you could use it to project a certain price, based on other factors like availability, consumer demand, and competition. More specifically, regression's main focus is to help you uncover the exact relationship between two (or more) variables in a given data set.

4. **Clustering:** Clustering is a data mining technique that makes a meaningful or useful cluster of objects which have similar characteristics using the automatic technique. The clustering technique defines the classes and puts objects in each class, while in the classification techniques, objects are assigned into predefined classes. To make the concept clearer, we can take book management in the library as an example. In a library, there is a wide range of books on various topics available. The challenge is how to keep those books in a way that readers can take several books on a particular topic without hassle. By using the clustering technique, we can keep books that have some kinds of similarities in one cluster or one shelf

and label them with a meaningful name. If readers want to grab books on that topic, they would only have to go to that shelf instead of looking for the entire library.

5. **Prediction** The prediction, as its name implied, is a data mining technique that discovers the relationship between independent variables and the relationship between dependent and independent variables. For instance, the prediction analysis technique can be used in the sale to predict profit for the future if we consider the sale is an independent variable, profit could be a dependent variable. Then based on the historical sale and profit data, we can draw a fitted regression curve that is used for profit prediction.

6. **Sequential Patterns**: Sequential patterns analysis is one of the data mining technique that seeks to discover or identify similar patterns, regular events or trends in transaction data over a business period. In sales, with historical transaction data, businesses can identify a set of items that customers buy together at different times in a year. Then businesses can use this information to recommend customers buy it with better deals based on their purchasing frequency in the past.

7. **Decision trees:** A decision tree is one of the most commonly used data mining techniques because its model is easy to understand for users. In this technique, the root of the decision tree is a simple question or condition that has multiple answers. Each answer then leads to a set of questions or conditions that help us determine the data so that we can make the final decision based on it.

Findings and Learnings :

We learned about data mining and its respective significance. We also learned about the open-source tools which can be used to assist us in the practice of data warehousing and data mining. Different techniques are available for data mining depending on the type of data and the amount of data available. Finally we learned about the features of the tools used for data mining. They also vary a lot in the way they approach data mining and the various algorithms they use.

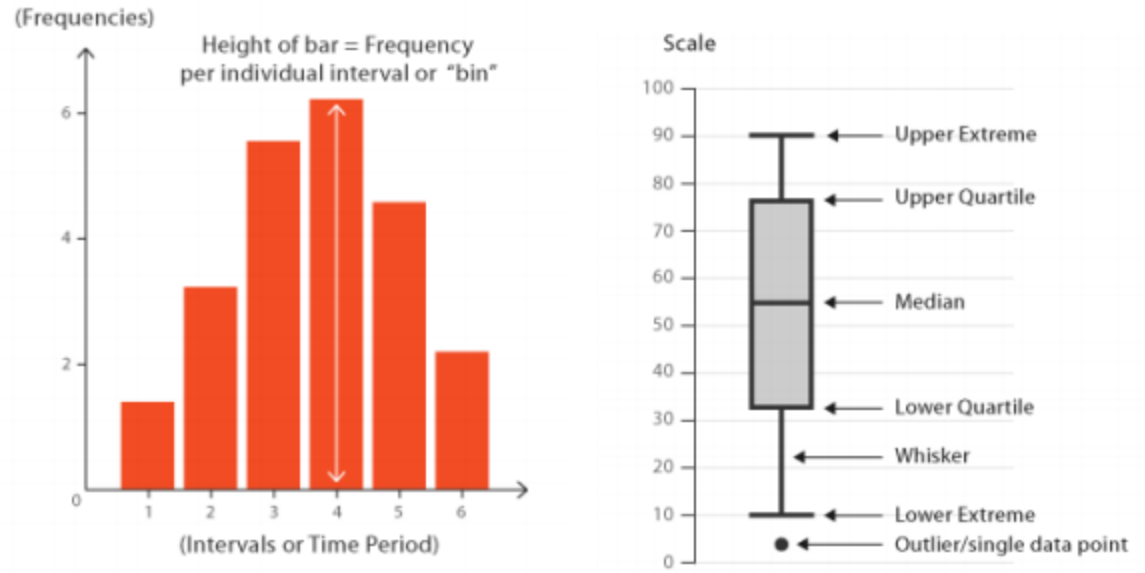
Experiment 3

Aim: Generate Histogram and Boxplot for a sample program in WEKA.

Theory:

A histogram is a plot that lets you discover, and show, the underlying frequency distribution (shape) of a set of continuous data. This allows the inspection of the data for its underlying distribution (e.g., normal distribution), outliers, skewness, etc. A Histogram visualises the distribution of data over a continuous interval or certain time period. Each bar in a histogram represents the tabulated frequency at each interval/bin. Histograms help give an estimate as to where values are concentrated, what the extremes are and whether there are any gaps or unusual values. They are also useful for giving a rough view of the probability distribution.

A Box and Whisker Plot (or Box Plot) is a convenient way of visually displaying the data distribution through their quartiles. The lines extending parallel from the boxes are known as the "whiskers", which are used to indicate variability outside the upper and lower quartiles. Outliers are sometimes plotted as individual dots that are in-line with whiskers. Box Plots can be drawn either vertically or horizontally. Although Box Plots may seem primitive in comparison to a Histogram or Density Plot, they have the advantage of taking up less space, which is useful when comparing distributions between many groups or datasets. Here are the types of observations one can make from viewing a Box Plot:



What the key values are, such as the average, median 25th percentile etc.

- If there are any outliers and what their values are.
- Is the data symmetrical.
- How tightly is the data grouped.

- If the data is skewed and if so, in what direction.

Two of the most commonly used variation of the Box Plot are variable-width Box Plots and notched Box Plots. A Five Number Summary includes:

- Minimum
- First Quartile
- Median (Second Quartile)
- Third Quartile
- Maximum

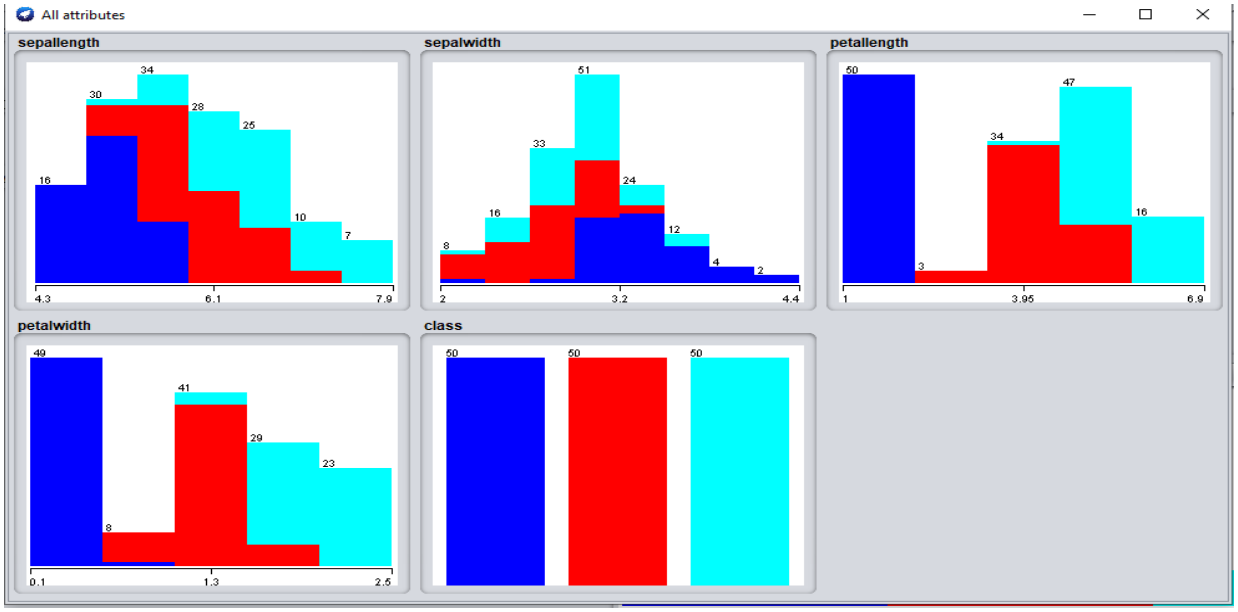
Procedure :

- Plotting histograms
  - Go to weka explorer.
  - Choose dataset in weka-3.8.3/data.
  - Above histogram, click visualize all.
- Plotting Box-Plots
  - Go to weka explorer
  - Choose dataset in weka-3.8.3/data
  - Use CPython scripting and pandas DataFrame boxplot method
  - Plot the boxplot using CPython

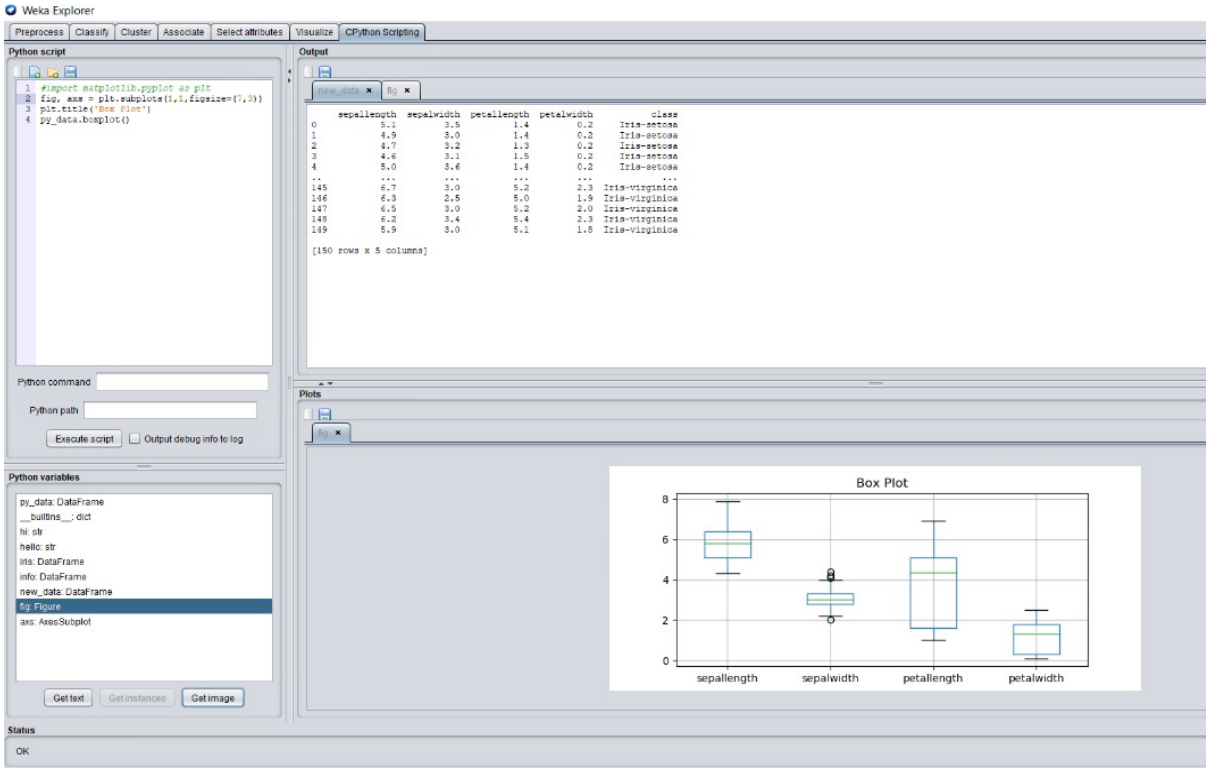
Output :

The following plots have been obtained for the dataset iris.arff

Histogram



Boxplot



Findings and Learnings :

Box plots and histograms are important tools to give an appropriate graphical representation of the raw data and hence are extensively used in data visualization. Weka Software provides a good set of functions to plot histograms and box plots with various combinations of attributes. We have successfully plotted histograms and boxplots in WEKA

Experiment 4

Aim:

A. Identify two data sets in which baseline classification is better then j48. Show the results using WEKA.
B. Calculate the sample mean and SD for any data set(at least 1500 instant and 6 classes) by setting the random number seed from 1 to 10 using WEKA.
C. Show the results for a handout(10%) over 10-cross validation using WEKA.

Part A

Identify two data sets in which baseline classification is better then j48. Show the results using WEKA.

Theory:

Baseline Classification: A baseline prediction algorithm provides a set of predictions that you can evaluate as you would any predictions for your problems, such as classification accuracy or RMSE. The scores from these algorithms provide the required point of comparison when evaluating all other machine learning algorithms on your problem. Once established, you can comment on how much better a given algorithm is as compared to the naive baseline algorithm, providing context on just how good a given method actually is.
The two most commonly used baseline algorithms are:

- Random Prediction Algorithm.
- Zero Rule Algorithm.

J48 Tree Algorithm: C4.5 (J48) is an algorithm used to generate a decision tree developed by Ross Quinlan mentioned earlier. C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier. It became quite popular after ranking #1 in the Top 10 Algorithms in Data Mining pre-eminent paper published by Springer LNCS in 2008.

Output:

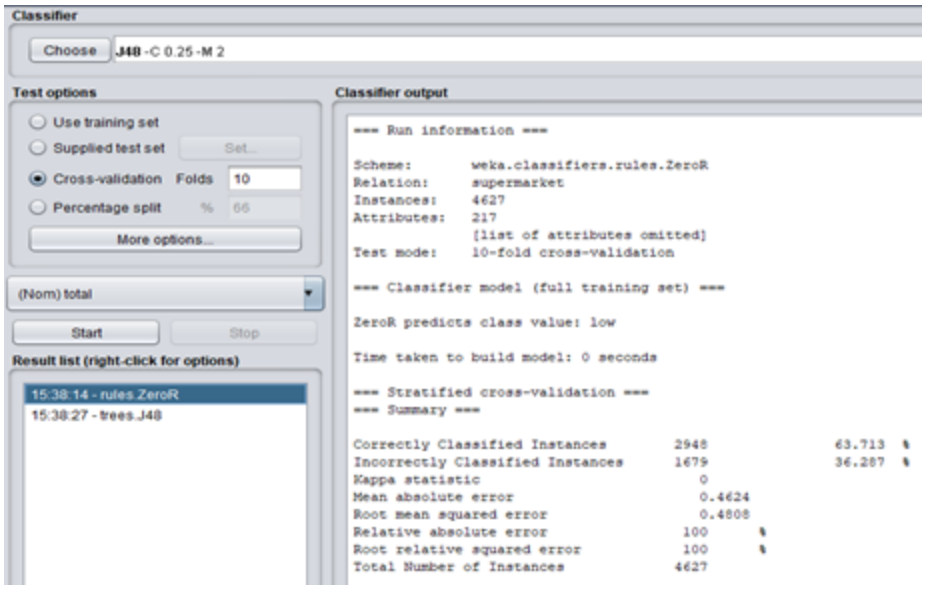
Dataset 1:

Supermarket. arff

1. **Zero R :**

   Accuracy 63.713

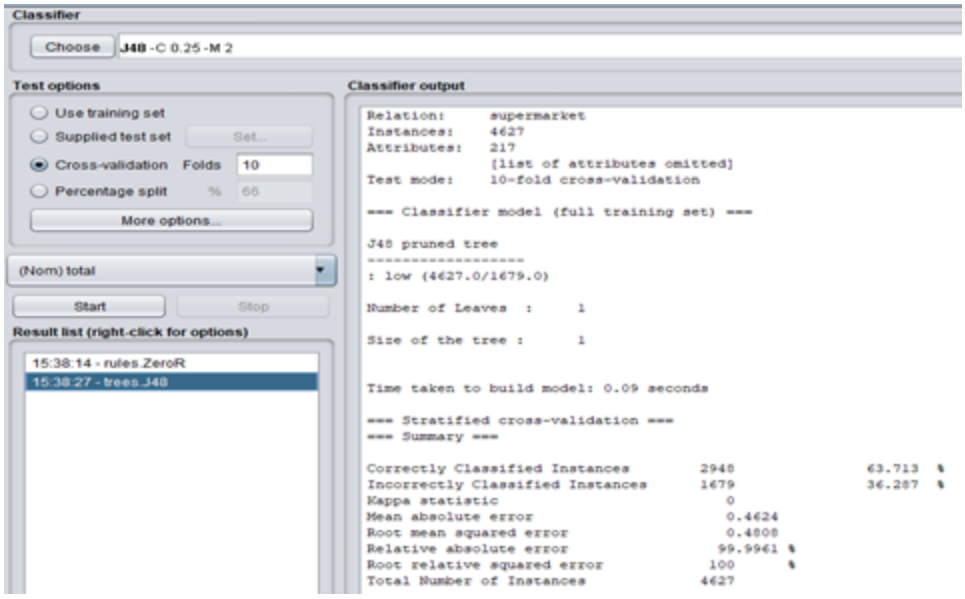
   Time is taken to build 0.01sec



2. **J48 :**

   Accuracy 63.713

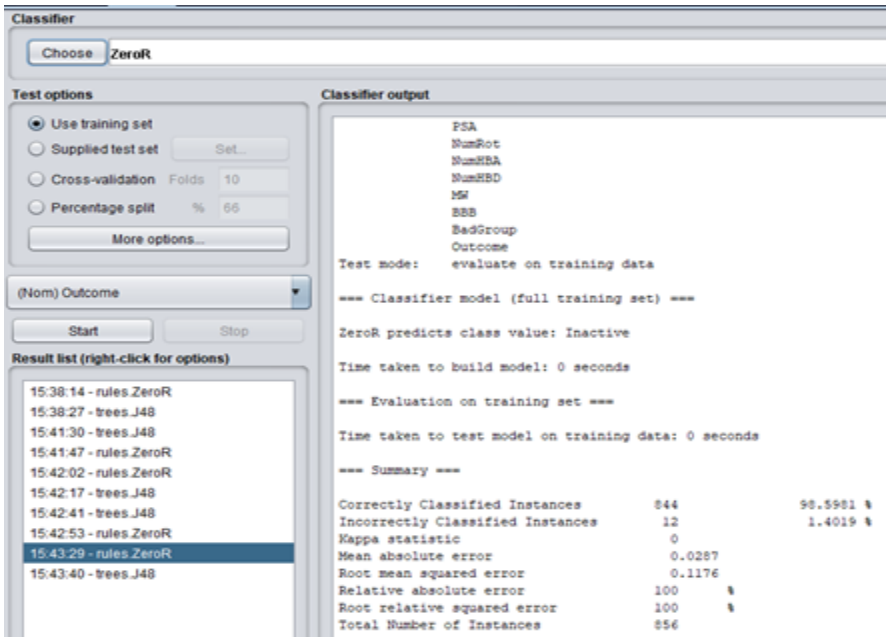
   Time is taken to build 0.09sec



Unbalanced. Arff

1. **Zero R：**Time is taken to build 0.01sec

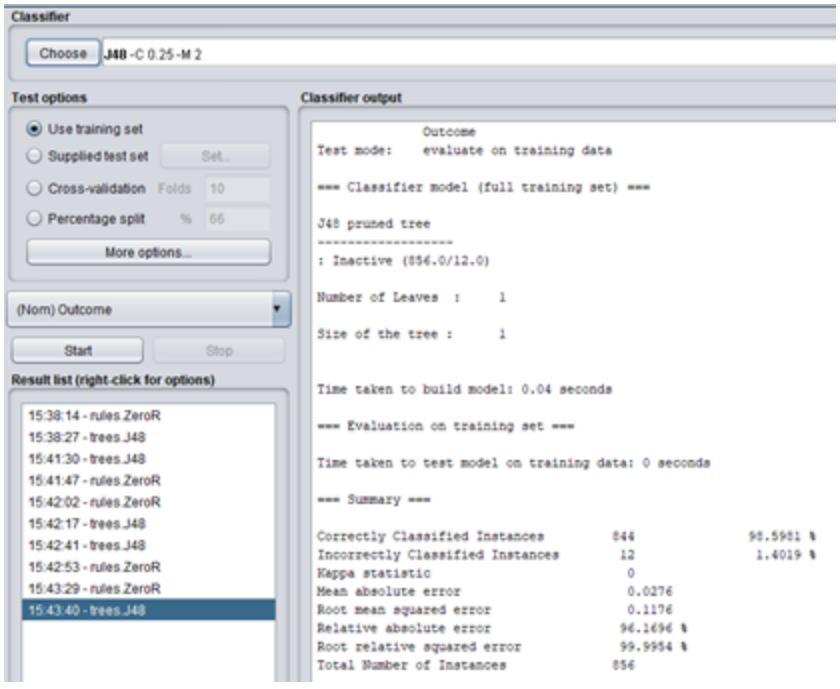
   Accuracy 98.5981



2. **J48：**

   Accuracy 98.5981

   Time is taken to build 0.04sec



Part B

Calculate the sample mean and SD for any data set(at least 1500 instant and 6 classes) by setting the random number seed from 1 to 10 using WEKA

Theory:

Why is randomness important?

It may be clear that reproducibility in machine learning is important, but how do we balance this with the need for randomness? There are both practical benefits for randomness and constraints that force us to use randomness.

Practically speaking, memory and time constraints have also forced us to 'lean' on randomness. Gradient Descent is one of the most popular and widely used algorithms for training machine learning models, however, computing the gradient step based on the entire dataset isn't feasible for large datasets and models. Stochastic Gradient Descent (SGD) only uses one or a mini-batch of randomly picked training samples from the training set to do the update for a parameter in a particular iteration.

While SGD might lead to a noisier error in the gradient estimate, this noise can actually encourage exploration to escape shallow local minima more easily. You can take this one step farther with simulated annealing, an extension of SGD, where the model purposefully take random steps in order to seek a better state.

Here are some important parts of the machine learning workflow where randomness appears:

1. **Data preparation**: in the case of a neural network, the shuffled batches will lead to different loss values across runs. This means your gradient values will be different across runs, and you will probably converge to a different local minima For specific types of data like time-series, audio, or text data plus specific types of models like LSTMs and RNNs, your data's input order can dramatically impact model performance.
2. **Data preprocessing**: over or upsampling data to address class imbalance involves randomly selecting an observation from the minority class with replacement. Upsampling can lead to overfitting because you're showing the model the same example multiple times.
3. **Cross-validation:** Both K-fold and Leave One Out Cross Validation (LOOCV) involve randomly splitting your data in order to evaluate the generalization performance of the model
4. **Weight initialization:** The initial weight values for machine learning models are often set to small, random numbers (usually in the range [-1, 1] or [0, 1]). Deep learning frameworks offer a variety of initialization methods from initializing with zeros to initializing from a normal distribution (see the Keras Initializers documentation as an example plus this excellent resource).

5. **Hidden layers in the network**: Dropout layers will randomly ignore a subset of nodes (each with a probability of being dropped, 1-p) during a particular forward or backwards pass. This leads to differences in layer activations even when the same input is used.
6. **Algorithms themselves**: Some models, such as random forest, are naturally dependent on randomness and others use randomness as a way of exploring the space.

These factors all contribute to variations across runs — making reproducibility very difficult even if you're working with the same model code and training data. Getting control over non-determinism and visibility into your experimentation process is crucial.

Source Code:

```
// Generated with Weka 3.8.3
// This code is public domain and comes with no warranty.
// Timestamp: Tue Mar 02 15:57:51 IST 2021

package weka.classifiers;
import weka.core.Attribute;
import weka.core.Capabilities;
import weka.core.Capabilities.Capability;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.RevisionUtils;
import weka.classifiers.Classifier;
import weka.classifiers.AbstractClassifier;

public class WekaWrapper
  extends AbstractClassifier {
 /**
  * Returns only the toString() method.
  *
  * @return a string describing the classifier
  */
 public String globalInfo() {
   return toString();
 }
 /**
  * Returns the capabilities of this classifier.
  *
  * @return the capabilities
```

```
 */
public Capabilities getCapabilities() {
  weka.core.Capabilities result = new weka.core.Capabilities(this);
  result.enable(weka.core.Capabilities.Capability.NOMINAL_ATTRIBUTES);
  result.enable(weka.core.Capabilities.Capability.NUMERIC_ATTRIBUTES);
  result.enable(weka.core.Capabilities.Capability.DATE_ATTRIBUTES);
  result.enable(weka.core.Capabilities.Capability.MISSING_VALUES);
  result.enable(weka.core.Capabilities.Capability.NOMINAL_CLASS);
  result.enable(weka.core.Capabilities.Capability.MISSING_CLASS_VALUES);

  result.setMinimumNumberInstances(0);
  return result;
}
/**
 * only checks the data against its capabilities.
 *
 * @param i the training data
 */
public void buildClassifier(Instances i) throws Exception {
  // can classifier handle the data?
  getCapabilities().testWithFail(i);
}
/**
 * Classifies the given instance.
 *
 * @param i the instance to classify
 * @return the classification result
 */
public double classifyInstance(Instance i) throws Exception {
  Object[] s = new Object[i.numAttributes()];
  for (int j = 0; j < s.length; j++) {
        if (!i.isMissing(j)) {
        if (i.attribute(j).isNominal())
        s[j] = new String(i.stringValue(j));
        else if (i.attribute(j).isNumeric())
        s[j] = new Double(i.value(j));
        }
  }
  // set class value to missing
```

```
    s[i.classIndex()] = null;
    return WekaClassifier.classify(s);
  }

  /**
   * Returns the revision string.
   *
   * @return     the revision
   */
  public String getRevision() {
    return RevisionUtils.extract("1.0");
  }
  /**
   * Returns only the classnames and what classifier it is based on.
   *
   * @return a short description
   */
  public String toString() {
     return "Auto-generated classifier wrapper, based on weka.classifiers.trees.J48 (generated with
Weka 3.8.3).\n" + this.getClass().getName() + "/WekaClassifier";
  }
  /**
   * Runs the classfier from commandline.
   *
   * @param args the commandline arguments
   */
  public static void main(String args[]) {
    runClassifier(new WekaWrapper(), args);
  }
}
class WekaClassifier {
   public static double classify(Object[] i)
    throws Exception {
     double p = Double.NaN;
    p = WekaClassifier.N5fba94830(i);
    return p;
  }
  static double N5fba94830(Object []i) {
    double p = Double.NaN;
```

```
  if (i[1] == null) {
        p = 1;
  } else if (((Double) i[1]).doubleValue() <= 155.0) {
  p = WekaClassifier.N2688c4c11(i);
  } else if (((Double) i[1]).doubleValue() > 155.0) {
  p = WekaClassifier.N7fa0ebc929(i);
  }
  return p;
}
static double N2688c4c11(Object []i) {
  double p = Double.NaN;
  if (i[16] == null) {
        p = 2;
  } else if (((Double) i[16]).doubleValue() <= 91.4444) {
  p = WekaClassifier.N10924fa22(i);
  } else if (((Double) i[16]).doubleValue() > 91.4444) {
        p = 1;
  }
  return p;
}
static double N10924fa22(Object []i) {
  double p = Double.NaN;
  if (i[10] == null) {
        p = 0;
  } else if (((Double) i[10]).doubleValue() <= 24.6667) {
  p = WekaClassifier.N2ce13ee13(i);
  } else if (((Double) i[10]).doubleValue() > 24.6667) {
  p = WekaClassifier.N383e78e426(i);
  }
  return p;
}
static double N2ce13ee13(Object []i) {
  double p = Double.NaN;
  if (i[18] == null) {
        p = 2;
  } else if (((Double) i[18]).doubleValue() <= -1.89048) {
  p = WekaClassifier.N2774b9724(i);
  } else if (((Double) i[18]).doubleValue() > -1.89048) {
  p = WekaClassifier.N168bc6ca21(i);
```

```
  }
  return p;
}
static double N2774b9724(Object []i) {
  double p = Double.NaN;
  if (i[18] == null) {
        p = 2;
  } else if (((Double) i[18]).doubleValue() <= -2.22266) {
  p = WekaClassifier.N473adf915(i);
  } else if (((Double) i[18]).doubleValue() > -2.22266) {
  p = WekaClassifier.N4557778b6(i);
  }
  return p;
}
static double N473adf915(Object []i) {
  double p = Double.NaN;
  if (i[1] == null) {
        p = 2;
  } else if (((Double) i[1]).doubleValue() <= 146.0) {
        p = 2;
  } else if (((Double) i[1]).doubleValue() > 146.0) {
        p = 3;
  }
  return p;
}
static double N4557778b6(Object []i) {
  double p = Double.NaN;
  if (i[10] == null) {
        p = 2;
  } else if (((Double) i[10]).doubleValue() <= 2.55556) {
  p = WekaClassifier.N60e45ff67(i);
  } else if (((Double) i[10]).doubleValue() > 2.55556) {
  p = WekaClassifier.N62543c2013(i);
  }
  return p;
}
static double N60e45ff67(Object []i) {
  double p = Double.NaN;
  if (i[18] == null) {
```

```
        p = 2;
  } else if (((Double) i[18]).doubleValue() <= -2.09121) {
  p = WekaClassifier.N62d9d5b18(i);
  } else if (((Double) i[18]).doubleValue() > -2.09121) {
        p = 4;
  }
  return p;
}
static double N62d9d5b18(Object []i) {
  double p = Double.NaN;
  if (i[1] == null) {
        p = 2;
  } else if (((Double) i[1]).doubleValue() <= 129.0) {
        p = 2;
  } else if (((Double) i[1]).doubleValue() > 129.0) {
  p = WekaClassifier.N730e482c9(i);
  }
  return p;
}
static double N730e482c9(Object []i) {
  double p = Double.NaN;
  if (i[0] == null) {
        p = 2;
  } else if (((Double) i[0]).doubleValue() <= 128.0) {
  p = WekaClassifier.Nd1b82e910(i);
  } else if (((Double) i[0]).doubleValue() > 128.0) {
  p = WekaClassifier.N4fc2675311(i);
  }
  return p;
}
static double Nd1b82e910(Object []i) {
  double p = Double.NaN;
  if (i[10] == null) {
        p = 2;
  } else if (((Double) i[10]).doubleValue() <= 0.666667) {
        p = 2;
  } else if (((Double) i[10]).doubleValue() > 0.666667) {
        p = 4;
  }
```

```
  return p;
}
static double N4fc2675311(Object []i) {
  double p = Double.NaN;
  if (i[5] == null) {
        p = 4;
  } else if (((Double) i[5]).doubleValue() <= 0.333333) {
        p = 4;
  } else if (((Double) i[5]).doubleValue() > 0.333333) {
  p = WekaClassifier.N631a474712(i);
  }
  return p;
}
static double N631a474712(Object []i) {
  double p = Double.NaN;
  if (i[0] == null) {
        p = 4;
  } else if (((Double) i[0]).doubleValue() <= 216.0) {
        p = 4;
  } else if (((Double) i[0]).doubleValue() > 216.0) {
        p = 2;
  }
  return p;
}
static double N62543c2013(Object []i) {
  double p = Double.NaN;
  if (i[1] == null) {
        p = 4;
  } else if (((Double) i[1]).doubleValue() <= 121.0) {
  p = WekaClassifier.N3309e64914(i);
  } else if (((Double) i[1]).doubleValue() > 121.0) {
  p = WekaClassifier.N1d4a278217(i);
  }
  return p;
}
static double N3309e64914(Object []i) {
  double p = Double.NaN;
  if (i[15] == null) {
        p = 0;
```

```
  } else if (((Double) i[15]).doubleValue() <= -15.4444) {
        p = 0;
  } else if (((Double) i[15]).doubleValue() > -15.4444) {
  p = WekaClassifier.N6fadf78415(i);
  }
  return p;
}
static double N6fadf78415(Object []i) {
  double p = Double.NaN;
  if (i[5] == null) {
        p = 4;
  } else if (((Double) i[5]).doubleValue() <= 2.94444) {
        p = 4;
  } else if (((Double) i[5]).doubleValue() > 2.94444) {
  p = WekaClassifier.N5ebcb54916(i);
  }
  return p;
}
static double N5ebcb54916(Object []i) {
  double p = Double.NaN;
  if (i[0] == null) {
        p = 3;
  } else if (((Double) i[0]).doubleValue() <= 134.0) {
        p = 3;
  } else if (((Double) i[0]).doubleValue() > 134.0) {
        p = 4;
  }
  return p;
}
static double N1d4a278217(Object []i) {
  double p = Double.NaN;
  if (i[10] == null) {
        p = 4;
  } else if (((Double) i[10]).doubleValue() <= 7.88889) {
  p = WekaClassifier.N618da1cc18(i);
  } else if (((Double) i[10]).doubleValue() > 7.88889) {
  p = WekaClassifier.N611c6bae19(i);
  }
  return p;
```

```
}
static double N618da1cc18(Object []i) {
  double p = Double.NaN;
  if (i[0] == null) {
        p = 0;
  } else if (((Double) i[0]).doubleValue() <= 43.0) {
    p = 0;
  } else if (((Double) i[0]).doubleValue() > 43.0) {
        p = 4;
  }
  return p;
}
static double N611c6bae19(Object []i) {
  double p = Double.NaN;
  if (i[17] == null) {
        p = 3;
  } else if (((Double) i[17]).doubleValue() <= 0.492526) {
        p = 3;
  } else if (((Double) i[17]).doubleValue() > 0.492526) {
  p = WekaClassifier.N5a319c3920(i);
  }
  return p;
}
static double N5a319c3920(Object []i) {
  double p = Double.NaN;
  if (i[0] == null) {
        p = 2;
  } else if (((Double) i[0]).doubleValue() <= 82.0) {
        p = 2;
  } else if (((Double) i[0]).doubleValue() > 82.0) {
        p = 3;
  }
  return p;
}
static double N168bc6ca21(Object []i) {
  double p = Double.NaN;
  if (i[15] == null) {
        p = 0;
  } else if (((Double) i[15]).doubleValue() <= -4.77778) {
```

```
  p = WekaClassifier.N1a04861622(i);
  } else if (((Double) i[15]).doubleValue() > -4.77778) {
  p = WekaClassifier.N4ec1588624(i);
  }
  return p;
}
static double N1a04861622(Object []i) {
  double p = Double.NaN;
  if (i[5] == null) {
        p = 0;
  } else if (((Double) i[5]).doubleValue() <= 2.77778) {
        p = 0;
  } else if (((Double) i[5]).doubleValue() > 2.77778) {
  p = WekaClassifier.N1847dfe323(i);
  }
  return p;
}
static double N1847dfe323(Object []i) {
  double p = Double.NaN;
  if (i[1] == null) {
        p = 0;
  } else if (((Double) i[1]).doubleValue() <= 115.0) {
        p = 0;
  } else if (((Double) i[1]).doubleValue() > 115.0) {
        p = 2;
  }
  return p;
}
static double N4ec1588624(Object []i) {
  double p = Double.NaN;
  if (i[7] == null) {
        p = 4;
  } else if (((Double) i[7]).doubleValue() <= 0.833336) {
  p = WekaClassifier.N6347a0225(i);
  } else if (((Double) i[7]).doubleValue() > 0.833336) {
        p = 6;
  }
  return p;
}
```

```
static double N6347a0225(Object []i) {
  double p = Double.NaN;
  if (i[0] == null) {
        p = 2;
  } else if (((Double) i[0]).doubleValue() <= 115.0) {
        p = 2;
  } else if (((Double) i[0]).doubleValue() > 115.0) {
        p = 4;
  }
  return p;
}
static double N383e78e426(Object []i) {
  double p = Double.NaN;
  if (i[18] == null) {
        p = 2;
  } else if (((Double) i[18]).doubleValue() <= -2.17742) {
  p = WekaClassifier.N1ea000c727(i);
  } else if (((Double) i[18]).doubleValue() > -2.17742) {
  p = WekaClassifier.N5ae16efd28(i);
  }
  return p;
}
static double N1ea000c727(Object []i) {
  double p = Double.NaN;
  if (i[5] == null) {
        p = 4;
  } else if (((Double) i[5]).doubleValue() <= 5.0) {
        p = 4;
  } else if (((Double) i[5]).doubleValue() > 5.0) {
        p = 2;
  }
  return p;
}
static double N5ae16efd28(Object []i) {
  double p = Double.NaN;
  if (i[12] == null) {
        p = 0;
  } else if (((Double) i[12]).doubleValue() <= 24.4444) {
        p = 0;
```

```
  } else if (((Double) i[12]).doubleValue() > 24.4444) {
        p = 3;
  }
  return p;
}
static double N7fa0ebc929(Object []i) {
  double p = Double.NaN;
  if (i[15] == null) {
        p = 5;
  } else if (((Double) i[15]).doubleValue() <= -2.0) {
  p = WekaClassifier.N3492558130(i);
  } else if (((Double) i[15]).doubleValue() > -2.0) {
        p = 6;
  }
  return p;
}
static double N3492558130(Object []i) {
  double p = Double.NaN;
  if (i[17] == null) {
        p = 5;
  } else if (((Double) i[17]).doubleValue() <= 0.385555) {
  p = WekaClassifier.N2a2acdec31(i);
  } else if (((Double) i[17]).doubleValue() > 0.385555) {
        p = 3;
  }
  return p;
}
static double N2a2acdec31(Object []i) {
  double p = Double.NaN;
  if (i[1] == null) {
        p = 3;
  } else if (((Double) i[1]).doubleValue() <= 159.0) {
  p = WekaClassifier.N7bedd8d232(i);
  } else if (((Double) i[1]).doubleValue() > 159.0) {
        p = 5;
  }
  return p;
}
static double N7bedd8d232(Object []i) {
```

```
    double p = Double.NaN;
    if (i[0] == null) {
            p = 3;
    } else if (((Double) i[0]).doubleValue() <= 208.0) {
            p = 3;
    } else if (((Double) i[0]).doubleValue() > 208.0) {
            p = 5;
    }
    return p;
  }
}
```

Output:

Seeds	1	2	3	4	5	6	7	8	9	10
accuracy	95.098	93.725	96.078	94.902	94.705	92.549	93.137	94.117	93.137	95.490

Mean: 94.29412
Standard Deviation: 1.148717

Part C

Show the results for a handout(10%) over 10-cross validation using WEKA

Theory:

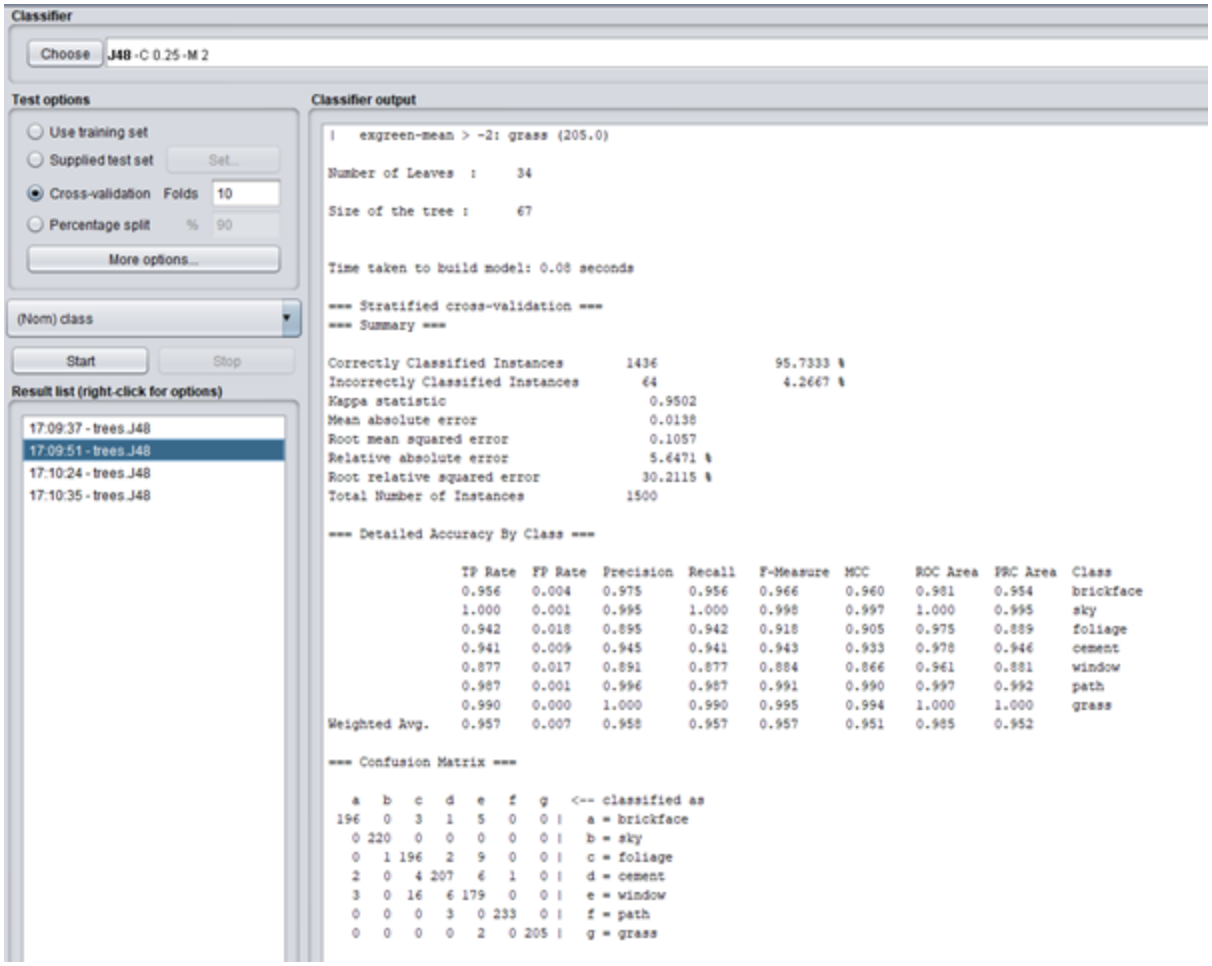
Handout

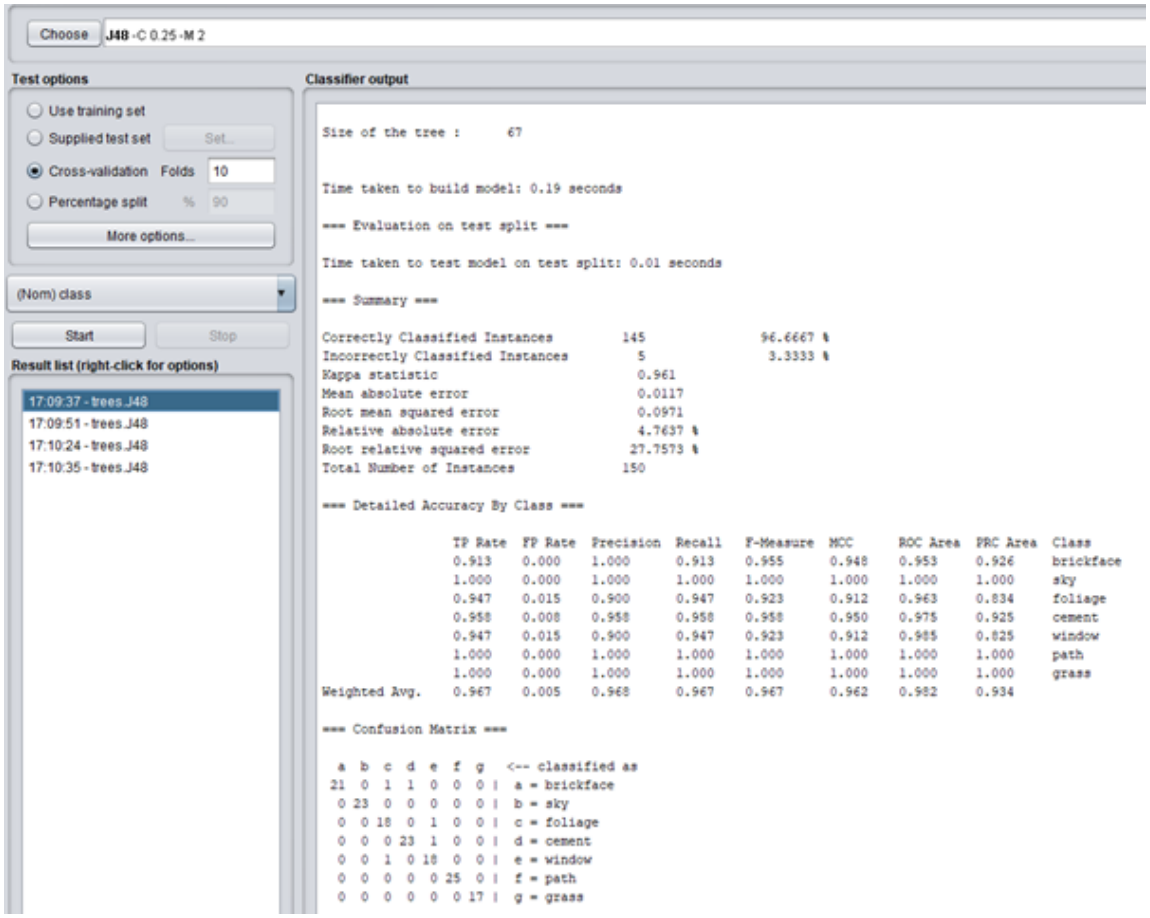
The handout is when you split up your dataset into a 'train' and 'test' set. The training set is what the model is trained on, and the test set is used to see how well that model performs on unseen data. A common split when using the handout method is using 80% of data for training and the remaining 20% of the data for testing.

Cross-validation

Cross-validation or 'k-fold cross-validation' is when the dataset is randomly split up into 'k' groups. One of the groups is used as the test set and the rest are used as the training set. The model is trained on the training set and scored on the test set. Then the process is repeated until each unique group as been used as the test set.

Output:





Findings and Learnings :

- Many times we find that baselines match or outperform complex models, especially when the complex model has been chosen without looking at where the baseline fails. In addition, complex models are usually harder to deploy, which means measuring their lift over a simple baseline is a necessary precursor to the engineering efforts needed to deploy them.
- By carefully setting the random seed across your pipeline you can achieve reproducibility. The “seed” is a starting point for the sequence and the guarantee is that if you start from the same seed you will get the same sequence of numbers.
- Cross-validation is usually the preferred method because it gives your model the opportunity to train on multiple train-test splits. This gives you a better indication of how well your model will perform on unseen data. Handout, on the other hand, is dependent on just one train-test split. That makes the handout method score dependent on how the data is split into train and test sets. The handout method is good to use when you have a very large dataset, you’re on a time crunch, or you are starting to build an initial model in your data science project.

Experiment 5

Aim:

A. To perform Decision Tree learning in WEKA.
B. To Implement Decision Tree classifier in python

Theory:

Decision tree induction is the learning of decision trees from class-labeled training tuples. A decision tree is a flowchart-like tree structure, where each internal node non leaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node.Given a tuple, X, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple. Decision trees can easily be converted to classification rules.

ID3, C4.5, and CART adopt a greedy (i.e., non backtracking) approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner. Most algorithms for decision tree induction also follow a top-down approach, which starts with a training set of tuples and their associated class labels. The training set is recursively partitioned into smaller subsets as the tree is being built.

An attribute selection measure is a heuristic for selecting the splitting criterion that "best" separates a given data partition, D, of class-labeled training tuples into individual classes. If we were to split D into smaller partitions according to the outcomes of the splitting criterion, ideally each partition would be pure (i.e., all the tuples that fall into a given partition would belong to the same class). Conceptually, the "best" splitting criterion is the one that most closely results in such a scenario. Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split

Strengths and Weakness of Decision Tree approach

Strengths:

- Decision trees are able to generate understandable rules.
- Decision trees perform classification without requiring much computation.
- Decision trees are able to handle both continuous and categorical variables.
- Decision trees provide a clear indication of which fields are most important for prediction or classification.

Weaknesses:

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many class and relatively small numbers of training examples.
- Decision trees can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.

Algorithm :

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, *D*.

Input:

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.

Method:

```
(1)  create a node N;
(2)  if tuples in D are all of the same class, C, then
(3)      return N as a leaf node labeled with the class C;
(4)  if attribute_list is empty then
(5)      return N as a leaf node labeled with the majority class in D; // majority voting
(6)  apply Attribute_selection_method(D, attribute_list) to find the "best" splitting_criterion;
(7)  label node N with splitting_criterion;
(8)  if splitting_attribute is discrete-valued and
         multiway splits allowed then // not restricted to binary trees
(9)      attribute_list ← attribute_list − splitting_attribute; // remove splitting_attribute
(10) for each outcome j of splitting_criterion
     // partition the tuples and grow subtrees for each partition
(11)     let Dj be the set of data tuples in D satisfying outcome j; // a partition
(12)     if Dj is empty then
(13)         attach a leaf labeled with the majority class in D to node N;
(14)     else attach the node returned by Generate_decision_tree(Dj, attribute_list) to node N;
     endfor
(15) return N;
```

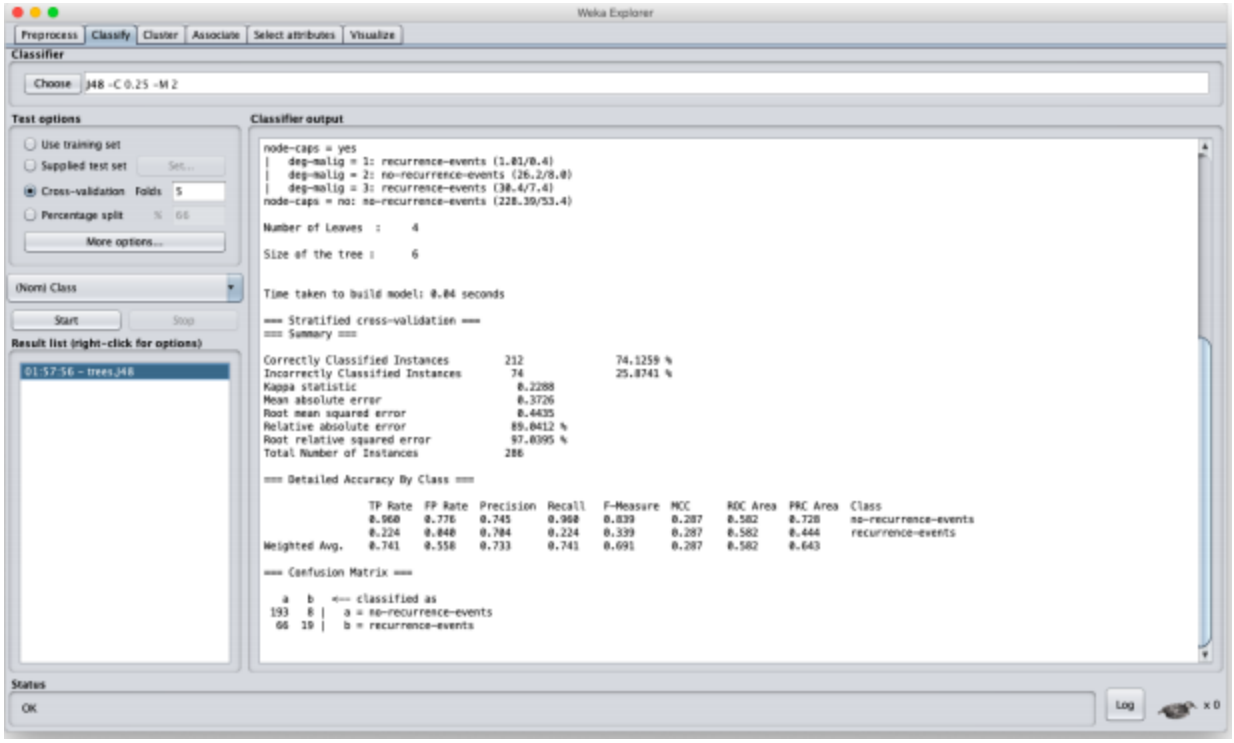
Part A

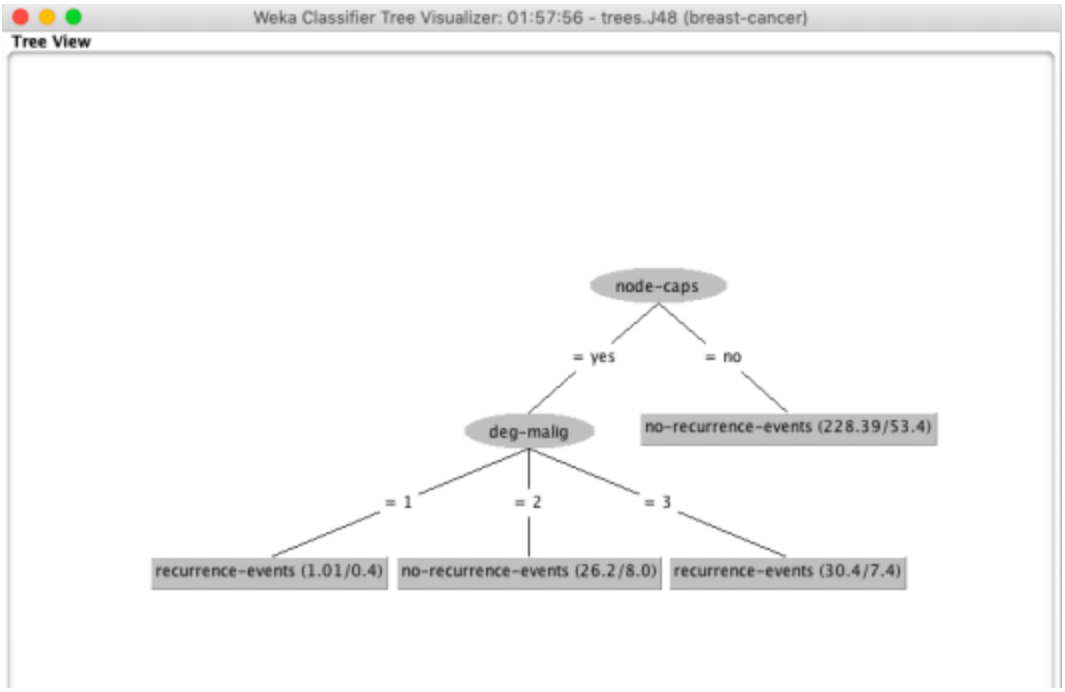
To perform Decision Tree learning in WEKA

Procedure :

1. Go to Weka Explorer.
2. Choose dataset in weka/data
3. Go to classify tab
4. Choose a classifier in trees/ID3 or any other.
5. Click start.
6. On the result, right click and visualize.

Output:





Findings and Learnings:

- Decision trees are the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.
- Weka software provides a good set of classification algorithms to be trained and tested on our dataset and makes it very simple to build a complex classifier using algorithms like decision trees.

Part B

To perform Decision Tree learning in Python

ID3 uses Information gain:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

$$Gain(A) = Info(D) - info_A(D)$$

C4.5 uses gain ratio:

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2 \frac{|D_j|}{|D|}$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

CART uses the GINI index:

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2$$

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

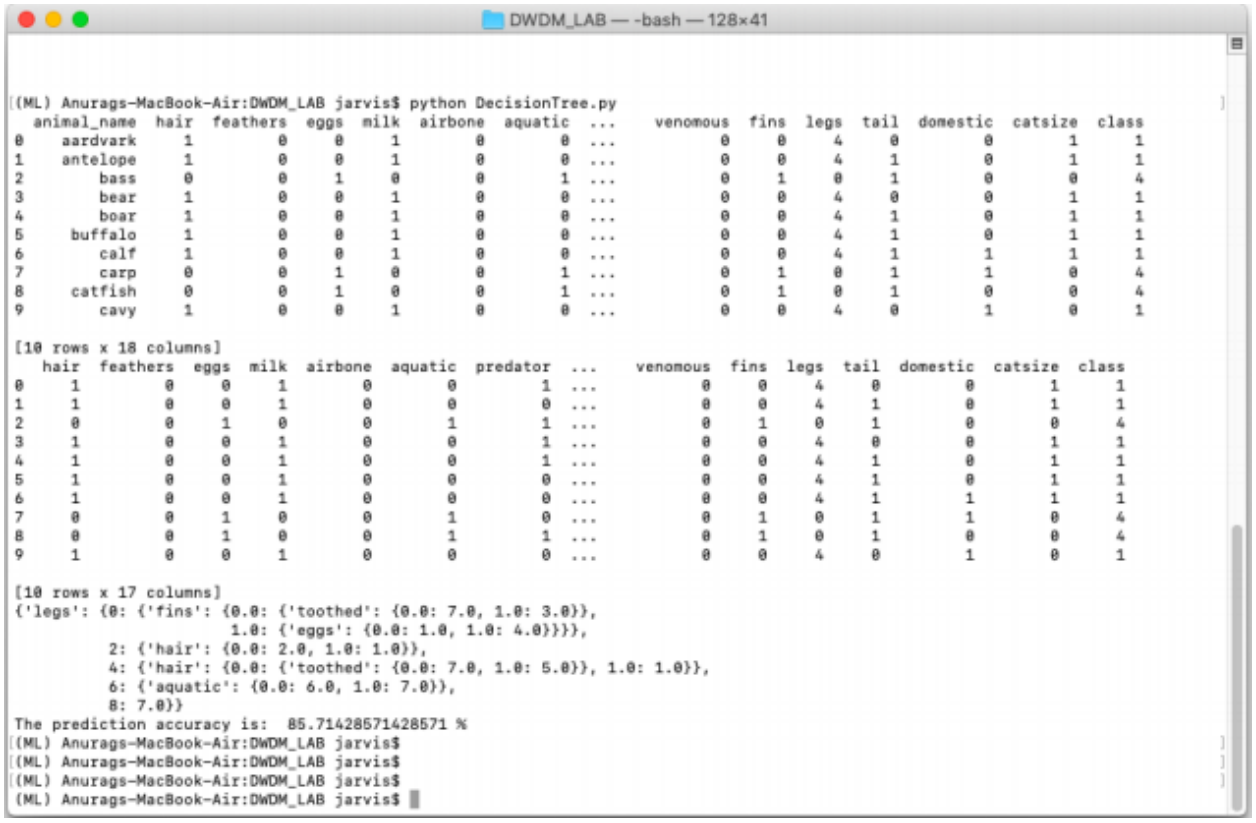
Source Code:

```
import pandas as pd
import numpy as np
from pprint import pprint

def entropy(target_col):
    elements, counts = np.unique(target_col, return_counts=True)
    result = np.sum([(-counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts)) for i in
range(len(elements))])
    return result

def information_gain(data, split_attribute_name, target_name="class"):
    total_entropy = entropy(data[target_name])
```

```
    vals, counts = np.unique(data[split_attribute_name], return_counts=True)
    weighted_entropy = np.sum(
        [(counts[i] / np.sum(counts)) * entropy(data.where(data[split_attribute_name] ==
vals[i]).dropna()[target_name])
         for i in range(len(vals))])
    information_gain_ = total_entropy - weighted_entropy
    return information_gain_

def id3(data, original_data, features, target_attribute_name="class", parent_node_class=None):
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]
    elif len(data) == 0:
        return np.unique(original_data[target_attribute_name])[
            np.argmax(np.unique(original_data[target_attribute_name], return_counts=True)[1])]
    elif len(features) == 0:
        return parent_node_class
    else:
        parent_node_class = np.unique(data[target_attribute_name])[
            np.argmax(np.unique(data[target_attribute_name], return_counts=True)[1])]
        item_values = [information_gain(data, feature, target_attribute_name) for feature in
                       features]  # Return the information gain values for the features in the dataset
        best_feature_index = np.argmax(item_values)
        best_feature = features[best_feature_index]
        tree = {best_feature: {}}
        features = [i for i in features if i != best_feature]
        for value in np.unique(data[best_feature]):
            value = value
            sub_data = data.where(data[best_feature] == value).dropna()
            subtree = id3(sub_data, dataset, features, target_attribute_name, parent_node_class)
            tree[best_feature][value] = subtree
        return tree

def predict(query, tree, default=1):
    for key in list(query.keys()):
        if key in list(tree.keys()):
            try:
                result = tree[key][query[key]]
            except:
```

```
            return default

        result = tree[key][query[key]]
        if isinstance(result, dict):
            return predict(query, result)
        else:
            return result

def train_test_split(dataset):
    training_data = dataset.iloc[:80].reset_index(drop=True)
    testing_data = dataset.iloc[80:].reset_index(drop=True)
    return training_data, testing_data

def test(data, tree):
    queries = data.iloc[:, :-1].to_dict(orient="records")
    predicted = pd.DataFrame(columns=["predicted"])
    for i in range(len(data)):
        predicted.loc[i, "predicted"] = predict(queries[i], tree, 1.0)
    print('The prediction accuracy is: ', (np.sum(predicted["predicted"] == data["class"]) /
len(data)) * 100, '%')

if __name__ == '__main__':
    # loading the dataset
    dataset = pd.read_csv('zoo.csv', names=['animal_name', 'hair', 'feathers', 'eggs', 'milk',
                                            'airbone', 'aquatic', 'predator', 'toothed', 'backbone',
                                            'breathes', 'venomous', 'fins', 'legs', 'tail', 'domestic', 'catsize',
                                            'class'])

    print(dataset.head(10))

    dataset = dataset.drop('animal_name', axis=1)
    print(dataset.head(10))
    training_data = train_test_split(dataset)[0]
    testing_data = train_test_split(dataset)[1]

    tree = id3(training_data, training_data, training_data.columns[:-1])
    pprint(tree)
    test(testing_data, tree)
```

Output:



Findings and Learnings:

- We have Implemented Decision tree through the ID3 algorithm in python 3.
- We have learned the nuances of the Decision tree learning.
- We have learnt about the applications, strengths and weaknesses of Decision tree Learning

Experiment 6

Aim:

A. To perform a simple k-mean clustering algorithm on any data set(at least 1500 instant and 6 classes).
B. To Implement k mean clustering algorithm in python.

Theory: K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition 'n' observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. The problem is computationally difficult (NP-hard); however, efficient heuristic algorithms converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both k-means and Gaussian mixture modelling. They both use cluster centres to model the data; however, k-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

The algorithm has a loose relationship to the k-nearest neighbour classifier, a popular machine learning technique for classification that is often confused with k-means due to the name. Applying the 1-nearest neighbour classifier to the cluster centres obtained by k-means classifies new data into the existing clusters. To process the learning data, the K-means algorithm in data mining starts with the first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids It halts creating and optimizing clusters when either:

- The centroids have stabilized—there is no change in their values because the clustering has been successful.
- The defined number of iterations has been achieved.

Algorithm:

A. Initialize k means with random values
B. For a given number of iterations:
    a. Iterate through items:
        i. Find the mean closest to the item
        ii. Assign item to mean
        iii. Update mean

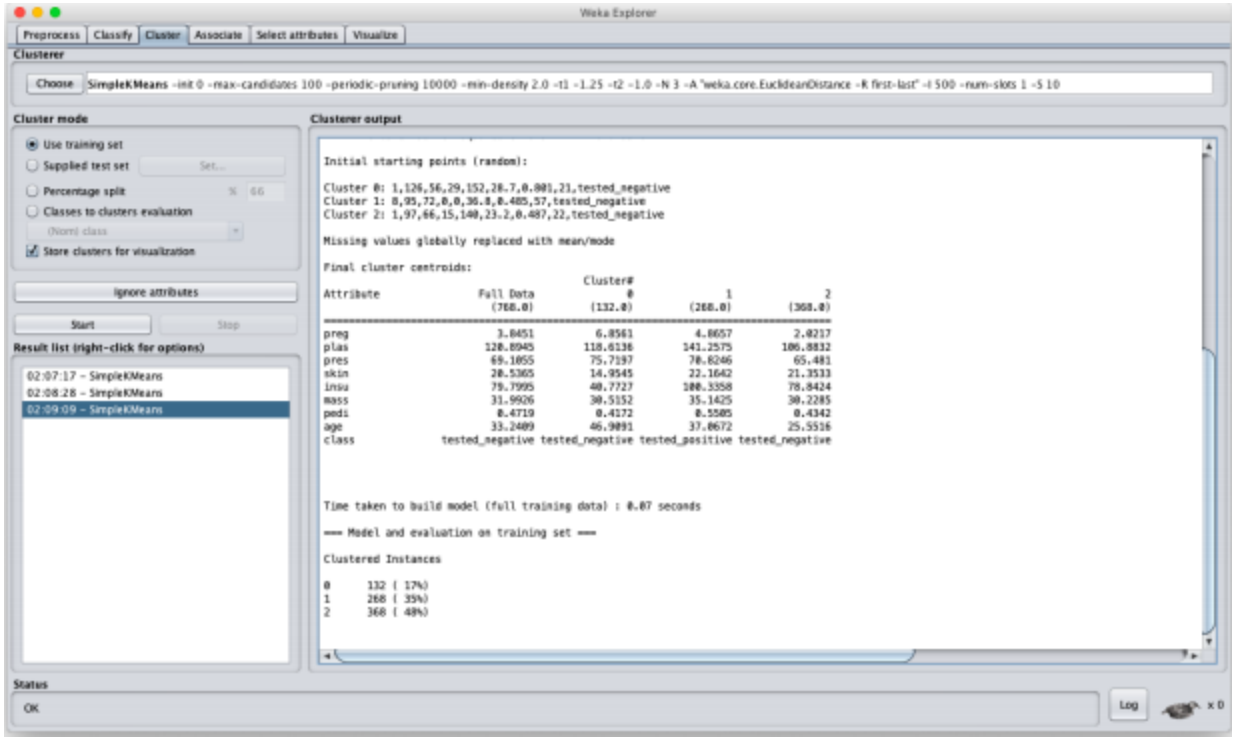
Part A

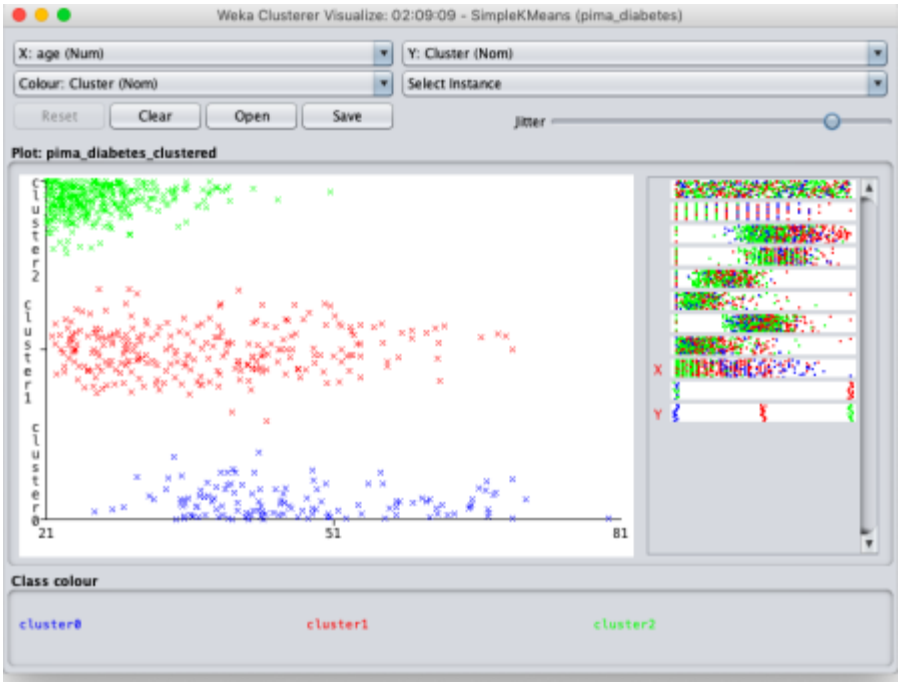
To perform a simple k-mean clustering algorithm on any data set.

Procedure:

1. Go to Weka Explorer.
2. Choose dataset in weka/data
3. Go to cluster tab
4. Choose cluster algorithm, in this case k-means
5. Click start.
6. Visualize the results

Output:





Findings and Learnings :

1. K-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem.
2. Weka software makes the implementation and visualization of K-means algorithms very simple and efficient.
3. It makes K-means easily available to non-programmer users as well.

Part B

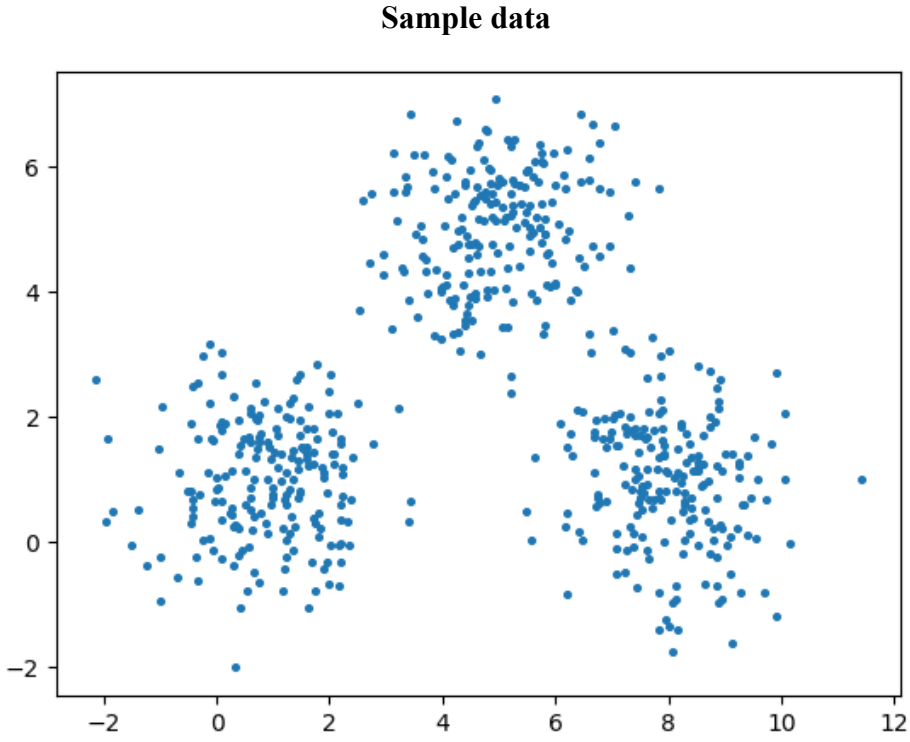
To Implement k mean clustering algorithm in python.

Source Code:

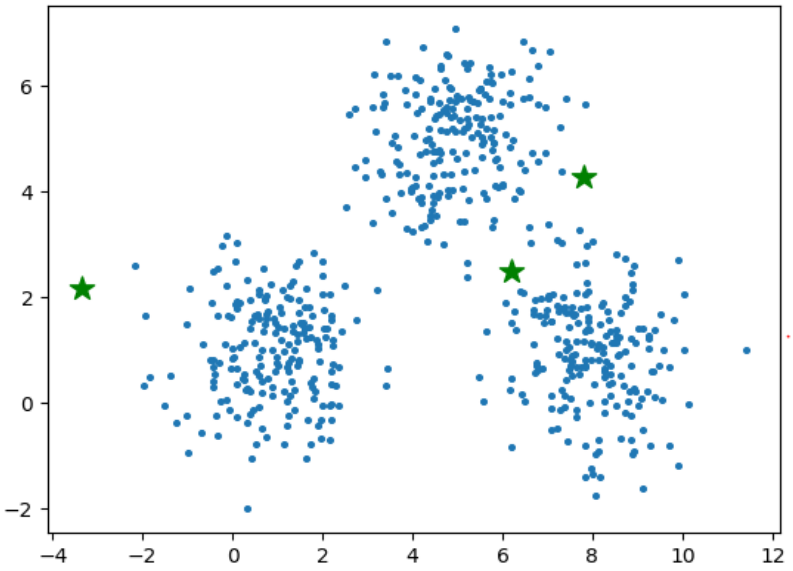
```
from copy import deepcopy
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

center_1 = np.array([1,1])
center_2 = np.array([5,5])
center_3 = np.array([8,1])
# Generate random data and center it to the three centers
data_1 = np.random.randn(200, 2) + center_1
data_2 = np.random.randn(200,2) + center_2
data_3 = np.random.randn(200,2) + center_3
data = np.concatenate((data_1, data_2, data_3), axis = 0)
plt.scatter(data[:,0], data[:,1], s=7)
plt.show()
# Number of clusters
k = 3
# Number of training data
n = data.shape[0]
# Number of features in the data
c = data.shape[1]
# Generate random centers, here we use sigma and mean to ensure it represent the whole data
mean = np.mean(data, axis = 0)
std = np.std(data, axis = 0)
centers = np.random.randn(k,c)*std + mean
# Plot the data and the centers generated as random
plt.scatter(data[:,0], data[:,1], s=7)
plt.scatter(centers[:,0], centers[:,1], marker='*', c='g', s=150)
plt.show()
centers_old = np.zeros(centers.shape) # to store old centers
centers_new = deepcopy(centers) # Store new centers
print(data.shape)
clusters = np.zeros(n)
distances = np.zeros((n, k))
error = np.linalg.norm(centers_new - centers_old)
```

```
# When, after an update, the estimate of that center stays the same, exit loop
while error != 0:
    for i in range(k):
        distances[:, i] = np.linalg.norm(data - centers[i], axis=1)
     # Assign all training data to closest center
    clusters = np.argmin(distances, axis=1)
    centers_old = deepcopy(centers_new)
     # Calculate mean for every cluster and update the center
    for i in range(k):
        centers_new[i] = np.mean(data[clusters == i], axis=0)
    error = np.linalg.norm(centers_new - centers_old)
print(centers_new)
# Plot the data and the centers generated as random
plt.scatter(data[:,0], data[:,1], s=7)
plt.scatter(centers_new[:,0], centers_new[:,1], marker='*', c='g',s=150)
plt.show()
```

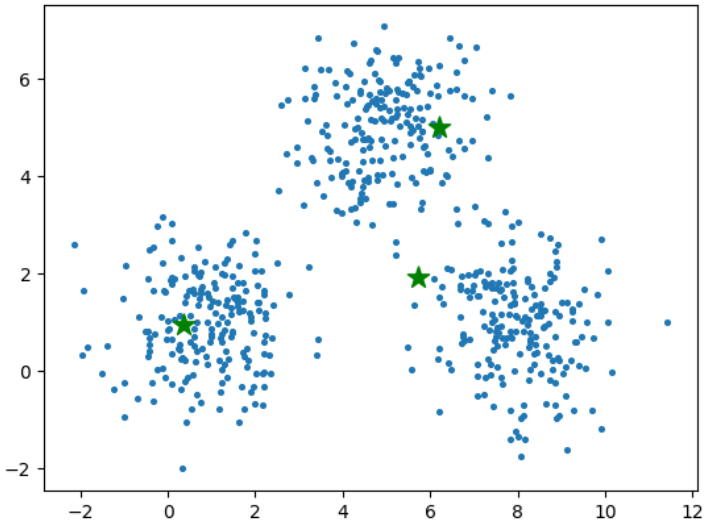
Output:



Random centroid initialization



Final cluster assignment



Findings and Learnings :

1. K-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem.
2. We have successfully implemented k-means clustering in Python

Experiment 7

Aim: To perform association rule mining on any dataset using any 2 algorithms

Theory:

Association rule learning is a rule-based machine learning method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using some measures of interestingness. Association rule mining finds interesting associations and relationships among large sets of data items. This rule shows how frequently an itemset occurs in a transaction.

Rule Evaluation Metrics

- Support. This says how popular an itemset is, as measured by the proportion of transactions in which an itemset appears. The support of an itemset X, supp(X) is the proportion of transactions in the database in which item X appears. It signifies the popularity of an itemset.

$$\text{supp(X)}=\frac{Number\ of\ transaction\ in\ which\ X\ appears}{Total\ number\ of\ transactions}$$

  If the sales of a particular product (item) above a certain proportion have a meaningful effect on profits, that proportion can be considered as the support threshold. Furthermore, we can identify itemsets that have support values beyond this threshold as significant itemsets.

- Confidence. This says how likely item Y is purchased when item X is purchased, expressed as {X -> Y}. This is measured by the proportion of transactions with item X, in which item Y also appears. Confidence of a rule is defined as follows:

$$conf\ (X \rightarrow Y) = \frac{supp(\ X \cup Y)}{supp(X)}$$

  It signifies the likelihood of item Y being purchased when item X is purchased. It can also be interpreted as the conditional probability P(Y|X), i.e, the probability of finding the itemset Yin transactions given the transaction already contains X.

- Lift. This says how likely item Y is purchased when item X is purchased, while controlling for how popular item Y is. A lift value greater than 1 means that item Y is likely to be bought if item X is bought, while a value less than 1 means that item Y is unlikely to be bought if item X is bought. The lift of a rule is defined as:

$$lift\ (X \rightarrow Y) = \frac{supp(\ X \cup Y)}{supp(X) * supp(Y)}$$

  This signifies the likelihood of the itemset Y being purchased when item X is purchased while taking into account the popularity of Y. If the value of lift is greater than 1, it means

that the itemset Y is likely to be bought with itemset X, while a value less than 1 implies that itemset Y is unlikely to be bought if the itemset X is bought

A typical example is Market Based Analysis. Market Based Analysis is one of the key techniques used by large relations to show associations between items.It allows retailers to identify relationships between the items that people buy together frequently. The "supermarket" dataset (supermarket.arff) is a real world transaction data set from a small NZ supermarket. Each instance represents a customer transaction – products purchased and the departments involved. The data contains 4,500 instances and 220 attributes. Each attribute is binary and either has a value (t for true) or no value ("?" for missing). We can easily understand how difficult it would be to detect the association between such a large number of attributes

We would be using two algorithms for this purpose

1. **Apriori:** The Apriori algorithm is used for mining frequent itemsets and devising association rules from a transactional database. The parameters "support" and "confidence" are used. Support refers to items' frequency of occurrence; confidence is a conditional probability. Items in a transaction form an item set. The algorithm begins by identifying frequent, individual items (items with a frequency greater than or equal to the given support) in the database and continues to extend them to larger, frequent itemsets.

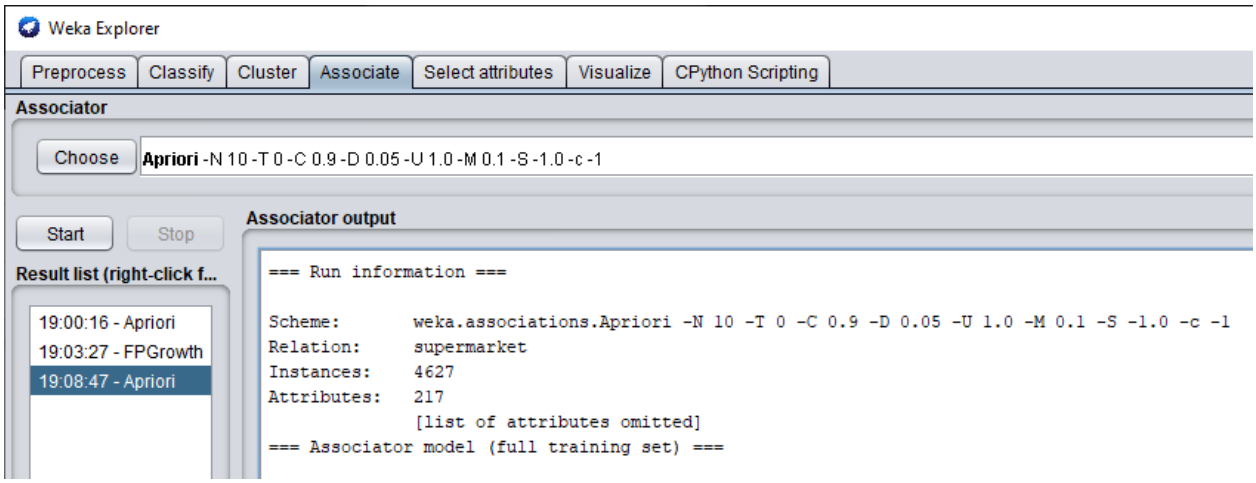
2. **FP Growth**: Fp Growth Algorithm (Frequent pattern growth) is an improvement of apriori algorithm. FP growth algorithm used for finding frequent itemset in a transaction database without candidate generation.FP growth represents frequent items in frequent pattern trees or FP-tree.Advantages of FP growth algorithm:-
    a. Faster than apriori algorithm
    b. No candidate generation
    c. Only two passes over dataset

Procedure:

- Go to Weka Explorer.
- Choose dataset in weka/data (supermarket.arff)
- Go to Associate tab
- Choose an algorithm
- Click start.
- Navigate to Associate tab and under associator choose Apriori and then hit start

Output:

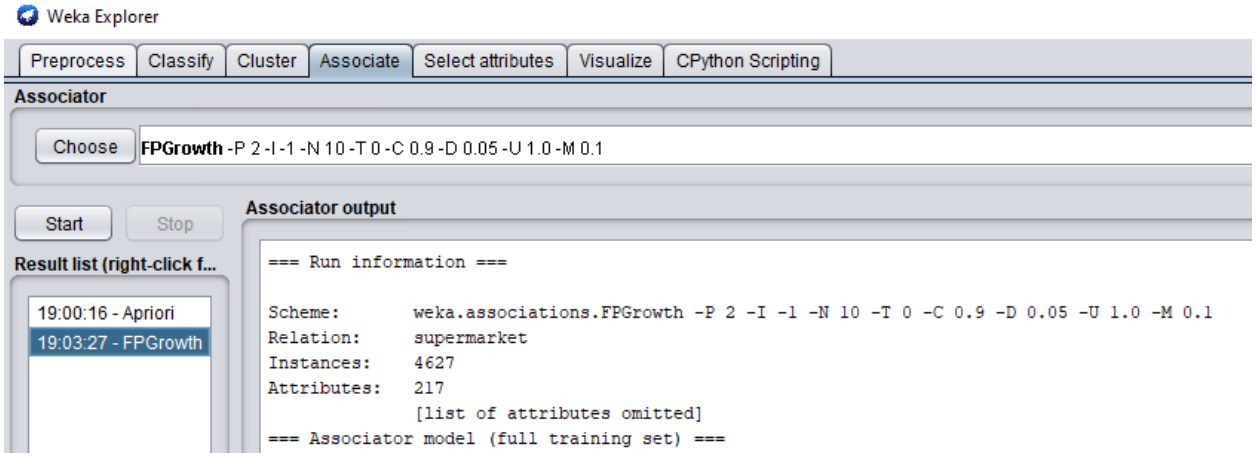
Using Apriori Algorithm



Zoomed Output View

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

Generated sets of large itemsets:

Size of set of large itemsets L(1): 44

Size of set of large itemsets L(2): 380

Size of set of large itemsets L(3): 910

Size of set of large itemsets L(4): 633

Size of set of large itemsets L(5): 105

Size of set of large itemsets L(6): 1

Best rules found:

 1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723    <conf:(0.92)> lift:(1.27) lev:(0.03) [155] conv:(3.35)
 2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696    <conf:(0.92)> lift:(1.27) lev:(0.03) [149] conv:(3.28)
 3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705    <conf:(0.92)> lift:(1.27) lev:(0.03) [150] conv:(3.27)
 4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746    <conf:(0.92)> lift:(1.27) lev:(0.03) [159] conv:(3.26)
 5. party snack foods=t fruit=t total=high 854 ==> bread and cake=t 779    <conf:(0.91)> lift:(1.27) lev:(0.04) [164] conv:(3.15)
 6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725    <conf:(0.91)> lift:(1.26) lev:(0.03) [151] conv:(3.06)
 7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701    <conf:(0.91)> lift:(1.26) lev:(0.03) [145] conv:(3.01)
 8. biscuits=t fruit=t total=high 954 ==> bread and cake=t 866    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(3)
 9. frozen foods=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757    <conf:(0.91)> lift:(1.26) lev:(0.03) [156] conv:(3)
10. frozen foods=t fruit=t total=high 969 ==> bread and cake=t 877    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(2.92)
```

Using FP Growth Algorithm



Zoomed Output View

```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1
Relation:     supermarket
Instances:    4627
Attributes:   217
              [list of attributes omitted]
=== Associator model (full training set) ===

FPGrowth found 16 rules (displaying top 10)

 1. [fruit=t, frozen foods=t, biscuits=t, total=high]: 788 ==> [bread and cake=t]: 723   <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.35)
 2. [fruit=t, baking needs=t, biscuits=t, total=high]: 760 ==> [bread and cake=t]: 696   <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.28)
 3. [fruit=t, baking needs=t, frozen foods=t, total=high]: 770 ==> [bread and cake=t]: 705   <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.27)
 4. [fruit=t, vegetables=t, biscuits=t, total=high]: 815 ==> [bread and cake=t]: 746   <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.26)
 5. [fruit=t, party snack foods=t, total=high]: 854 ==> [bread and cake=t]: 779   <conf:(0.91)> lift:(1.27) lev:(0.04) conv:(3.15)
 6. [vegetables=t, frozen foods=t, biscuits=t, total=high]: 797 ==> [bread and cake=t]: 725   <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3.06)
 7. [vegetables=t, baking needs=t, biscuits=t, total=high]: 772 ==> [bread and cake=t]: 701   <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3.01)
 8. [fruit=t, biscuits=t, total=high]: 954 ==> [bread and cake=t]: 866   <conf:(0.91)> lift:(1.26) lev:(0.04) conv:(3)
 9. [fruit=t, vegetables=t, frozen foods=t, total=high]: 834 ==> [bread and cake=t]: 757   <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3)
10. [fruit=t, frozen foods=t, total=high]: 969 ==> [bread and cake=t]: 877   <conf:(0.91)> lift:(1.26) lev:(0.04) conv:(2.92)
```

Findings and Learnings :

- We learnt about Association rule mining and the associated terms such as confidence , support and lift.
- We learnt to use Association rule mining in WEKA.
- We learnt the use of Apriori and FP Growth in WEKA.

Experiment 8

Aim: To implement the Apriori algorithm in C++.

Theory: Association rule mining, at a basic level, involves the use of machine learning models to analyze data for patterns, or co-occurrence, in a database. It identifies frequent if-then associations, which are called association rules. An association rule has two parts: an antecedent (if) and a consequent (then). An antecedent is an item found within the data. A consequent is an item found in combination with the antecedent.

Apriori algorithm is given by R. Agrawal and R. Srikant in 1994 for finding frequent itemsets in a dataset for boolean association rule. Name of the algorithm is Apriori because it uses prior knowledge of frequent itemset properties. We apply an iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets.

The key concept of Apriori algorithm is its anti-monotonicity of support measure. Apriori assumes that All subsets of a frequent itemset must be frequent(Apriori property). If an itemset is infrequent, all its supersets will be infrequent. With the help of these association rule, it determines how strongly or how weakly two objects are connected. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset associations efficiently. It is the iterative process for finding the frequent itemsets from the large dataset.

The Apriori algorithm is used for mining frequent itemsets and devising association rules from a transactional database. The parameters “support” and “confidence” are used. Support refers to items’ frequency of occurrence; confidence is a conditional probability. Items in a transaction form an item set. The algorithm begins by identifying frequent, individual items (items with a frequency greater than or equal to the given support) in the database and continues to extend them to larger, frequent itemsets.

Algorithm:

Below are the steps for the apriori algorithm:

1. Determine the support of itemsets in the transactional database, and select the minimum support and confidence.
2. Take all supports in the transaction with higher support value than the minimum or selected support value.
3. Find all the rules of these subsets that have higher confidence value than the threshold or minimum confidence.
4. Sort the rules as the decreasing order of lift values.

$$
\begin{aligned}
&\text{Apriori}(T, \epsilon) \\
&\quad L_1 \leftarrow \{\text{large } 1-\text{itemsets}\} \\
&\quad k \leftarrow 2 \\
&\quad \textbf{while } L_{k-1} \neq \emptyset \\
&\qquad C_k \leftarrow \{c = a \cup \{b\} \mid a \in L_{k-1} \wedge b \notin a, \{s \subseteq c \mid |s| = k-1\} \subseteq L_{k-1}\} \\
&\qquad \textbf{for transactions } t \in T \\
&\qquad\quad D_t \leftarrow \{c \in C_k \mid c \subseteq t\} \\
&\qquad\quad \textbf{for candidates } c \in D_t \\
&\qquad\qquad count[c] \leftarrow count[c] + 1 \\
&\qquad L_k \leftarrow \{c \in C_k \mid count[c] \geq \epsilon\} \\
&\qquad k \leftarrow k + 1 \\
&\quad \textbf{return } \bigcup_k L_k
\end{aligned}
$$

Applications:

1. Market Basket Analysis:This is the most typical example of association mining. Data is collected using barcode scanners in most supermarkets.Knowing which groups are inclined towards which set of items gives these shops the freedom to adjust the store layout and the store catalogue to place the optimally concerning one another.

2. Medical Diagnosis: Association rules in medical diagnosis can be useful for assisting physicians for curing patients. Using relational association rule mining, we can identify the probability of the occurrence of an illness concerning various factors and symptoms.

3. Census Data: This data can be used to plan efficient public services. This application of association rule mining and data mining has immense potential in supporting sound public policy and bringing forth an efficient functioning of a democratic society.

Limitations:

- Apriori Algorithm can be slow.
- The main limitation is time required to hold a vast number of candidate sets with much frequent itemsets, low minimum support or large itemsets i.e. it is not an efficient approach for large number of datasets.
- Furthermore, to detect frequent patterns in size 100 i.e. v1, v2… v100, it has to generate 2^100 candidate itemsets that yield on costly and wasting of time of candidate generation. So, it will check for many sets from candidate itemsets, also it will scan the database many times repeatedly for finding candidate itemsets.
- Apriori will be very low and inefficient when memory capacity is limited with a large number of transactions.

Source Code:

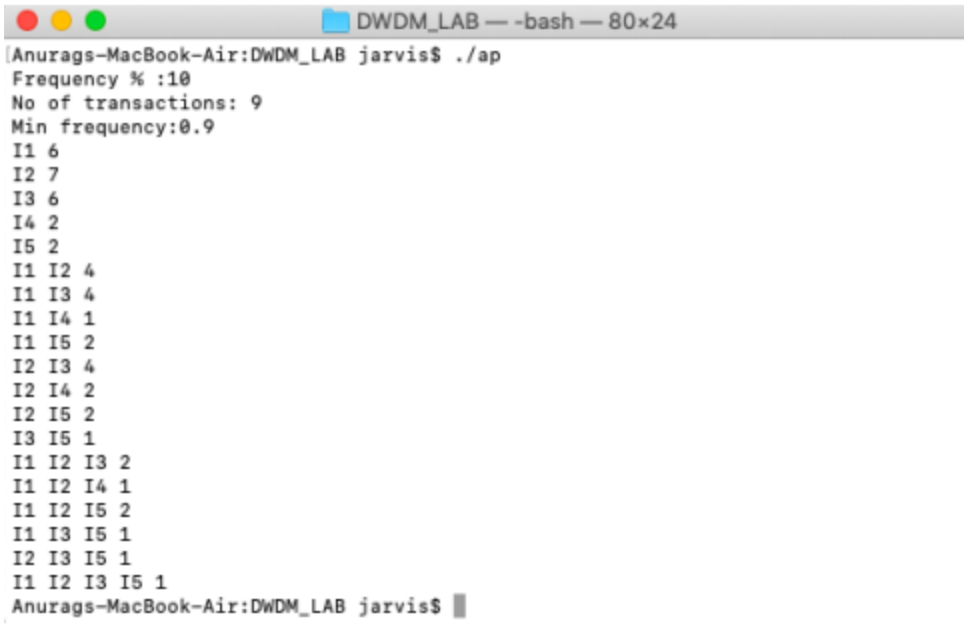
```
#include <bits/stdc++.h>
#include <map>
using namespace std;
ifstream fin;
double minfree;
vector<set<string> > datatable;
set<string> products;
map<string, int> freq;
vector<string> wordsof(string str)
{ vector<string> tmpset;
 string tmp = "";
 int i = 0;
 while (str[i])
 {
 if (isalnum(str[i]))
 tmp += str[i];
 else
 {
 if (tmp.size() > 0)
 tmpset.push_back(tmp);
 tmp = "";
 }
 i++;
 }
 if (tmp.size() > 0)
 tmpset.push_back(tmp);
 return tmpset;
}
string combine(vector<string> &arr, int miss)
{
 string str;
 for (int i = 0; i < arr.size(); i++)
 if (i != miss)
 str += arr[i] + " ";
 str = str.substr(0, str.size() - 1);
 return str;
}
set<string> cloneit(set<string> &arr)
```

```
{
 set<string> dup;
 for (set<string>::iterator it = arr.begin(); it != arr.end();it++)
 dup.insert(*it);
 return dup;
}
set<string> apriori_gen(set<string> &sets, int k)
{
 set<string> set2;
 for (set<string>::iterator it1 = sets.begin(); it1 != sets.end(); it1++)
 {
 set<string>::iterator it2 = it1;
 it2++;
 for (; it2 != sets.end(); it2++)
 {
 vector<string> v1 = wordsof(*it1);
 vector<string> v2 = wordsof(*it2);
 bool alleq = true;
 for (int i = 0; i < k - 1 && alleq; i++)
 if (v1[i] != v2[i])
 alleq = false;
 if (!alleq)
 continue;
 v1.push_back(v2[k - 1]);
 if (v1[v1.size() - 1] < v1[v1.size() - 2])
 swap(v1[v1.size() - 1], v1[v1.size() - 2]);
 for (int i = 0; i < v1.size() && alleq; i++)
 {
 string tmp = combine(v1, i);
 if (sets.find(tmp) == sets.end())
 alleq = false;
 }
 if (alleq)
 set2.insert(combine(v1, -1)); }
 }
 return set2;
}
int main()
{          fin.open("apriori.in");
```

```
        cout << "Frequency % :";
        cin >> minfre;
        string str;
        while (!fin.eof())
        {
getline(fin, str);
vector<string> arr = wordsof(str); //taking data from file ,
set<string> tmpset;
for (int i = 0; i < arr.size(); i++)
tmpset.insert(arr[i]);
datatable.push_back(tmpset);
for (set<string>::iterator it = tmpset.begin(); it != tmpset.end(); it++)
{       products.insert(*it);
        freq[*it]++;
        }
}
fin.close();
cout << "No of transactions: " << datatable.size() << endl;
minfre = minfre * datatable.size() / 100;
cout << "Min frequency:" << minfre << endl;
queue<set<string>::iterator> q;
for (set<string>::iterator it = products.begin(); it != products.end(); it++)
        if (freq[*it] < minfre)
                q.push(it);
while (q.size() > 0)
{
products.erase(*q.front());
q.pop();
}
for (set<string>::iterator it = products.begin(); it != products.end(); it++)
cout << *it << " " << freq[*it] << endl;
int i = 2;
set<string> prev = cloneit(products);
while (i)
{
set<string> cur = apriori_gen(prev, i - 1);
if (cur.size() < 1)
break;
for (set<string>::iterator it = cur.begin(); it != cur.end(); it++)
```

```
{
vector<string> arr = wordsof(*it);
int tot = 0;
for (int j = 0; j < datatable.size(); j++)
{
bool pres = true;
for (int k = 0; k < arr.size() && pres; k++)
if (datatable[j].find(arr[k]) == datatable[j].end())
pres = false;
if (pres)
tot++;
}
if (tot >= minfre)
freq[*it] += tot;
else
q.push(it);
}
while (q.size() > 0)
{          cur.erase(*q.front());
           q.pop();
}
for (set<string>::iterator it = cur.begin(); it != cur.end(); it++)
          cout << *it << " " << freq[*it] << endl;
prev = cloneit(cur);
i++;
}
}
```

Output:



Findings and Learnings :

1. We learned about association rule mining and familiarize ourselves with the test.arff dataset.
2. We implemented and tested Apriori in C++.

Experiment 9

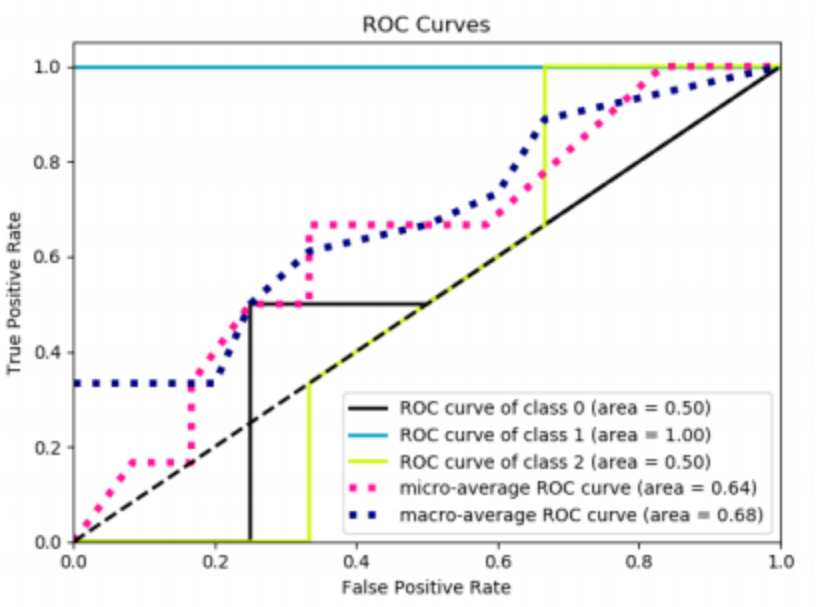
Aim: To perform ROC analysis, Forecast analysis and Survival analysis on any sample dataset

Theory:

Receiver Operator Curve(ROC) Analysis

A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection in machine learning. The false-positive rate is also known as the fall-out or probability of false alarm and can be calculated as (1 − specificity). It can also be thought of as a plot of the power as a function of the Type I Error of the decision rule (when the performance is calculated from just a sample of the population, it can be thought of as estimators of these quantities). The ROC curve is thus the sensitivity as a function of fall-out. In general, if the probability distributions for both detection and false alarm are known, the ROC curve can be generated by plotting the cumulative distribution function (area under the probability distribution from negative infinity to the discrimination threshold) of the detection probability in the y-axis versus the cumulative distribution function of the false-alarm probability on the x-axis.

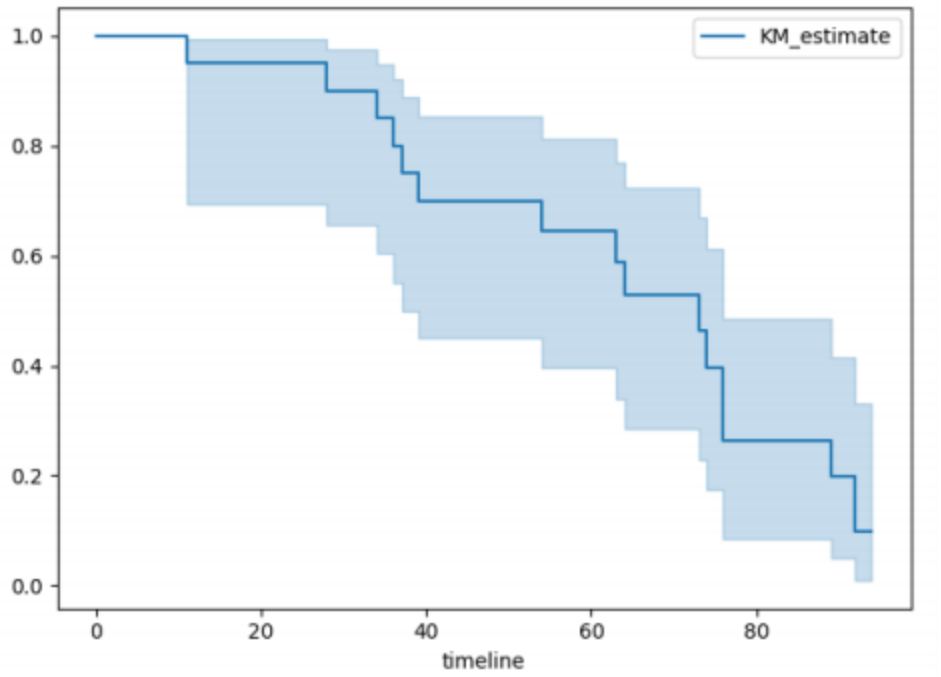
ROC analysis provides tools to select possibly optimal models and to discard suboptimal ones independently from the cost context or the class distribution. ROC analysis is related in a direct and natural way to cost/benefit analysis of diagnostic decision making.



Survival Analysis

Survival analysis is a branch of statistics for analyzing the expected duration of time until one or more events happen, such as death in biological organisms and failure in mechanical systems. This topic is called reliability theory or reliability analysis in engineering, duration analysis or duration modelling in economics, and event history analysis in sociology. Survival analysis attempts to answer questions such as: what is the proportion of a population which will survive past a certain time? Of those that survive, at what rate will they die or fail? Can multiple causes of death or failure be taken into account? How do particular circumstances or characteristics increase or decrease the probability of survival? To answer such questions, it is necessary to define "lifetime". In the case of biological survival, death is unambiguous, but for mechanical reliability, failure may not be well-defined, for there may well be mechanical systems in which failure is partial, a matter of degree, or not otherwise localized in time. Even in biological problems, some events (for example, heart attack or other organ failure) may have the same ambiguity. The theory outlined below assumes well-defined events at specific times; other cases may be better treated by models which explicitly account for ambiguous events.

More generally, survival analysis involves the modelling of time to event data; in this context, death or failure is considered an "event" in the survival analysis literature – traditionally only a single event occurs for each subject, after which the organism or mechanism is dead or broken. Recurring event or repeated event models relax that assumption. The study of recurring events is relevant in systems reliability, and in many areas of social sciences and medical research.



Forecast Analysis

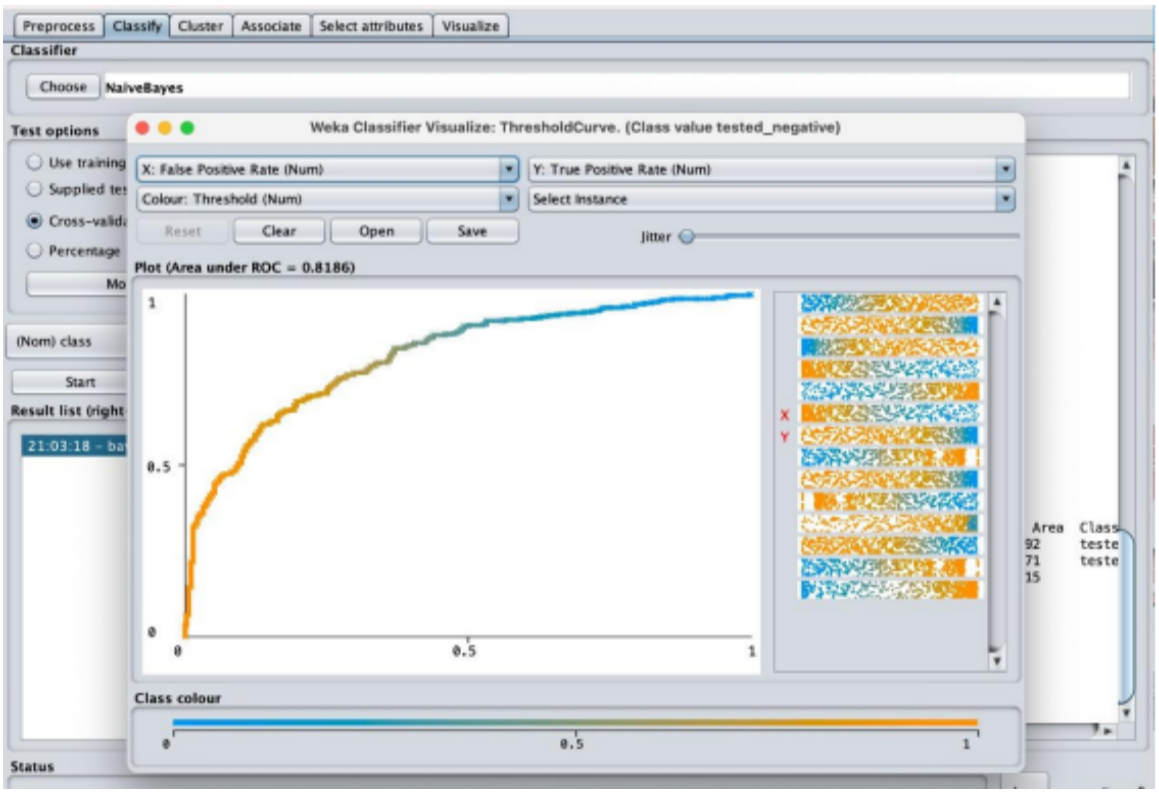
Forecasting is the process of making predictions of the future based on past and present data and most commonly by analysis of trends. A commonplace example might be estimation of some variable of interest at some specified future date. Prediction is a similar, but more general term. Both might refer to formal statistical methods employing time series, cross-sectional or longitudinal data, or alternatively to less formal judgmental methods. Usage can differ between areas of application: for example, in hydrology the terms "forecast" and "forecasting" are sometimes reserved for estimates of values at certain specific future times, while the term "prediction" is used for more general estimates, such as the number of times floods will occur over a long period.

Risk and uncertainty are central to forecasting and prediction; it is generally considered good practice to indicate the degree of uncertainty attaching to forecasts. In any case, the data must be up to date in order for the forecast to be as accurate as possible. In some cases the data used to predict the variable of interest is itself forecasted.

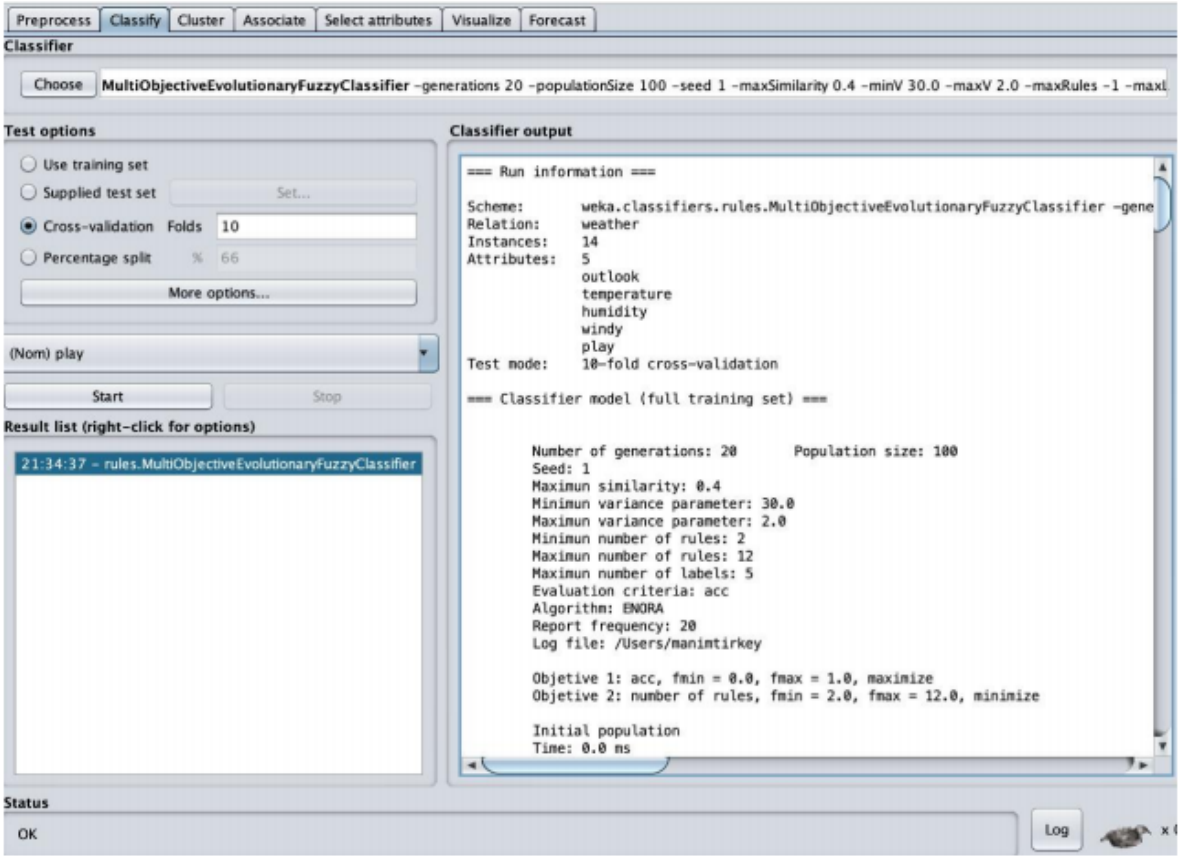
Output:

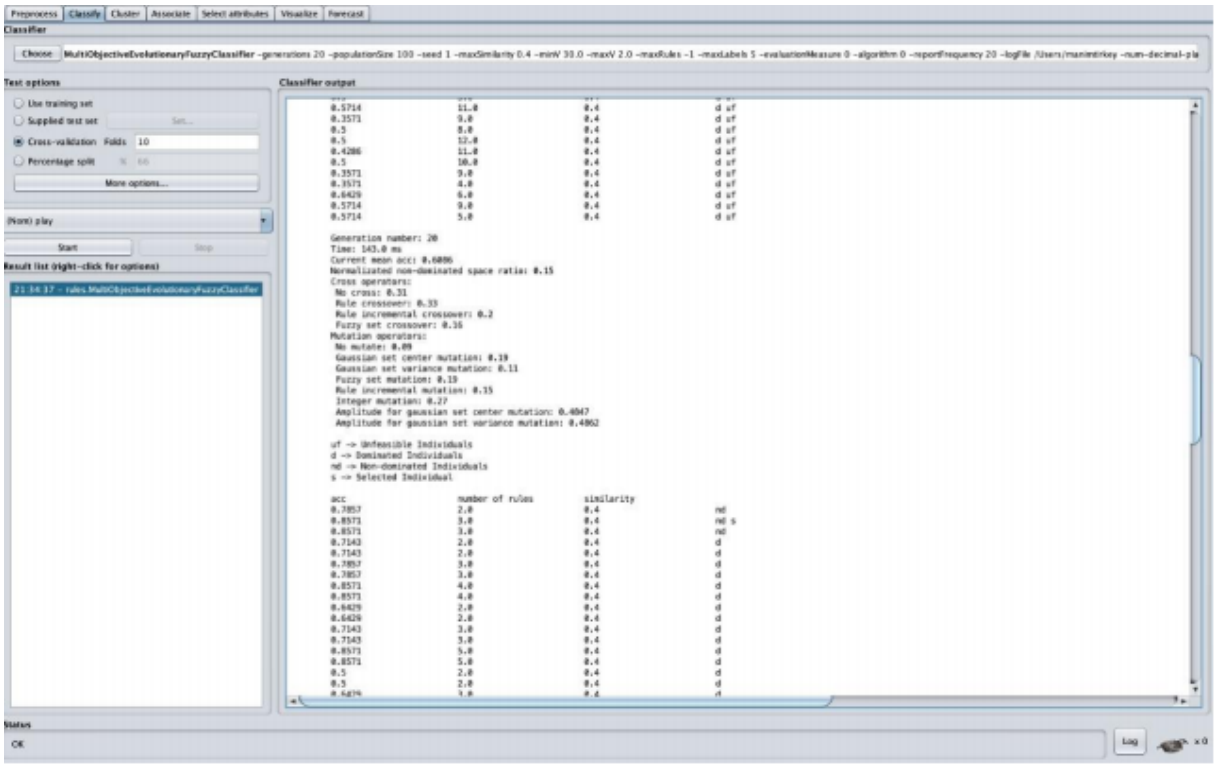
- **ROC Analysis**

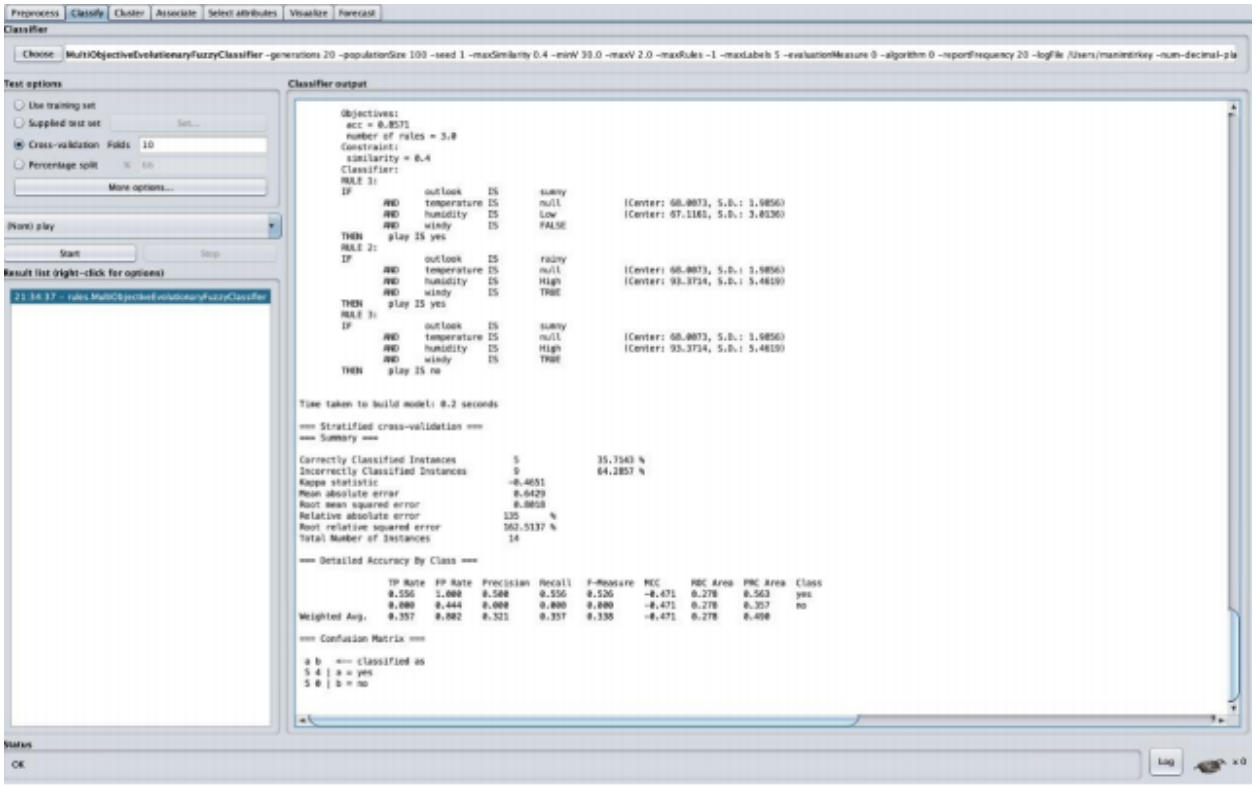
ROC Curve for class tested_negative



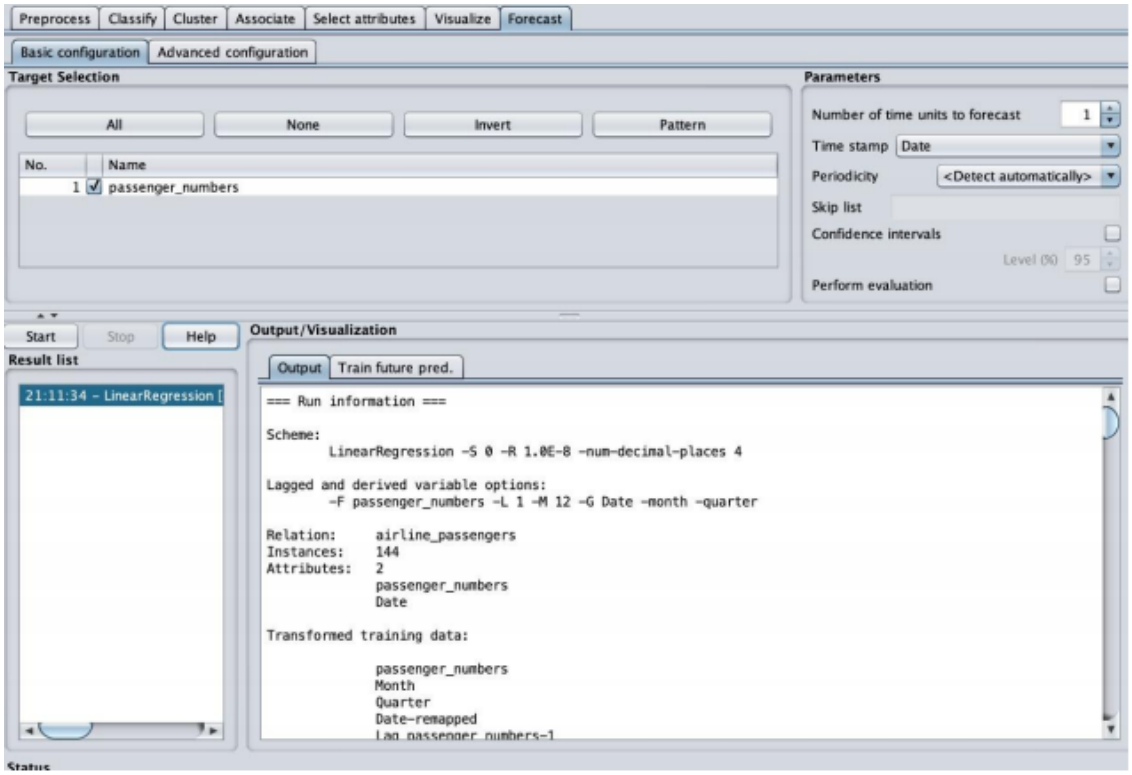
- **Survival Analysis using evolutionary fuzzy classifier**

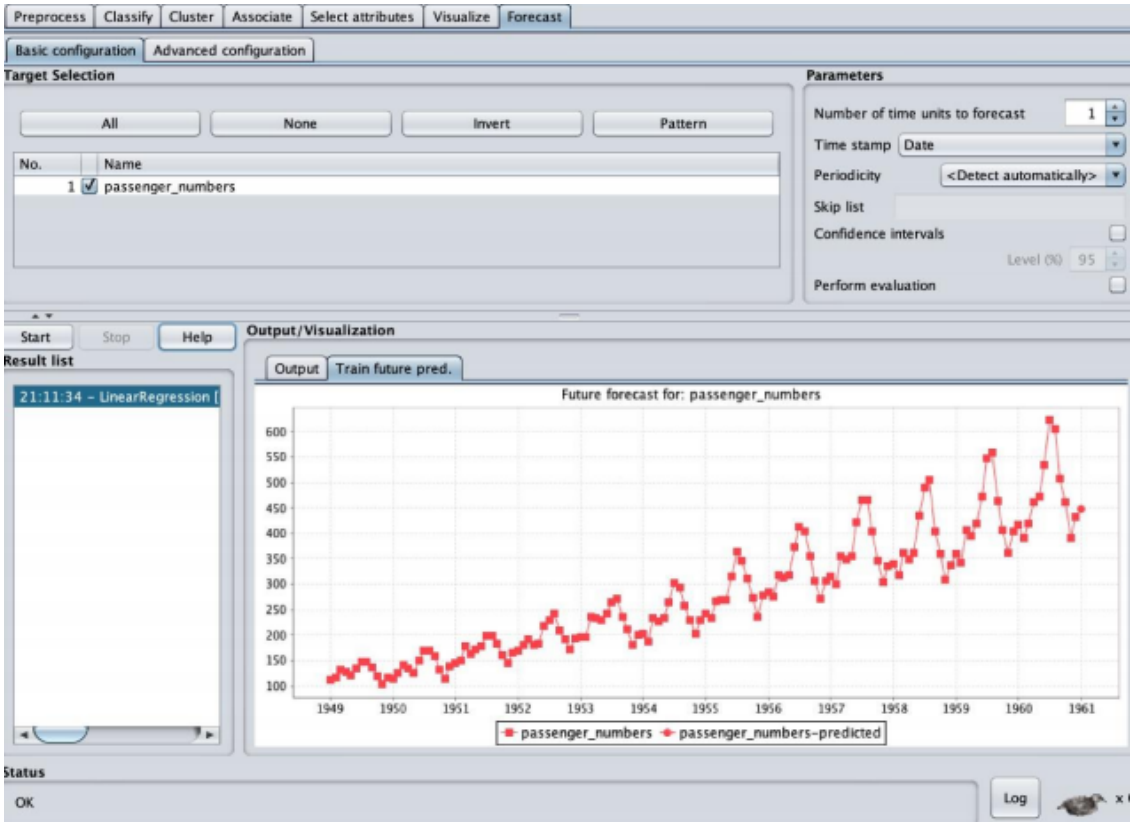






- **Forecast Analysis**





Findings and Learnings :

1. What is ROC Curve and how ROC analysis is useful.
2. What is survival analysis and where it is used.
3. Various ways to perform forecast analysis and its importance.

Experiment 10

Aim: WAP to implement the KNN and model evaluation in python

Theory:

The k-nearest-neighbor method was first described in the early 1950s. The method is labor intensive when given large training sets, and did not gain popularity until the 1960s when increased computing power became available. It has since been widely used in the area of pattern recognition.Nearest-neighbor classifiers are based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it. The training tuples are described by n attributes. Each tuple represents a point in an n-dimensional space. In this way, all the training tuples are stored in an n-dimensional pattern space. When given an unknown tuple, a k-nearest neighbor classifier searches the pattern space for the k training tuples that are closest to the unknown tuple. These k training tuples are the k "nearest neighbors" of the unknown tuple.

For k-nearest-neighbor classification, the unknown tuple is assigned the most common class among its k-nearest neighbors. When k = 1, the unknown tuple is assigned the class of the training tuple that is closest to it in pattern space. Nearest-neighbor classifiers can also be used for numeric prediction, that is, to return a real-valued prediction for a given unknown tuple. In this case, the classifier returns the average value of the real-valued labels associated with the k-nearest neighbors of the unknown tuple

Strengths and Weakness:

Strengths:

- K-NN is pretty intuitive and simple: K-NN algorithm is very simple to understand and equally easy to implement.
- K-NN has no assumptions: K-NN is a non-parametric algorithm which means there are assumptions to be met to implement K-NN.
- No Training Step: K-NN does not explicitly build any model, it simply tags the new data entry based learning from historical data. New data entry would be tagged with majority class in the nearest neighbour.
- It constantly evolves: Given it's an instance-based learning; k-NN is a memory-based approach. The classifier immediately adapts as we collect new training data.
- Very easy to implement for multi-class problems: Most of the classifier algorithms are easy to implement for binary problems and need effort to implement for multi-class whereas K-NN adjusts to multi-class without any extra efforts.
- Can be used both for Classification and Regression: One of the biggest advantages of K-NN is that K-NN can be used both for classification and regression problems.

- One Hyper Parameter: K-NN might take some time while selecting the first hyper parameter but after that rest of the parameters are aligned to it.
- Variety of distance criteria to be chosen from: K-NN algorithm gives users the flexibility to choose distance while building K-NN models.
    - Euclidean , Hamming, Manhattan and Minkowski Distance

Weakness:

- K-NN slow algorithm: K-NN might be very easy to implement but as the dataset grows efficiency or speed of algorithm declines very fast.
- K-NN needs homogeneous features: If you decide to build k-NN using a common distance, like Euclidean or Manhattan distances, it is completely necessary that features have the same scale, since absolute differences in features weight the same, i.e., a given distance in feature 1 must mean the same for feature 2.
- Optimal number of neighbors: One of the biggest issues with K-NN is to choose the optimal number of neighbors to consider while classifying the new data entry.
- Imbalanced data causes problems: k-NN doesn't perform well on imbalanced data. If we consider two classes, A and B, and the majority of the training data is labeled as A, then the model will ultimately give a lot of preference to A.
- Outlier sensitivity: K-NN algorithm is very sensitive to outliers as it simply chose the neighbors based on distance criteria.
- Missing Value treatment: K-NN inherently has no capability of dealing with missing value problem.

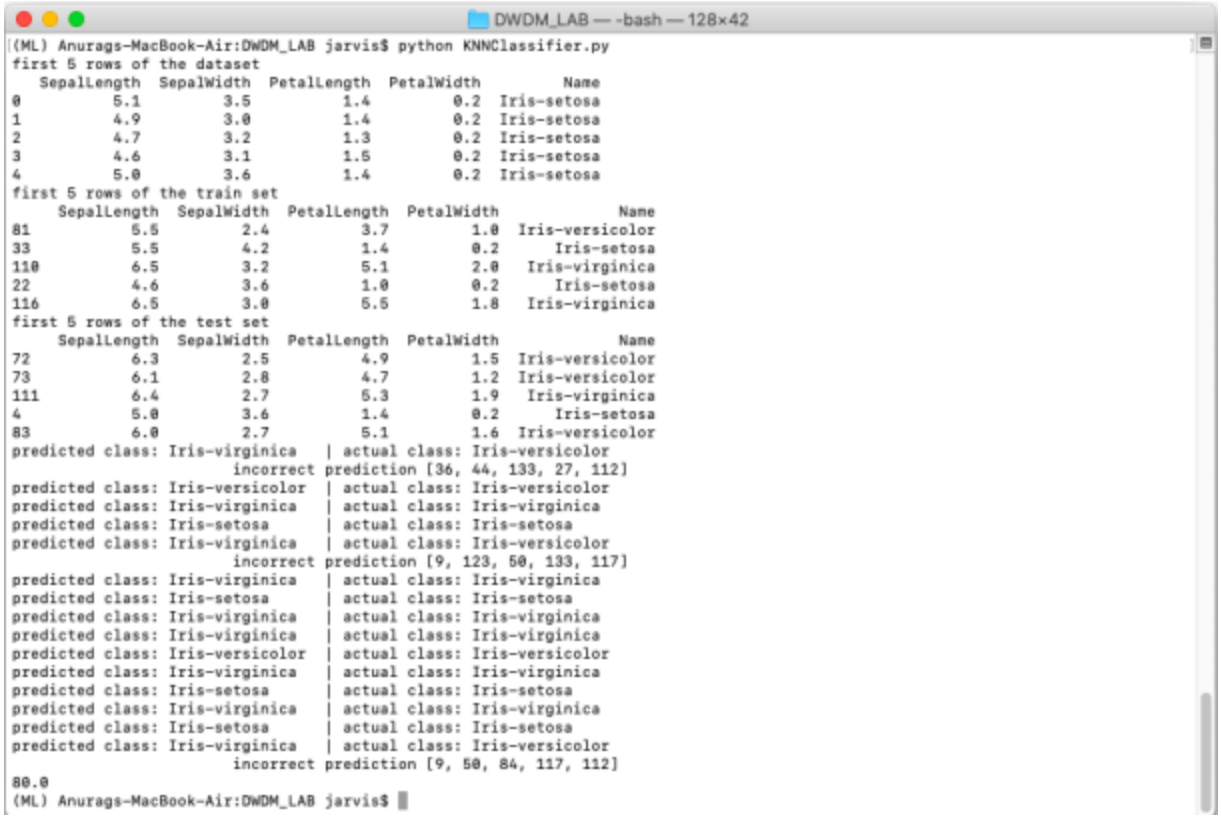
Source Code:

```
import pandas as pd
import numpy as np
import math
import operator

def get_train_test_split(dataset, split_ratio=0.8):
    datalen = len(dataset)
    shuffled = dataset.iloc[np.random.permutation(len(dataset))]
    train_data = shuffled[:int(split_ratio * datalen)]
    test_data = shuffled[int(split_ratio * datalen):]
    return train_data, test_data

def euclidean_dist(p1, p2):
    dist = 0.0
```

```
    for i in range(len(p1)-1):
        dist += pow((float(p1[i])-float(p2[i])),2)
        dist = math.sqrt(dist)
    return dist

def knn(training_dataset, test_value, k):
    distances = {}
    sort = {}
    for x in range(len(training_dataset)):
        dist = euclidean_dist(test_value, training_dataset.iloc[x])
        distances[x] = dist
    sorted_d = sorted(distances.items(), key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(sorted_d[x][0])
    class_votes = {}
    for x in range(len(neighbors)):
        response = training_dataset.iloc[neighbors[x]][-1]
        if response in class_votes:
            class_votes[response] += 1
        else:
            class_votes[response] = 1
    sorted_votes = sorted(class_votes.items(), key=operator.itemgetter(1), reverse=True)
    return sorted_votes[0][0], neighbors
def cal_accuracy(data, test_data, K=5):
    correct = 0
    total = len(test_data)
    for i in range(total):
        pred_class, neigh = knn(data, test_data.iloc[i], K)
                print("predicted class: {:16} | actual class: {:16}".format(pred_class,
test_data.iloc[i]['Name']))
        if pred_class == test_data.iloc[i]['Name']:
            correct += 1
        else:
            print("\t\t\tincorrect prediction", neigh)
    return (correct/total) * 100
```

```
if __name__ == '__main__':
    dataset = pd.read_csv('iris.csv')
    print('first 5 rows of the dataset')
    print(dataset.head(5))
    train_data, test_data = get_train_test_split(dataset, 0.9)
    print('first 5 rows of the train set')
    print(train_data.head(5))
    print('first 5 rows of the test set')
    print(test_data.head(5))
    print(cal_accuracy(train_data, test_data, 5))
```

Output:



Findings and Learnings :

- We have Implemented K-nearest-neighbours algorithm in python.
- We have learned the nuances of the KNN-algorithm.
- We have learnt about the applications, strengths and weaknesses of KNN-algorithm