

DATA WAREHOUSING AND DATA MINING LAB



SUBMITTED TO:
Ms. Shalini Agarwal

MADE BY:
Suman Nandi
2K20/SWE/23
MTech
DTU

Table of Contents

Table of Contents

2

S No.	AIM	PAGE	SIGN
1.	List down various open source data warehouse tools and techniques.	3	
2.	List out various open source data mining tools and techniques and explain them.	7	
3.	Generate Histogram and Box Plot for a sample data in WEKA.	11	
4.	Explain missing value treatment and Outliers treatment in WEKA using Sample dataset.	15	
5.	To perform pre-processing and use filters in Weka using arff file format.	19	
6.	To perform Decision Tree learning using Classification in 1. WEKA. 2. PYTHON	22	
7.	Implement K-Means clustering algorithm on iris dataset using python.	32	
8.	Perform association rule mining using apriori algorithm on any sample dataset.	38	
9.	Implement K-Nearest Neighbor (KNN) in python language.	42	
10.	Write a program in python language to implement model evaluation (Cross validation and generate confusion matrix)	46	

PROGRAM 1

AIM: List down various open source data warehousing tools and techniques.

INTRODUCTION AND THEORY:

In computing, a data warehouse (DW or DWH), also known as an enterprise data warehouse (EDW), 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. They store current and historical data in one single place that are used for creating analytical reports for workers throughout the enterprise.

The data stored in the warehouse is uploaded from the operational systems (such as marketing or sales). The data may pass through an operational data store and may require data cleansing for additional operations to ensure data quality before it is used in the DW for reporting.

The typical extract, transform, load (ETL)-based data warehouse uses staging, data integration, and access layers to house its key functions. The staging layer or staging database stores raw data extracted from each of the disparate source data systems. The integration layer integrates the disparate data sets by transforming the data from the staging layer often storing this transformed data in an operational data store (ODS) database. The integrated data are then moved to yet another database, often called the data warehouse database, where the data is arranged into hierarchical groups, often called dimensions, and into facts and aggregate facts. The combination of facts and dimensions is sometimes called a star schema. The access layer helps users retrieve data.

MarkLogic

MarkLogic is a data warehousing solution that makes data integration easier and faster using an array of enterprise features. This tool helps to perform very complex search operations. It can query data including documents, relationships, and metadata.

Features:

1. The Optic API can perform joins and aggregates over documents, triples, and rows.
2. It allows specifying more complex security rules for all the elements within documents
3. Writing, reading, patching, and deleting documents in JSON, XML, text, or binary formats
4. Database Replication for Disaster Recovery
5. Specify Output Options on the App Server Configuration
6. Importing and Exporting Configuration Information

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.

Features:

1. Distributes data in the same way across disks to offer uniform performance
2. Works for single-instance and real application clusters
3. Offers real application testing
4. Common architecture between any Private Cloud and Oracle's public cloud
5. Hi-Speed Connection to move large data
6. Works seamlessly with UNIX/Linux and Windows platforms
7. It provides support for virtualization
8. Allows connecting to the remote database, table, or view

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.

Features:

1. Help you to build your dream dashboard
2. Stay connected anywhere you go
3. Integrates all existing business data
4. Helps you to get true insights into your business data
5. Connects all of your existing business data
6. Easy Communication & messaging platform
7. It provides support for ad-hoc queries using SQL
8. It can handle most concurrent users for running complex and multiple queries

SAS

SAS is a leading Data warehousing tool that allows accessing data across multiple sources. It can perform sophisticated analyses and deliver information across the organization.

Features:

1. Activities managed from central locations. Hence, user can access applications remotely via the Internet
2. Application delivery typically closer to a one-to-many model instead of one-to-one model
3. Centralized feature updating, allows the users to download patches and upgrades.
4. Allows viewing raw data files in external databases
5. Manage data using tools for data entry, formatting, and conversion
6. Display data using reports and statistical graphics.

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.

Features:

1. Support for Big Data and Hadoop
2. Additional storage or services can be accessed without need to install new software and hardware
3. Real time data integration
4. Provide trusted ETL data anytime, anywhere
5. Solve complex big data challenges
6. Optimize hardware utilization and prioritize mission-critical tasks
7. Deploy on-premises or in the cloud

Informatica:

Informatica PowerCenter is Data Integration tool developed by Informatica Corporation. The tool offers the capability to connect & fetch data from different sources.

Features:

1. It has a centralized error logging system which facilitates logging errors and rejecting data into relational tables
2. Build in Intelligence to improve performance
3. Limit the Session Log
4. Ability to Scale up Data Integration
5. Foundation for Data Architecture Modernization
6. Better designs with enforced best practices on code development
7. Code integration with external Software Configuration tools
8. Synchronization amongst geographically distributed team members

QuerySurge

QuerySurge is 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.

Features:

1. Improve data quality & data governance
2. Accelerate your data delivery cycles
3. Helps to automate manual testing effort
4. Provide testing across the different platform like Oracle, Teradata, IBM, Amazon, Cloudera, etc.
5. It speeds up testing process up to 1,000 x and also providing up to 100% data coverage
6. It integrates an out-of-the-box DevOps solution for most Build, ETL & QA management software
7. Deliver shareable, automated email reports and data health dashboards

Findings and Learnings:

1. We learned about data warehousing and its significance.
2. We learned which tools can be used to assist us.
3. We learned about the features of the tools.

PROGRAM 2

AIM: List down the various open source data mining tools and techniques and their advantages and disadvantages.

INTRODUCTION AND THEORY:

Rapid Miner

Rapid Miner 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

1. Rapid Miner Studio- This module is for workflow design, prototyping, validation etc.
2. Rapid Miner Server- To operate predictive data models created in studio
3. Rapid Miner Radoop- Executes processes directly in Hadoop cluster to simplify predictive analysis.

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

1. Showing data table and allowing to select features
2. Reading the data
3. Training predictors and to compare learning algorithms
4. 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 short time by quickly comparing & analyzing the data.

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

1. Extensible programming environment
2. Pre-made algorithms
3. Math experimentation environment
4. GPU computes for performance improvement

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.

WEKA

Waikato Environment for Knowledge Analysis (Weka) is a suite of machine learning software written in Java, developed at the University of Waikato, New Zealand. It is free software licensed under the GNU General Public License.

Weka contains a collection of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces for easy access to these functions. The original non-Java version of Weka was a Tcl/Tk front-end to (mostly third-party) modeling algorithms implemented in other programming languages, plus data preprocessing utilities in C, and a Make file-based system for running machine learning experiments. This original version was primarily designed as a tool for analyzing data from agricultural domains, but the more recent fully Java-based version (Weka 3), for which development started in 1997, is now used in many different application areas, in particular for educational purposes and research.

Advantages of Weka include:

1. Free availability under the GNU General Public License.
2. Portability, since it is fully implemented in the Java programming language and thus runs on almost any modern computing platform.
3. A comprehensive collection of data preprocessing and modeling techniques.
4. Ease of use due to its graphical user interfaces.

Weka supports several standard data mining tasks, more specifically, data preprocessing, clustering, classification, regression, visualization, and feature selection. All of Weka's techniques are predicated on the assumption that the data is available as one flat file or relation, where each data point is described by a fixed number of attributes (normally, numeric or nominal attributes, but some other attribute types are also supported). Weka provides access to SQL databases using Java Database Connectivity and can process the result returned by a database query.

Interface

Weka's main user interface is the Explorer, but essentially the same functionality can be accessed through the component-based Knowledge Flow interface and from the command line. There is also the Experimenter, which allows the systematic comparison of the predictive performance of Weka's machine learning algorithms on a collection of datasets. The Explorer interface features several panels providing access to the main components of the workbench:

- The Preprocess panel has facilities for importing data from a database, a comma separated values (CSV) file, etc., and for preprocessing this data using a so-called filtering algorithm. These filters can be used to transform the data (e.g., turning numeric attributes into discrete ones) and make it possible to delete instances and attributes according to specific criteria.
- The Associate panel provides access to association rule learners that attempt to identify all important interrelationships between attributes in the data.
- The Cluster panel gives access to the clustering techniques in Weka, e.g., the simple k-means algorithm. There is also an implementation of the expectation maximization algorithm for learning a mixture of normal distributions.
- The Select attributes panel provides algorithms for identifying the most predictive attributes in a dataset.
- The Visualize panel shows a scatter plot matrix, where individual scatter plots can be selected and enlarged, and analyzed further using various selection operators.

Regression

Regression, used primarily as a form of planning and modeling, 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.

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. Classification method makes use of mathematical techniques such as decision trees, linear programming, neural network, and statistics. In classification, we develop the 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 that named "leave" and "stay". And then we can ask our data mining software to classify the employees into separate groups.

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

Sequential Patterns

Sequential patterns analysis is one of 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 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.

Decision trees

The A decision tree is one of the most commonly used data mining techniques because its model is easy to understand for users. In decision tree 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:

1. There are wide variety of open source data mining tools available.
2. They vary greatly in their core specialties.
3. They also vary a lot in the way they approach data mining and the various algorithms they use.
4. Different techniques are available for data mining depending on the type of data and amount of data available.

PROGRAM 3

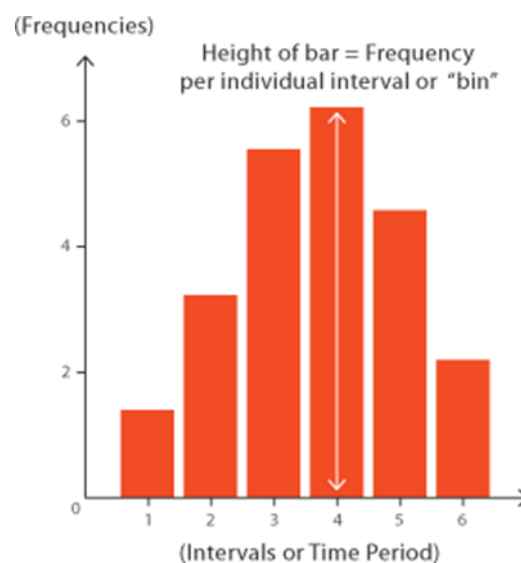
AIM: To generate histogram and box plot for a sample data in Weka.

INTRODUCTION AND 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.

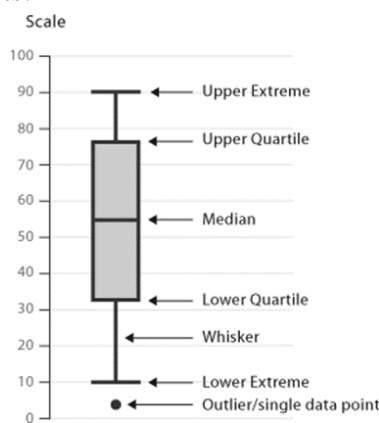
A Histogram visualizes the distribution of data over a continuous interval or certain time period. Each bar in a histogram represents the tabulated frequency at each interval/bin.

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 Box Plot are: variable-width Box Plots and notched Box Plots.

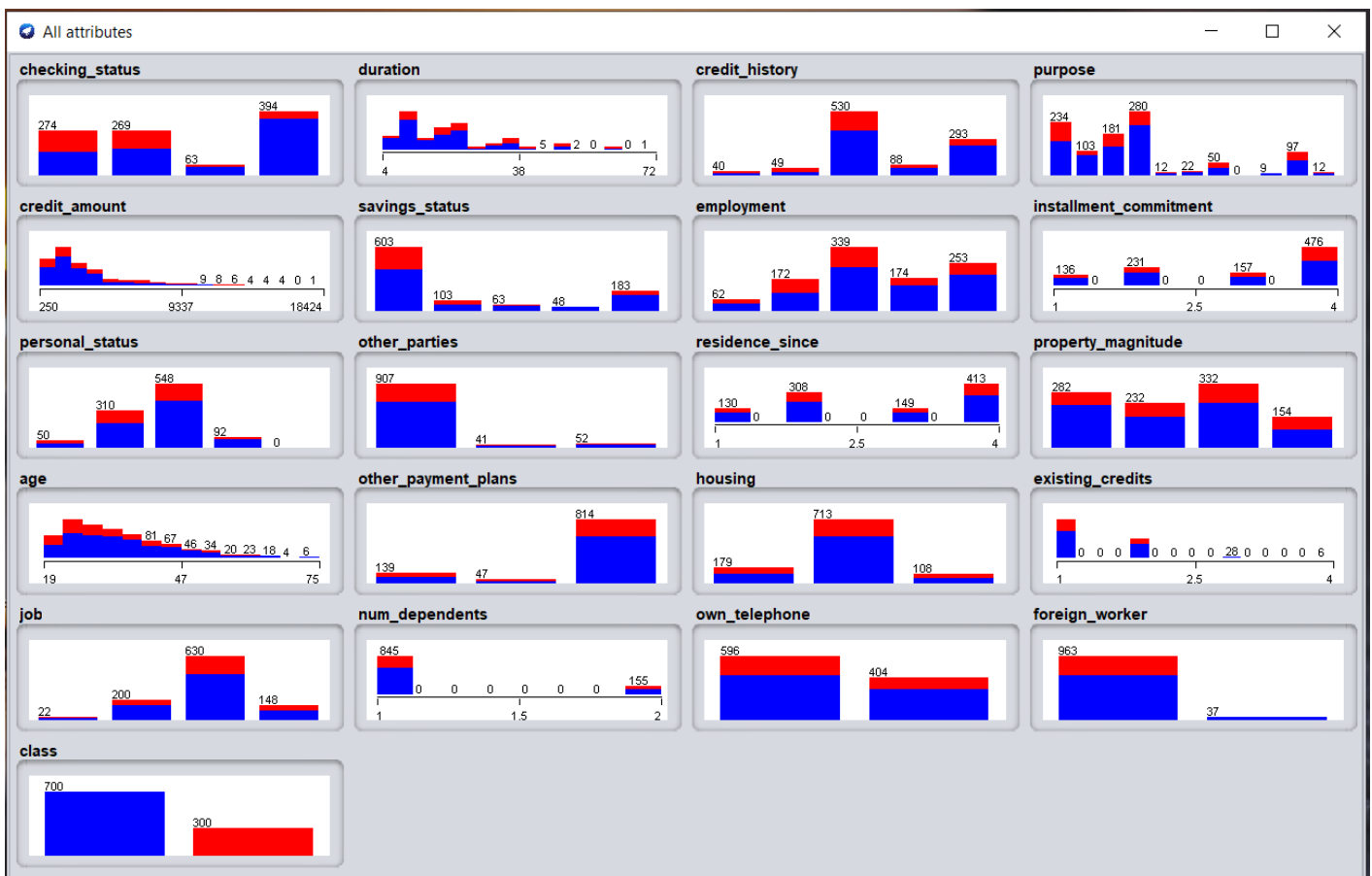
A Five Number Summary includes:

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

PROCEDURE

1. Plotting histograms

1. Go to weka explorer.
2. Choose dataset in weka-3.8.3/data. (Dataset Chosen: german_credit)
3. Above histogram click visualize all.



2. Plotting Box-Plots

1. Go to weka explorer
2. Choose dataset in weka-3.8.3/data
3. Use Cpython scripting and pandas DataFrame boxplot method
4. Plot the boxplot using Cpython

Python script

```
1 fig, axs = plt.subplots(1, 1, figsize=(10, 5))
2 plt.title('BoxPlot')
3 py_data.boxplot()
4
```

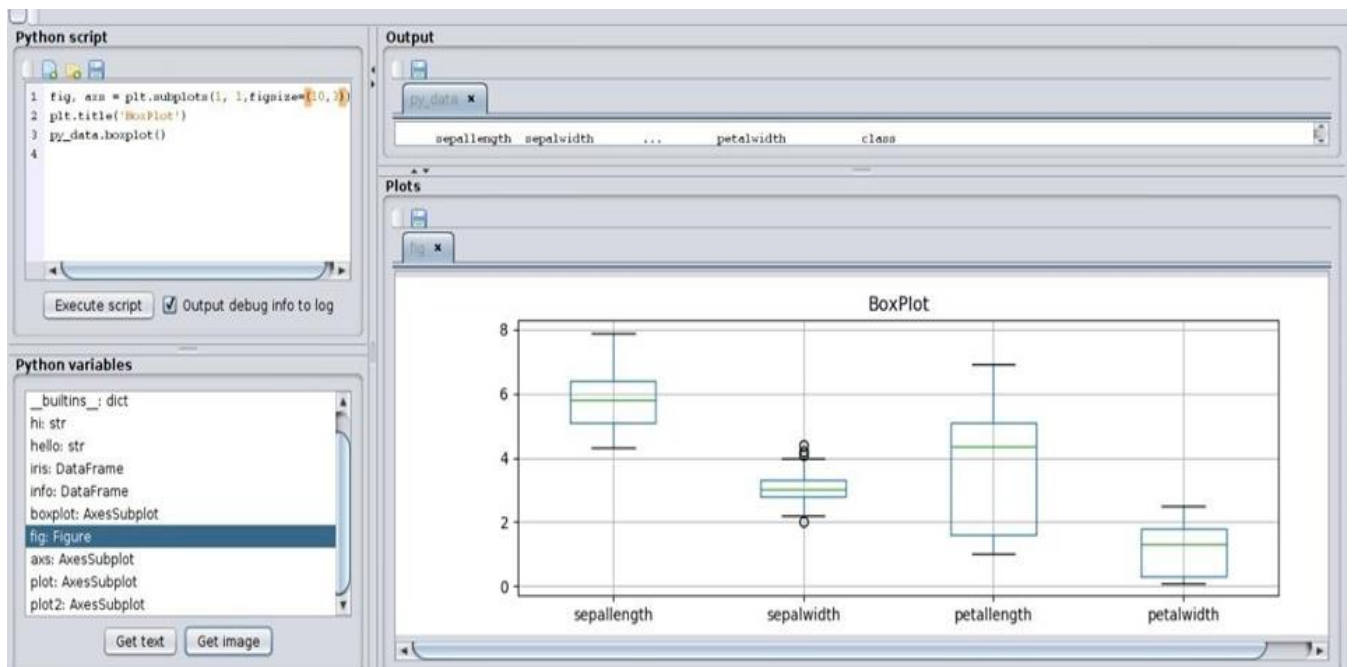
Output

	sepalwidth	sepalwidth	...	petalwidth	class
0	5.1	3.5	...	0.2	Iris-setosa
1	4.9	3.0	...	0.2	Iris-setosa
2	4.7	3.2	...	0.2	Iris-setosa
3	4.6	3.1	...	0.2	Iris-setosa
4	5.0	3.6	...	0.2	Iris-setosa
5	5.4	3.9	...	0.4	Iris-setosa
6	4.6	3.4	...	0.3	Iris-setosa
7	5.0	3.4	...	0.2	Iris-setosa
8	4.4	2.9	...	0.2	Iris-setosa
9	4.9	3.1	...	0.1	Iris-setosa
10	5.4	3.7	...	0.2	Iris-setosa
11	4.8	3.4	...	0.2	Iris-setosa
12	4.8	3.0	...	0.1	Iris-setosa
13	4.3	3.0	...	0.1	Iris-setosa
14	5.8	4.0	...	0.2	Iris-setosa
15	5.7	4.4	...	0.4	Iris-setosa
16	5.4	3.9	...	0.4	Iris-setosa
17	5.1	3.5	...	0.3	Iris-setosa

Python variables

- __builtins__: dict
- hi: str
- hello: str
- iris: DataFrame
- info: DataFrame
- boxplot: AxesSubplot
- fig: Figure
- axs: AxesSubplot
- plot: AxesSubplot
- plot2: AxesSubplot

Plots



FINDINGS AND LEARNINGS:

1. 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.
2. Weka Software provides a good set of functions to plot histograms and box plots with various combination of attributes.
3. We have successfully plotted histograms and boxplots in WEKA.

PROGRAM 4

AIM: Explain missing value treatment and outlier treatment in WEKA on sample dataset.

INTRODUCTION AND THEORY

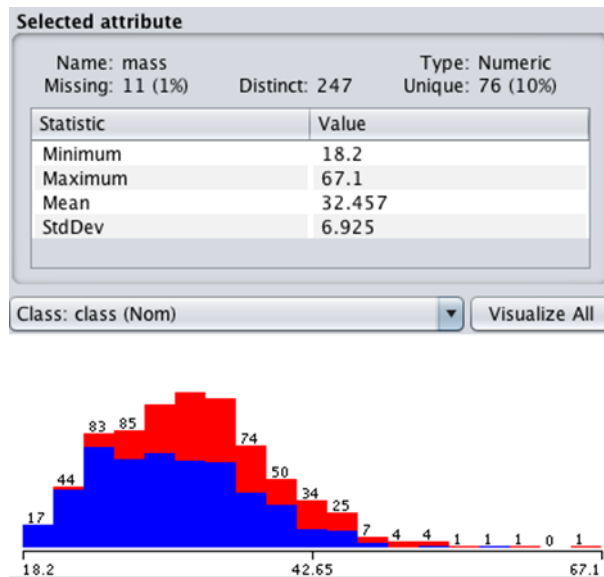
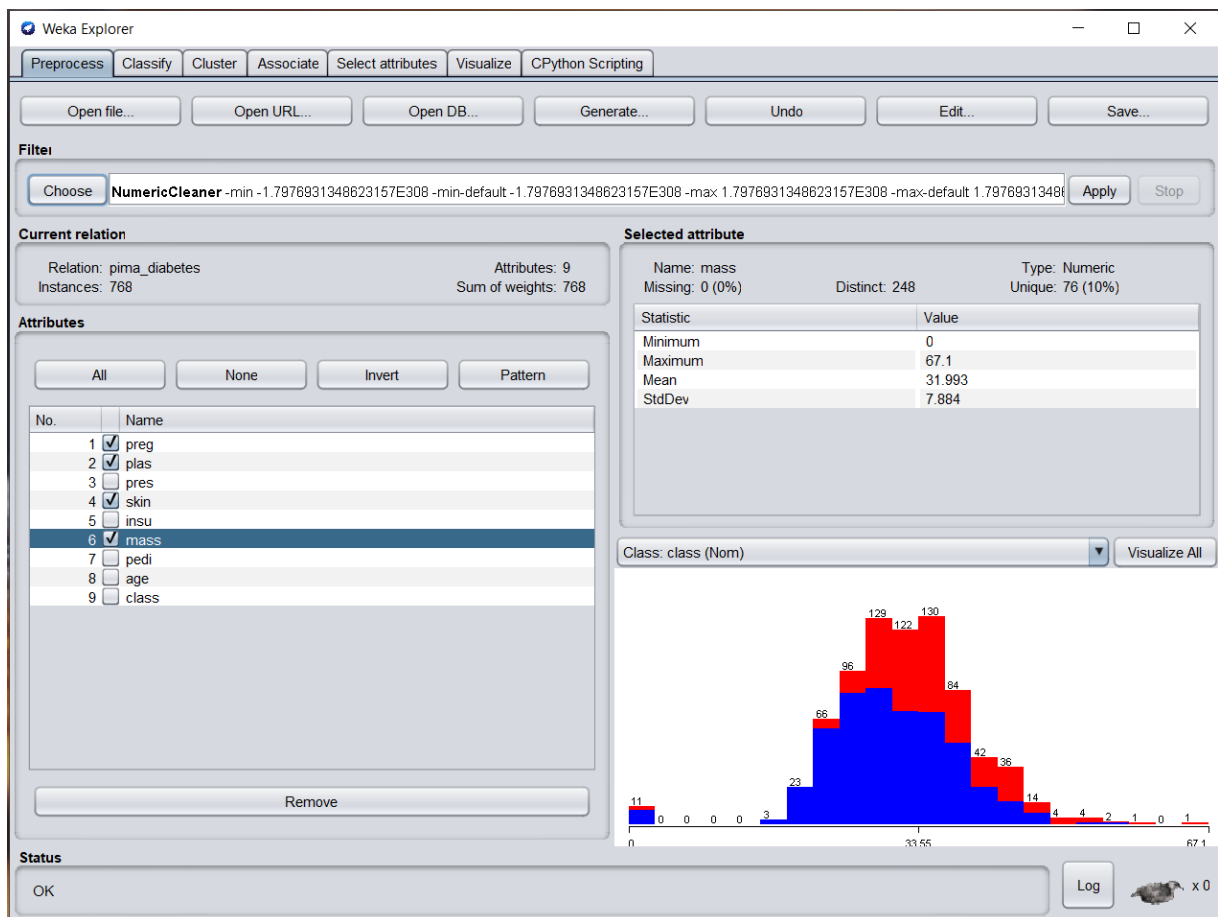
Data is rarely clean and often we can have corrupt or missing values. It is important to identify, mark and handle missing data when developing machine learning models in order to get the best performance.

In most cases it is normal to find some values missing in the data, in other cases there may be some data elements whose values don't match with any other data points, such values are called outliers, normally any value outside of 1.5 quantiles of the data mean is taken as an outlier.

PROCEDURE

1. MARK MISSING VALUES

1. Open the WEKA Explorer.
2. Load the Pima Indians onset of diabetes dataset.
3. Click the "Choose" button for the Filter and select NumericalCleaner, it is under unsupervised.attribute.NumericalCleaner.
4. Click on the filter to configure it.
5. Set the attributeIndices to 6, the index of the mass attribute.
6. Set minThreshold to 0.1E-8 (close to zero), which is the minimum value allowed for the attribute.
7. Set minDefault to NaN, which is unknown and will replace values below the threshold.
8. Click the "OK" button on the filter configuration.
9. Click the "Apply" button to apply the filter.



2. REMOVING MISSING VALUES

1. Click the “Choose” button for the Filter and select RemoveWithValues, it is under unsupervised.instance.RemoveWithValues.
2. Click on the filter to configure it.
3. Set the attributeIndicies to 6, the index of the mass attribute.
4. Set matchMissingValues to “True”.
5. Click the “OK” button to use the configuration for the filter.
6. Click the “Apply” button to apply the filter.

Selected attribute

Name: mass
Missing: 0 (0%)

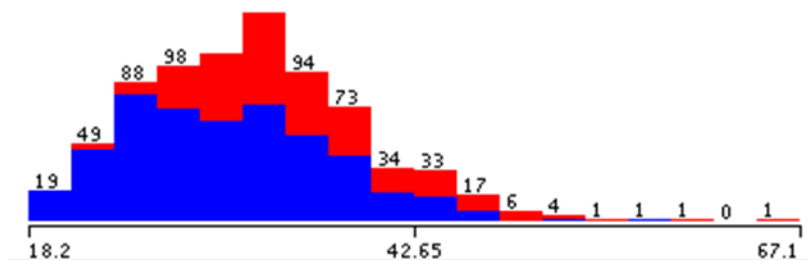
Distinct: 247

Type: Numeric
Unique: 76 (10%)

Statistic	Value
Minimum	18.2
Maximum	67.1
Mean	32.457
StdDev	6.925

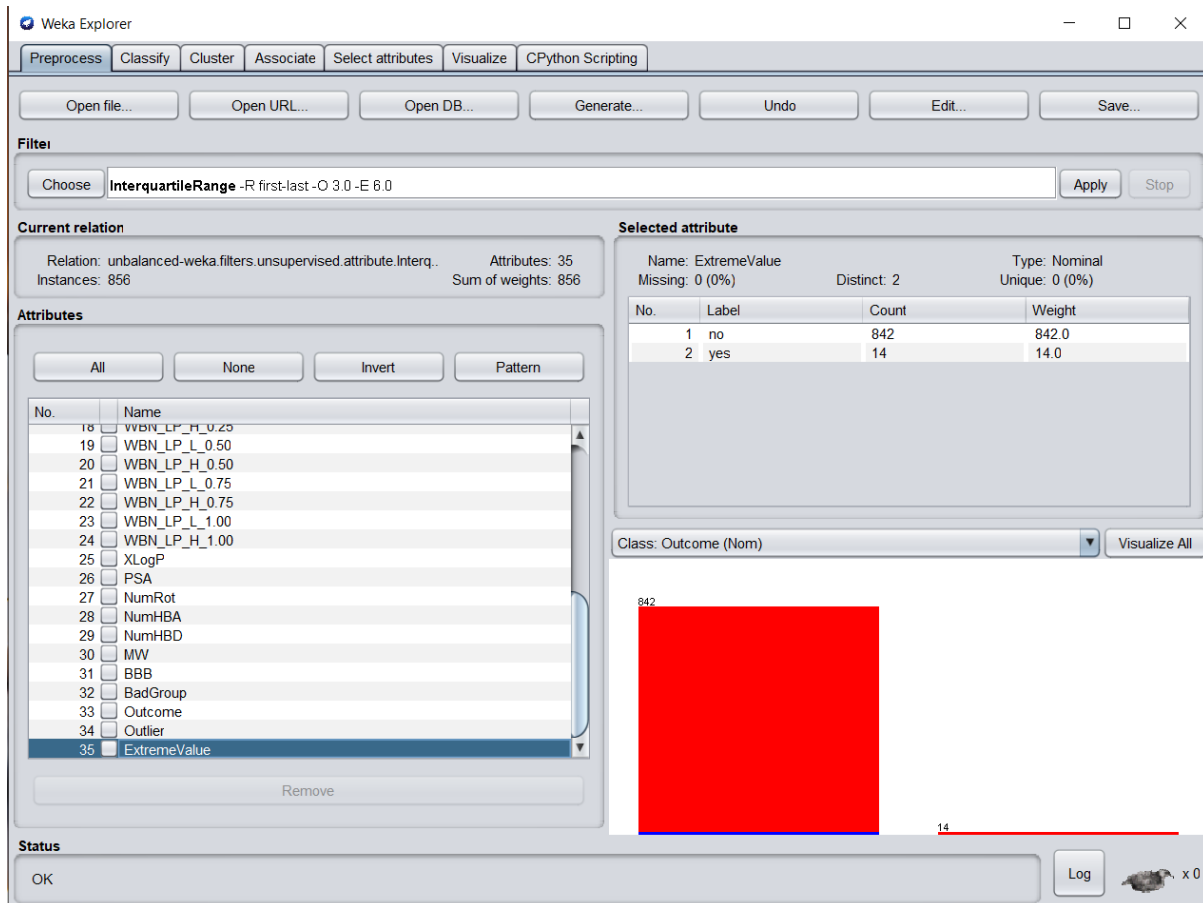
Class: class (Nom)

Visualize All



3. OUTLINER TREATMENT

1. Open the Weka Explorer.
2. Load the Unbalanced dataset.
3. Click the “Choose” button for the Filter and select InterquartileRange, it under unsupervised.attribute.InterquartileRange
4. Click on the filter to configure it.
5. Click the “OK” button on the filter configuration.
6. Click the “Apply” button to apply the filter.



FINDINGS AND LEARNINGS:

1. Missing value treatment and outlier removal are very important steps to be performed before data analysis and statistical modeling.
2. Weka Software makes it easy and simple to visualize data and apply appropriate treatments to remove the anomalies.

PROGRAM 5

AIM: To perform pre-processing and use filters in Weka.

INTRODUCTION AND THEORY

Data preprocessing is an important step in the data mining process. The phrase "garbage in, garbage out" is particularly applicable to data mining and machine learning projects. Data gathering methods are often loosely controlled, resulting in out-of-range values (e.g., Income: -100), impossible data combinations (e.g., Sex: Male, Pregnant: Yes), missing values, etc. Analyzing data that has not been carefully screened for such problems can produce misleading results. Thus, the representation and quality of data is first and foremost before running an analysis. Often, data preprocessing is the most important phase of a machine learning project, especially in computational biology.

If there is much irrelevant and redundant information present or noisy and unreliable data, then knowledge discovery during the training phase is more difficult. Data preparation and filtering steps can take considerable amount of processing time. Data preprocessing includes cleaning, Instance selection, normalization, transformation, feature extraction and selection, etc. The product of data preprocessing is the final training set.

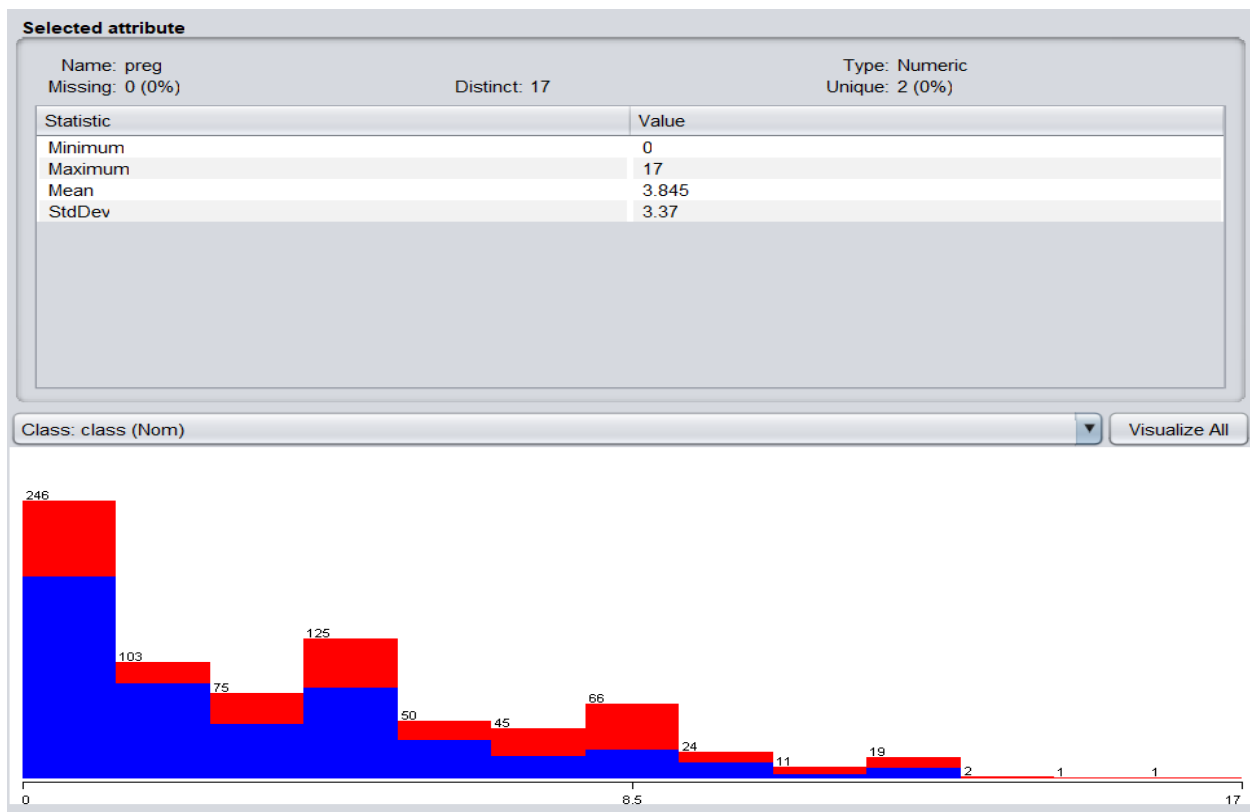
STEPS IN DATA PREPROCESSING

1. Import libraries
2. Read data
3. Checking for missing values
4. Checking for categorical data
5. Standardize the data
6. PCA transformation
7. Data splitting

PROCEDURE

PRE-PROCESSING

1. Open the Weka Explorer.
2. Load the Pima Indians onset of diabetes dataset.
3. Choose->unsupervised->instance and select any filter like RemoveRange
4. Select it then click the text box side to choose button.
5. Add range value in the dialog box
6. Click apply button.



Before Remove Range



After Remove Range

FINDINGS AND LEARNINGS:

1. Data preprocessing and filtering is an important step that needs to be looked into before formal statistical modeling and analysis.
2. Weka Software provides a good interface and set of functions to preprocess and filter the data according to our needs.

PROGRAM 6

AIM: To perform Decision Tree learning using Classification in WEKA and Python.

INTRODUCTION AND 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., nonbacktracking) 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.

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* \leftarrow *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 D_j be the set of data tuples in D satisfying outcome j ; // a partition
- (12) **if** D_j 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**(D_j , *attribute_list*) to node N ;
- endfor**
- (15) return N ;

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

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

WEAKNESS

1. Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
2. Decision trees are prone to errors in classification problems with many class and relatively small number of training examples.
3. Decision tree 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.

PROCEDURE

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

CLASSIFICATION USING PYTHON:

ID3 uses Information gain:

$$\begin{aligned} Info(D) &= - \sum_{i=1}^m p_i \log_2(p_i) \\ Info_A(D) &= \sum_{j=1}^v \frac{|D_j|}{|D|} \times Info(D_j) \\ Gain(A) &= Info(D) - info_A(D) \end{aligned}$$

C4.5 uses gain ratio

$$\begin{aligned} SplitInfo_A(D) &= - \sum_{j=1}^v \frac{|D_j|}{|D|} \times \log_2 \frac{|D_j|}{|D|} \\ GainRatio(A) &= \frac{Gain(A)}{SplitInfo_A(D)} \end{aligned}$$

CART uses the GINI index

$$\begin{aligned} Gini(D) &= 1 - \sum_{i=1}^m p_i^2 \\ Gini_A(D) &= \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2) \\ \Delta Gini(A) &= Gini(D) - Gini_A(D) \end{aligned}$$

PYTHON CODE:

```
1 # importing libraries
2 import pandas as pd
3 import numpy as np
4 from pprint import pprint
5
6 # calculates the entropy
7 def entropy(target_col):
8     elements, counts = np.unique(target_col, return_counts=True)
9     result = np.sum([(
10 counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts)) for i in
11 range(len(elements))])
12     return result
13
14 # calculate the information gain = Gain(A) / SplitInfo_A(D)
15 def information_gain(data, split_attribute_name,
16 target_name="class"):
17     """
18     calculates the information gain of the dataset for attribute A
19     :param data: the dataset for which we're calculating the
20 information gain
21     :param split_attribute_name: name of the splitting attribute.
22     :param target_name: name of the target / class.
23     :return: the information gain
24     """
25     # Calculate the entropy of the total dataset
26     total_entropy = entropy(data[target_name])
27
28     # Calculate the values and the corresponding counts for the
29 split attribute
30     vals, counts = np.unique(data[split_attribute_name],
31 return_counts=True)
32     # Calculate the weighted entropy
33     weighted_entropy = np.sum(
34 [(counts[i] / np.sum(counts)) *
35 entropy(data.where(data[split_attribute_name] ==
36 vals[i]).dropna()[target_name])
37 for i in range(len(vals))])
38     # Calculate the information gain
39     information_gain_ = total_entropy - weighted_entropy
40     return information_gain_
41
42 # the ID3 Decision tree algorithm
43 def id3(data, original_data, features,
44 target_attribute_name="class", parent_node_class=None):
45     """
46     id3 algorithm
47     :param data: the dataset for which we're running the id3
48 algorithm
49     :param original_data: the original dataset needed to calculate
50 the mode target feature value of the original dataset
51 in the case the dataset delivered by the first parameter is
52 empty
53     :param features: the feature space of the dataset . This is
54 needed for the recursive call since during the tree
55
```

```

56     growing process we have to remove features from our dataset ->
57     Splitting at each node
58     :param target_attribute_name: the name of the target attribute
59     :param parent_node_class: This is the value or class of the mode
60     target feature value of the parent node for a
61     specific node. This is also needed for the recursive call since
62     if the splitting leads to a situation that there are
63     no more features left in the feature space, we want to return
64     the mode target feature value of the direct parent node.
65
66     :return: The learnt decision tree
67     """
68     # Define the stopping criteria -> If one of this is satisfied,
69     we want to return a leaf node
70     # If all target_values have the same value, return this value
71     if len(np.unique(data[target_attribute_name])) <= 1:
72         return np.unique(data[target_attribute_name])[0]
73     # If the dataset is empty, return the mode target feature value
74     in the original dataset
75     elif len(data) == 0:
76         return np.unique(original_data[target_attribute_name])[
77
78     np.argmax(np.unique(original_data[target_attribute_name],
79     return_counts=True)[1])]
80     # If the feature space is empty, return the mode target feature
81     value of the direct parent node -> Note that
82     # the direct parent node is that node which has called the
83     current run of the ID3 algorithm and hence
84     # the mode target feature value is stored in the
85     parent_node_class variable.
86     elif len(features) == 0:
87         return parent_node_class
88     # If none of the above holds true, grow the tree!
89     else:
90         # Set the default value for this node -> The mode target
91         feature value of the current node
92         parent_node_class = np.unique(data[target_attribute_name])[
93         np.argmax(np.unique(data[target_attribute_name],
94     return_counts=True)[1])]
95         # Select the feature which best splits the dataset
96         item_values = [information_gain(data, feature,
97     target_attribute_name) for feature in
98         features] # Return the information gain
99     values for the features in the dataset
100         best_feature_index = np.argmax(item_values)
101         best_feature = features[best_feature_index]
102         # Create the tree structure. The root gets the name of the
103     feature (best_feature) with the maximum information
104         # gain in the first run
105         tree = {best_feature: {}}
106         # Remove the feature with the best information gain from the
107     feature space
108         features = [i for i in features if i != best_feature]
109         # Grow a branch under the root node for each possible value
110     of the root node feature
111         for value in np.unique(data[best_feature]):
112             value = value

```

```

113         # Split the dataset along the value of the feature with
114 the largest
115         # information gain and then create sub_datasets
116         sub_data = data.where(data[best_feature] ==
117 value).dropna()
118
119         # Call the ID3 algorithm for each of those sub_datasets
120 with the
121         # new parameters -> Here the recursion comes in!
122         subtree = id3(sub_data, dataset, features,
123 target_attribute_name, parent_node_class)
124
125         # Add the sub tree, grown from the sub_dataset to the
126 tree under the root node
127         tree[best_feature][value] = subtree
128
129     return tree
130
131
132 def predict(query, tree, default=1):
133     """
134     prediction function for new unseen data
135     :param query: a dictionary entry, {"feature_name" : value, ... ,
136 "feature_name" : value}
137     :param tree: the ID3 tree
138     :param default: base value
139     :return: predicted class
140     """
141     # 1. Check for every feature in the query instance if this feature is
142 existing in the tree.keys() for the first call, tree.keys() only contains
143 the value for the root node -> if this value is not existing, we can not
144 make a prediction and have to return the default value which is the majority
145 value of the target feature
146     for key in list(query.keys()):
147         if key in list(tree.keys()):
148             # 2. First of all we have to take care of a important
149 fact: Since we train our model with a database A and then show our
150 model a unseen query it may happen that the feature values of these
151 query are not existing in our tree model because none of the training
152 instances has had such a value for this specific feature.
153             # For instance imagine the situation where your model
154 has only seen animals with one to four legs - The "legs" node in
155 your model will only have four outgoing branches (from one to four).
156 If you now show your model a new instance (animal) which has for the
157 legs feature the value 5, you have to tell your model what to do in
158 such a situation because otherwise there is no classification
159 possible
160             # because in the classification step you try to run
161 down the outgoing branch with the value 5 but there is no such a
162 branch. Hence: Error and no Classification! We can address this
163 issue with a classification value of for instance (999) which tells
164 us that there is no classification possible or we assign the most
165 frequent target feature value of our dataset used to train the
166 model. Or, in for instance medical application we can return the
167 most worse case - just to make sure... We can also return the most
168 frequent value of the direct parent node. To make a long story
169 short, we have to tell the model what to do in this situation. In

```

```

170 our example, since we are dealing with animal species where a false
171 classification is not that critical, we will assign the value 1
172 which is the value for the mammal species (for convenience).
173     try:
174         result = tree[key][query[key]]
175     except:
176         return default
177
178     # 3. Address the key in the tree which fits the value
179     for key --> Note that key == the features in the query. Because we
180     want the tree to predict the value which is hidden under the key
181     value (imagine you have a drawn tree model on the table in front of
182     you and you have a query instance for which you want to predict the
183     target feature - What are you doing? - Correct: You start at the
184     root node and wander down the tree comparing your query to the node
185     values. Hence you want to have the value which is hidden under the
186     current node. If this is a leaf, perfect, otherwise you wander the
187     tree deeper until you get to a leaf node. Though, you want to have
188     this "something" [either leaf or sub_tree] which is hidden under the
189     current node and hence we must address the node in the tree which ==
190     the key value from our query instance. This is done with
191     tree[keys]. Next you want to run down the branch of this node which
192     is equal to the value given "behind" the key value of your query
193     instance e.g. if you find "legs" == to tree.keys() that is, for the
194     first run == the root node. You want to run deeper and therefore you
195     have to address the branch at your node whose value is == to the
196     value behind key. This is done with query[key]
197     #     e.g. query[key] == query['legs'] == 0 --> Therewith
198     we run down the branch of the node with the value 0. Summarized, in
199     this step we want to address the node which is hidden behind a
200     specific branch of the root node (in the first run) this is done
201     with: result = [key][query[key]]
202         result = tree[key][query[key]]
203     # 4. As said in the 2. step, we run down the tree along
204     nodes and branches until we get to a leaf node. That is, if result
205     = tree[key][query[key]] returns another tree object (we have
206     represented this by a dict object -> that is if result is a dict
207     object) we know that we have not arrived at a root node and have to
208     run deeper the tree. Okay... Look at your drawn tree in front of
209     you... what are you doing?...well, you run down the next branch...
210     exactly as we have done it above with the slight difference that we
211     already have passed a node and therewith have to run only a fraction
212     of the tree --> You clever guy! That "fraction of the tree" is
213     exactly what we have stored under 'result'. So we simply call our
214     predict method using the same query instance (we do not have to drop
215     any features from the query instance since for instance the feature
216     for the root node will not be available in any of the deeper
217     sub_trees and hence we will simply not find that feature) as well as
218     the "reduced / sub_tree" stored in result.
219         if isinstance(result, dict):
220             return predict(query, result)
221         else:
222             return result
223     # splitting the data into test-train splits for checking performance
224     on unseen data
225     def train_test_split(dataset):
226

```

```

227     training_data = dataset.iloc[:80].reset_index(drop=True) # We
228     drop the index respectively relabel the index
229     # starting form 0, because we do not want to run into errors
230     regarding the row labels / indexes
231     testing_data = dataset.iloc[80:].reset_index(drop=True)
232     return training_data, testing_data
233
234 # testing the tree model, get prediction accuracy
235 def test(data, tree):
236     # Create new query instances by simply removing the target
237     feature column from the original dataset and
238     # convert it to a dictionary
239     queries = data.iloc[:, :-1].to_dict(orient="records")
240
241     # Create a empty DataFrame in whose columns the prediction of
242     the tree are stored
243     predicted = pd.DataFrame(columns=["predicted"])
244
245     # Calculate the prediction accuracy
246     for i in range(len(data)):
247         predicted.loc[i, "predicted"] = predict(queries[i], tree,
248         1.0)
249     print('The prediction accuracy is: ',
250     (np.sum(predicted["predicted"] == data["class"]) / len(data)) * 100,
251     '%')
252
253
254
255 """
256 Train the tree, Print the tree and predict the accuracy
257 """
258
259 if __name__ == ' main ':
260     # loading the dataset
261     dataset = pd.read_csv('zoo.csv', names=['animal_name', 'hair',
262     'feathers', 'eggs', 'milk',
263     'airbone', 'aquatic',
264     'predator', 'toothed', 'backbone',
265     'breathes', 'venomous',
266     'fins', 'legs', 'tail', 'domestic', 'catsize',
267     'class'])
268     print(dataset.head(10))
269     # dropping the class column
270     dataset = dataset.drop('animal_name', axis=1)
271     print(dataset.head(10))
272     # splitting data
273     training_data = train_test_split(dataset)[0]
274     testing_data = train_test_split(dataset)[1]
275     # training the tree
276     tree = id3(training_data, training_data,
277     training_data.columns[:-1])
278     # printing the learnt tree in the form of a dictionary
279     pprint(tree)
280     # get test performance of the tree
281     test(testing_data, tree)
282

```

RESULTS: WEKA OUTPUT

Weka Explorer

Preprocess **Classify** Cluster Associate Select attributes Visualize CPython Scripting

Classifier

Choose **J48 -C 0.25 -M 2**

Test options

☐ Use training set
☐ Supplied test set Set...
☒ Cross-validation Folds **10**
☐ Percentage split % 66
 More options...

(Norm) class

Start Stop

Result list (right-click for options)

Classifier output

Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances	216	75.5245 %
Incorrectly Classified Instances	70	24.4755 %
Kappa statistic	0.2826	
Mean absolute error	0.3676	
Root mean squared error	0.4324	
Relative absolute error	87.8635 %	
Root relative squared error	94.6093 %	
Total Number of Instances	286	

=== Detailed Accuracy By Class ===

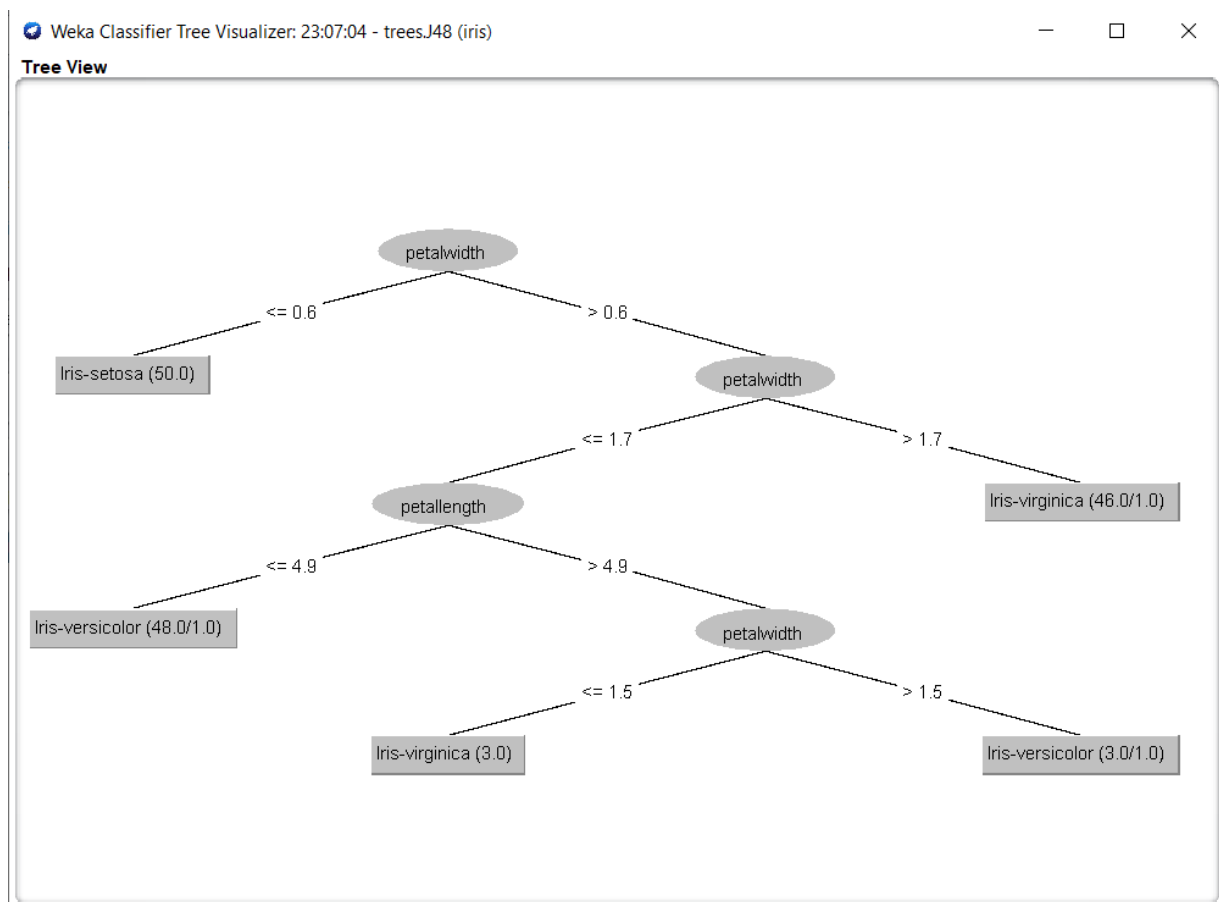
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Cl
	0.960	0.729	0.757	0.960	0.846	0.339	0.584	0.736	no
	0.271	0.040	0.742	0.271	0.397	0.339	0.584	0.436	re
Weighted Avg.	0.755	0.524	0.752	0.755	0.713	0.339	0.584	0.647	

=== Confusion Matrix ===

a	b	<-- classified as	
193	8	a = no-recurrence-events	
62	23	b = recurrence-events	

Status

OK Log x 0



PYTHON OUTPUT:

```

animal_name hair feathers eggs milk airborne aquatic ... venomous fins legs tail domestic catsize class
0 aardvark 1 0 0 1 0 0 ... 0 0 4 0 0 0 1 1
1 antelope 1 0 0 1 0 0 ... 0 0 4 1 0 1 1
2 bass 0 0 1 0 0 1 ... 0 1 0 1 0 0 4
3 bear 1 0 0 1 0 0 ... 0 0 4 0 0 1 1
4 boar 1 0 0 1 0 0 ... 0 0 4 1 0 1 1
5 buffalo 1 0 0 1 0 0 ... 0 0 4 1 0 1 1
6 calf 1 0 0 1 0 0 ... 0 0 4 1 1 1 1
7 carp 0 0 1 0 0 1 ... 0 1 0 1 1 0 4
8 catfish 0 0 1 0 0 1 ... 0 1 0 1 0 0 4
9 cavy 1 0 0 1 0 0 ... 0 0 4 0 1 0 1

[10 rows x 18 columns]
hair feathers eggs milk airborne aquatic predator ... venomous fins legs tail domestic catsize class
0 1 0 0 1 0 0 1 ... 0 0 4 0 0 1 1
1 1 0 0 1 0 0 0 ... 0 0 4 1 0 1 1
2 0 0 1 0 0 1 1 ... 0 1 0 1 0 0 4
3 1 0 0 1 0 0 1 ... 0 0 4 0 0 1 1
4 1 0 0 1 0 0 1 ... 0 0 4 1 0 1 1
5 1 0 0 1 0 0 0 ... 0 0 4 1 0 1 1
6 1 0 0 1 0 0 0 ... 0 0 4 1 1 1 1
7 0 0 1 0 0 1 0 ... 0 1 0 1 1 0 4
8 0 0 1 0 0 1 1 ... 0 1 0 1 0 0 4
9 1 0 0 1 0 0 0 ... 0 0 4 0 1 0 1

[10 rows x 17 columns]
{'legs': {0: {'fins': {0.0: {'toothed': {0.0: 7.0, 1.0: 3.0}},
1.0: {'eggs': {0.0: 1.0, 1.0: 4.0}}}},
2: {'hair': {0.0: 2.0, 1.0: 1.0}},
4: {'hair': {0.0: {'toothed': {0.0: 7.0, 1.0: 5.0}}, 1.0: 1.0}},
6: {'aquatic': {0.0: 6.0, 1.0: 7.0}},
8: 7.0}}
The prediction accuracy is: 85.71428571428571 %
```

FINDINGS AND LEARNINGS:

1. Decision tree is 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.
2. 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.

PROGRAM 7

AIM: Implement K-Means clustering algorithm on iris dataset using python.

INTRODUCTION AND 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 centers 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 centers obtained by k-means classifies new data into the existing clusters. This is known as nearest centroid classifier or Rocchio algorithm.

To process the learning data, the K-means algorithm in data mining starts with a 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.

K-MEANS PSEUDOCODE

1. Initialize k means with random values
2. For a given number of iterations:
3. Iterate through items:
4. Find the mean closest to the item
5. Assign item to mean
6. Update mean

hub.gke2.mybinder.org/user/python-ipynb7/notebooks/binder/Untitled.ipynb?kernel_name=python3

Jupyter Untitled Last Checkpoint: 7 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Memory: 260 / 2048 MB

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets

In [2]: iris=datasets.load_iris()
print(iris)
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
                [4.9, 3. , 1.4, 0.2],
                [4.7, 3.2, 1.3, 0.2],
                [4.6, 3.1, 1.5, 0.2],
                [5. , 3.6, 1.4, 0.2],
                [5.4, 3.9, 1.7, 0.4],
                [4.6, 3.4, 1.4, 0.3],
                [5. , 3.4, 1.5, 0.2],
                [4.4, 2.9, 1.4, 0.2],
                [4.9, 3.1, 1.5, 0.1],
                [5.4, 3.7, 1.5, 0.2],
                [4.8, 3.4, 1.6, 0.2],
                [4.8, 3. , 1.4, 0.1],
                [4.3, 3. , 1.1, 0.1],
                [5.8, 4. , 1.2, 0.2],
                [5.7, 4.4, 1.5, 0.4],
                [5.4, 3.9, 1.3, 0.4],
                [5.1, 3.5, 1.4, 0.3],
                [5.7, 3.8, 1.7, 0.3],
                [5.1, 3.8, 1.6, 0.1]])
```

In []:

33

hub.gke2.mybinder.org/user/ipython-ipython-in-depth-d0q6hby7/notebooks/binder/Untitled.ipynb?kernel_name=python3

Jupyter Untitled Last Checkpoint: 14 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Run Download GitHub Binder Memory: 208 / 2048 MB

```

...
145 6.7 3.0 2
146 6.3 2.5 2
147 6.5 3.0 2
148 6.2 3.4 2
149 5.9 3.0 2
...
150 rows x 3 columns

In [5]: centroids={}
for i in range(3):
    result_list=[]
    result_list.append(df.loc[df['cluster']==i]['x'].mean())
    result_list.append(df.loc[df['cluster']==i]['y'].mean())
    centroids[i]=result_list

In [6]: centroids
Out[6]: {0: [5.006, 3.428],
1: [5.936, 2.7700000000000005],
2: [6.587999999999998, 2.974]}

In [ ]:

```

07:19 PM 03-12-2020

hub.gke2.mybinder.org/user/ipython-ipython-in-depth-d0q6hby7/notebooks/binder/Untitled.ipynb?kernel_name=python3

Jupyter Untitled Last Checkpoint: 17 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Run Download GitHub Binder Memory: 212 / 2048 MB

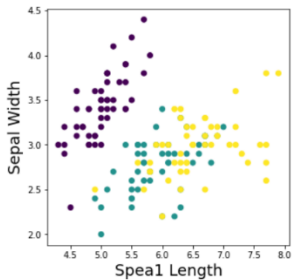
```

Out[6]: {0: [5.006, 3.428],
1: [5.936, 2.7700000000000005],
2: [6.587999999999998, 2.974]}

In [7]: fig=plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], c=iris.target)
plt.xlabel('Sepal Length', fontsize=18)
plt.ylabel('Sepal Width', fontsize=18)

Out[7]: Text(0, 0.5, 'Sepal Width')

```



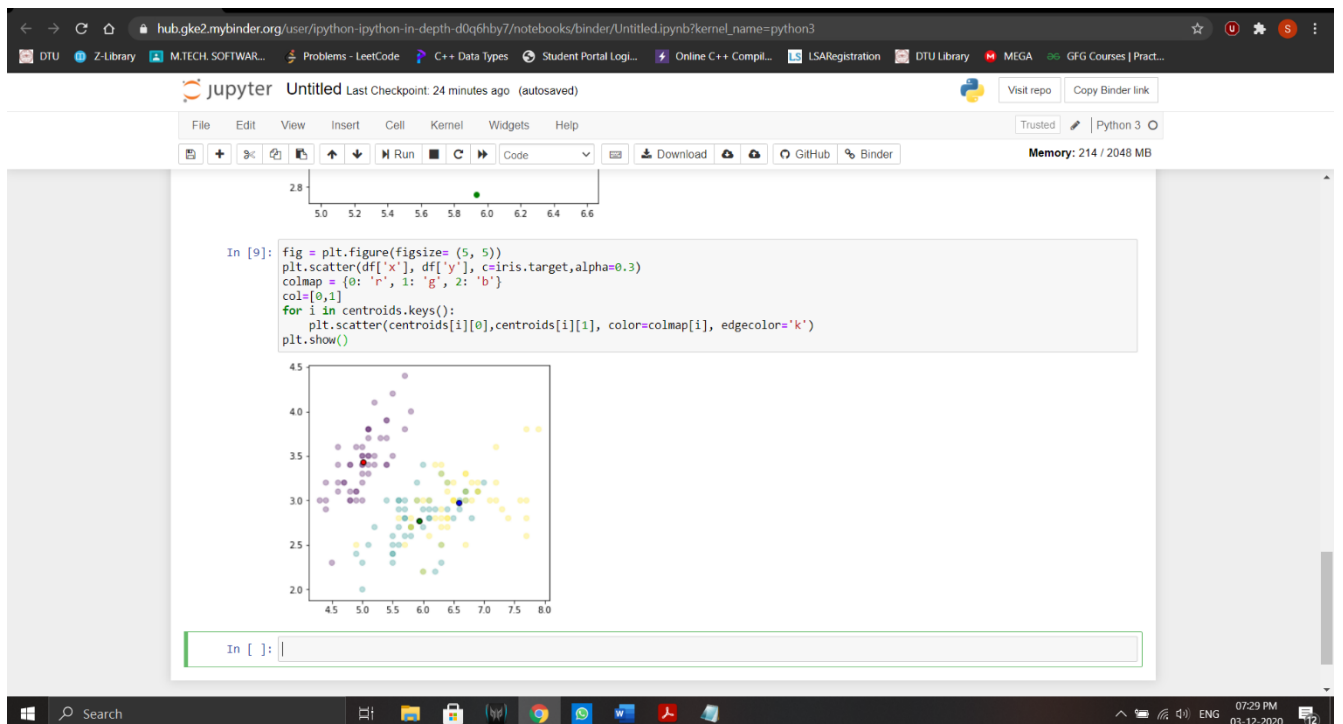
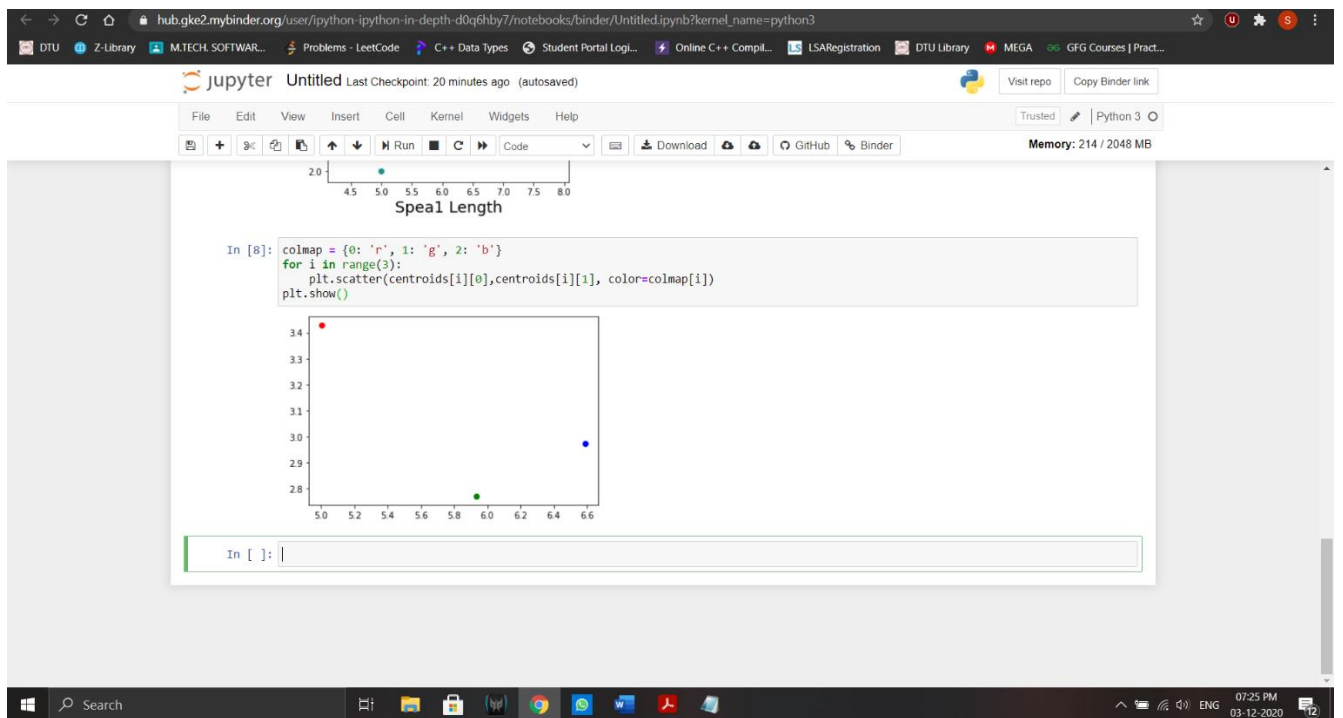
The scatter plot shows the relationship between Sepal Length (x-axis, ranging from 4.5 to 8.0) and Sepal Width (y-axis, ranging from 2.0 to 4.5). The data points are colored based on the iris target variable, showing three distinct clusters: purple (target 0), yellow (target 1), and teal (target 2). The clusters are well-separated, indicating good clustering performance.

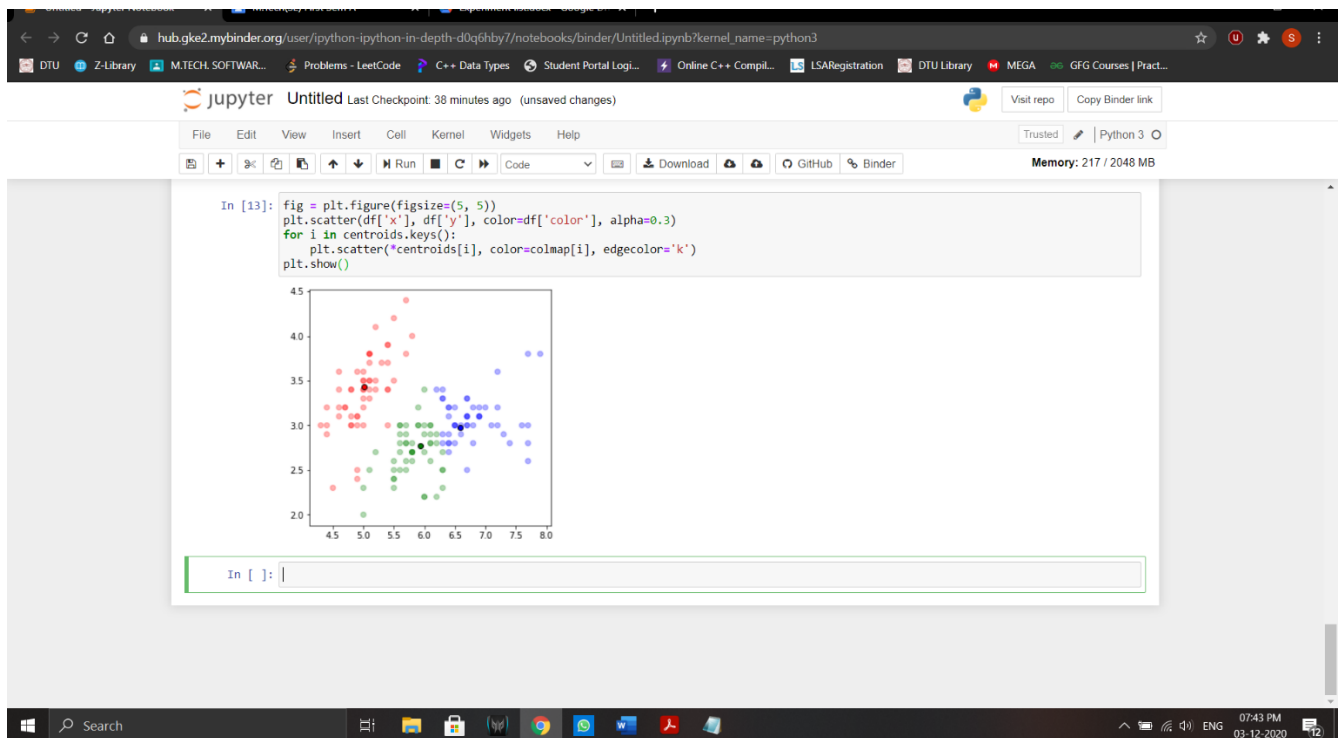
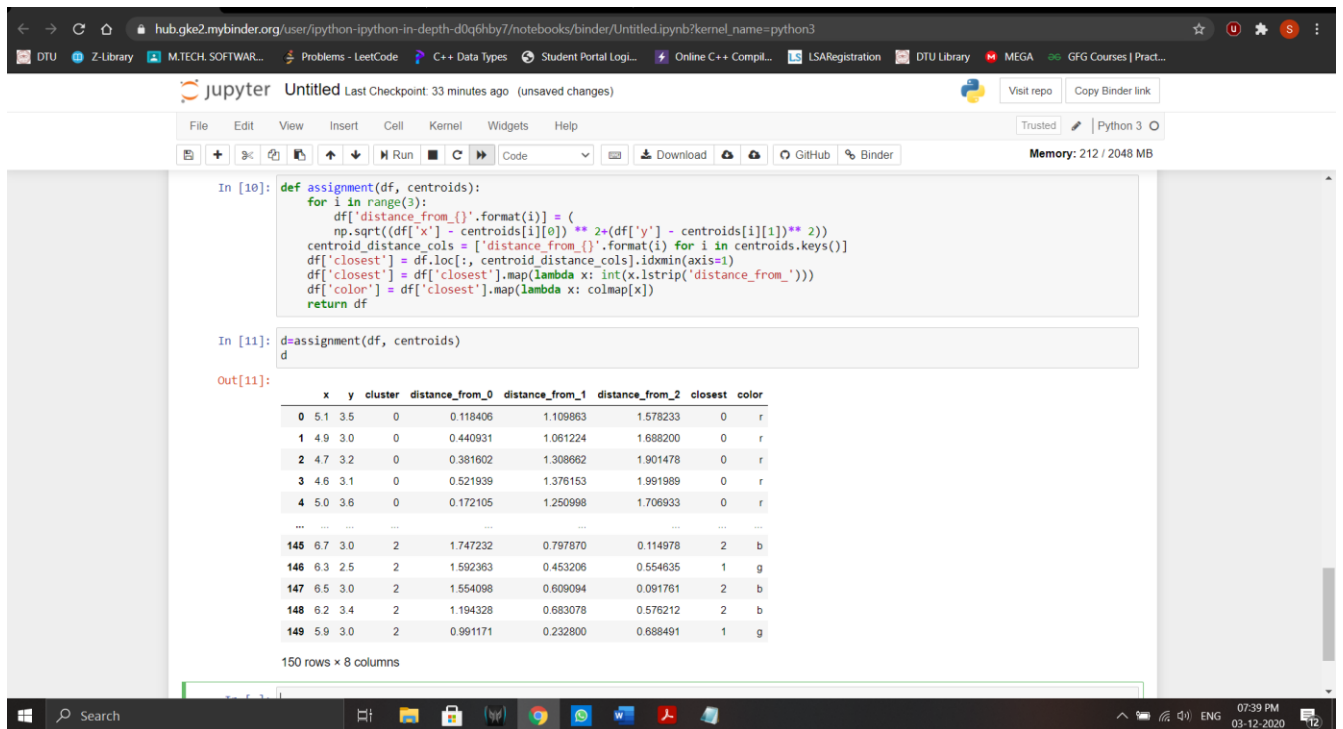
```

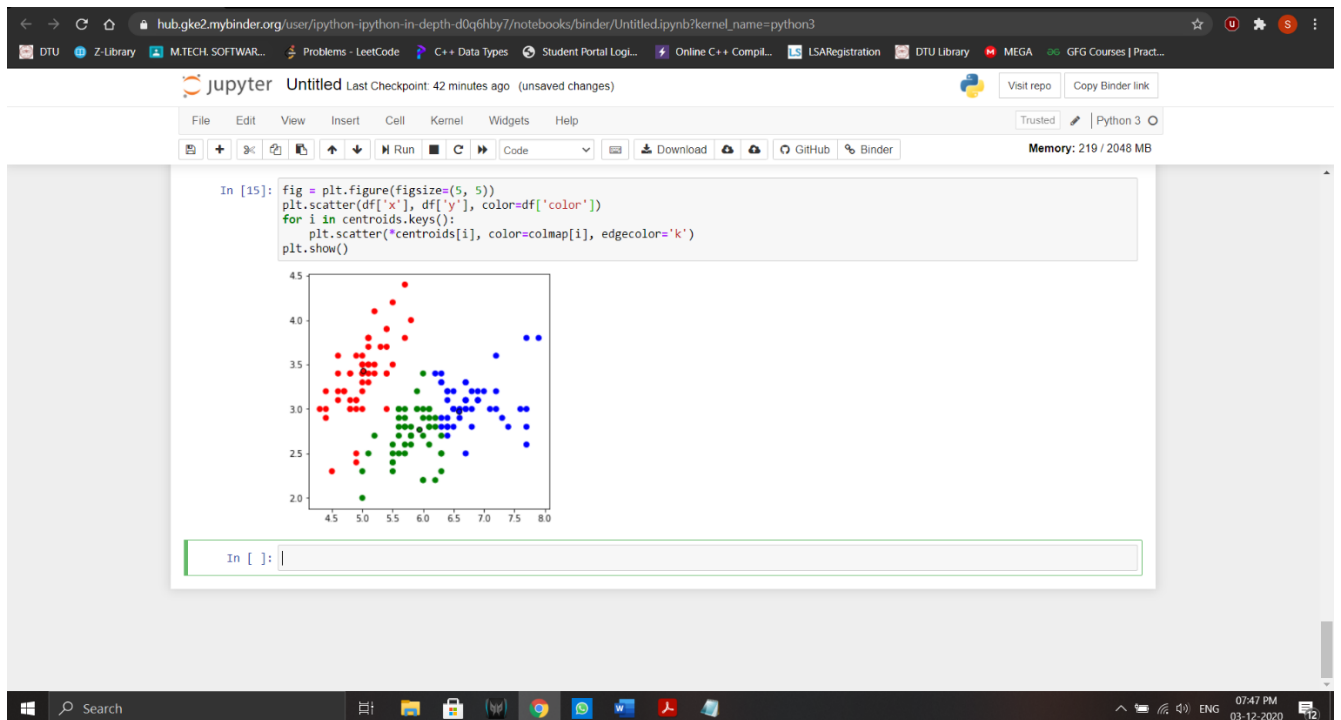
In [ ]:

```

07:22 PM 03-12-2020







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.

PROGRAM 8

AIM: Perform association rule mining using Apriori algorithm on any sample dataset.

INTRODUCTION AND THEORY:

Association rules are if-then statements that help to show the probability of relationships between data items within large data sets in various types of databases. Association rule mining has a number of applications and is widely used to help discover sales correlations in transactional data or in medical data sets.

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.

Association rules are created by searching data for frequent if-then patterns and using the criteria support and confidence to identify the most important relationships. Support is an indication of how frequently the items appear in the data. Confidence indicates the number of times the if-then statements are found true. A third metric, called lift, can be used to compare confidence with expected confidence.

Association rules are calculated from itemsets, which are made up of two or more items. If rules are built from analyzing all the possible itemsets, there could be so many rules that the rules hold little meaning. With that, association rules are typically created from rules well-represented in data.

Measure 1: 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, $\text{supp}(X)$ is the proportion of transaction in the database in which the item X appears. It signifies the popularity of an itemset.

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

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.

Measure 2: Confidence. This says how likely item Y is purchased when item X is purchased, expressed as $\{X \rightarrow Y\}$. 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:

$$conf(X \rightarrow Y) = supp(X \cup Y) / supp(X)$$

It signifies the likelihood of item Y being purchased when item X is purchased.

This implies that for 75% of the transactions containing onion and potatoes, the rule is correct. It can also be interpreted as the conditional probability $P(Y|X)$, i.e, the probability of finding the itemset Y in transactions given the transaction already contains X.

It can give some important insights, but it also has a major drawback. It only takes into account the popularity of the itemset X and not the popularity of Y. If Y is equally popular as X then there will be a higher probability that a transaction containing X will also contain Y thus increasing the confidence. To overcome this drawback there is another measure called lift.

Measure 3: 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:

$$lift(X \rightarrow Y) = supp(X \cup Y) / (supp(X) * supp(Y))$$

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.

APPLICATIONS:

MARKET BASKET ANALYSIS:

This is the most typical example of association mining. Data is collected using barcode scanners in most supermarkets. This database, known as the “market basket” database, consists of a large number of records on past transactions. A single record lists all the items bought by a customer in one sale. 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.

MEDICAL DIAGNOSIS:

Association rules in medical diagnosis can be useful for assisting physicians for curing patients. Diagnosis is not an easy process and has a scope of errors which may result in unreliable end-results. Using relational association rule mining, we can identify the probability of the occurrence of an illness concerning various factors and symptoms. Further, using learning techniques, this interface can be extended by adding new symptoms and defining relationships between the new signs and the corresponding diseases.

CENSUS DATA:

Every government has tones of census data. This data can be used to plan efficient public services (education, health, transport) as well as help public businesses (for setting up new factories, shopping malls, and even marketing particular products). 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.

PROTEIN SEQUENCE:

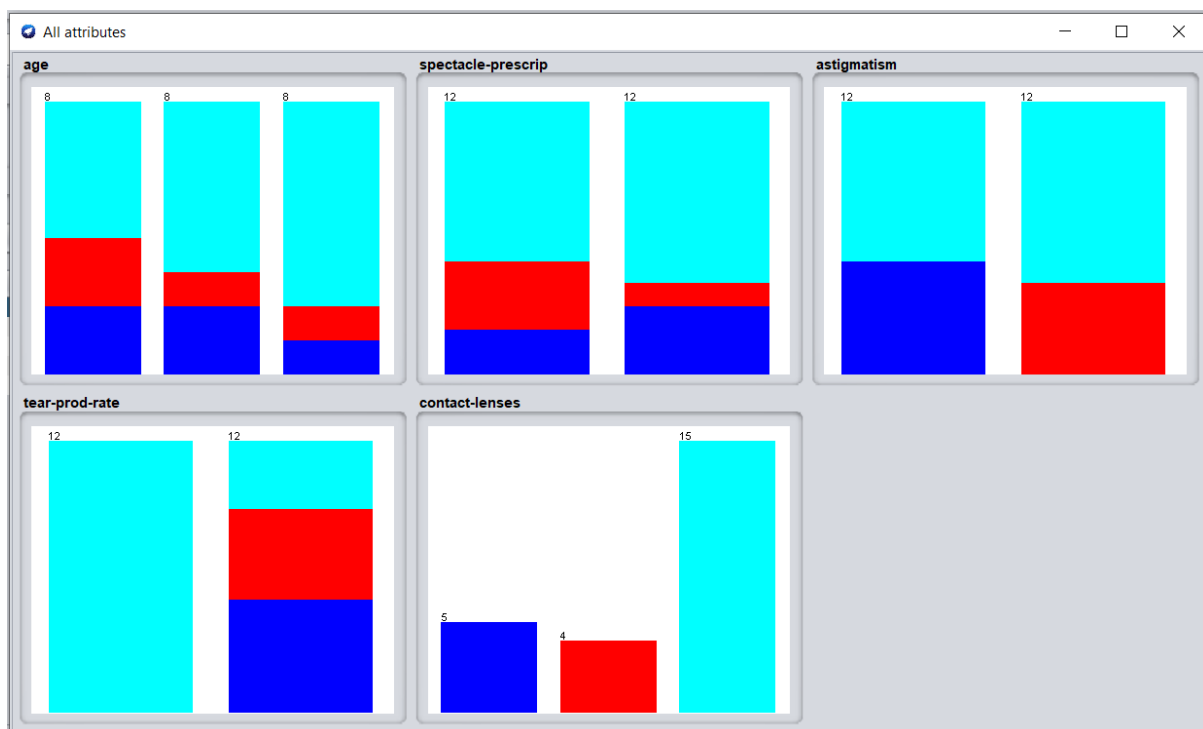
Proteins are sequences made up of twenty types of amino acids. Each protein bears a unique 3D structure which depends on the sequence of these amino acids. A slight change in the sequence can cause a change in structure which might change the functioning of the protein. This dependency of the protein functioning on its amino acid sequence has been a subject of great research. Earlier it was thought that these sequences are random, but now it's believed that they aren't. Knowledge and understanding of these association rules will come in extremely helpful during the synthesis of artificial proteins.

PROCEDURE:

1. Open the data file in Weka Explorer. It is presumed that the required data fields have been discretized. In this example I have taken contact-lenses data file.
2. Clicking on the associate tab will bring up the interface for association rule algorithm.
3. We will use Apriori algorithm. This is the default algorithm.
4. In order to change the parameters for the run (example support, confidence etc) we click on the text box immediately to the right of the choose button.

RESULTS AND OUTPUT:

DATASET CHOSEN



OUTPUT

```
=== Run information ===

Scheme:      weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:    contact-lenses
Instances:    24
Attributes:   5
              age
              spectacle-prescrip
              astigmatism
              tear-prod-rate
              contact-lenses

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

Apriori
=====

Minimum support: 0.2 (5 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 16

Generated sets of large itemsets:

Size of set of large itemsets L(1): 11
Size of set of large itemsets L(2): 21
Size of set of large itemsets L(3): 6

Best rules found:

1. tear-prod-rate=reduced 12 ==> contact-lenses=none 12   <conf:(1)> lift:(1.6) lev:(0.19) [4] conv:(4.5)
2. spectacle-prescrip=myope tear-prod-rate=reduced 6 ==> contact-lenses=none 6   <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
3. spectacle-prescrip=hypermetrope tear-prod-rate=reduced 6 ==> contact-lenses=none 6   <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
```

```
tear-prod-rate
contact-lenses

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

Apriori
=====

Minimum support: 0.2 (5 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 16

Generated sets of large itemsets:

Size of set of large itemsets L(1): 11
Size of set of large itemsets L(2): 21
Size of set of large itemsets L(3): 6

Best rules found:

1. tear-prod-rate=reduced 12 ==> contact-lenses=none 12   <conf:(1)> lift:(1.6) lev:(0.19) [4] conv:(4.5)
2. spectacle-prescrip=myope tear-prod-rate=reduced 6 ==> contact-lenses=none 6   <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
3. spectacle-prescrip=hypermetrope tear-prod-rate=reduced 6 ==> contact-lenses=none 6   <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
4. astigmatism=no tear-prod-rate=reduced 6 ==> contact-lenses=none 6   <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
5. astigmatism=yes tear-prod-rate=reduced 6 ==> contact-lenses=none 6   <conf:(1)> lift:(1.6) lev:(0.09) [2] conv:(2.25)
6. contact-lenses=soft 5 ==> astigmatism=no 5   <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
7. contact-lenses=soft 5 ==> tear-prod-rate=normal 5   <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
8. tear-prod-rate=normal contact-lenses=soft 5 ==> astigmatism=no 5   <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
9. astigmatism=no contact-lenses=soft 5 ==> tear-prod-rate=normal 5   <conf:(1)> lift:(2) lev:(0.1) [2] conv:(2.5)
10. contact-lenses=soft 5 ==> astigmatism=no tear-prod-rate=normal 5   <conf:(1)> lift:(4) lev:(0.16) [3] conv:(3.75)
```

FINDINGS AND LEARNINGS:

1. We learned about association rule mining and familiarized ourselves with the contact-lenses.arff dataset.
2. We then used weka to implement association rule mining on the dataset.

PROGRAM 9

AIM: Implement K-Nearest Neighbor (KNN) in python language.

INTRODUCTION AND 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 OF KNN:

- K-NN is pretty intuitive and simple: K-NN algorithm is very simple to understand and equally easy to implement. To classify the new data point K-NN algorithm reads through whole dataset to find out K nearest neighbors.
- K-NN has no assumptions: K-NN is a non-parametric algorithm which means there are no assumptions to be met to implement K-NN. Parametric models like linear regression have lots of assumptions to be met by data before it can be implemented which is not the case with K-NN.
- No Training Step: K-NN does not explicitly build any model, it simply tags the new data entry based on learning from historical data. New data entry would be tagged with majority class in the nearest neighbor.
- 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. It allows the algorithm to respond quickly to changes in the input during real-time use.
- Very easy to implement for multi-class problem: 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 choose from: K-NN algorithm gives user the flexibility to choose distance while building K-NN model.
 - Hamming Distance
 - Manhattan Distance
 - Euclidean Distance

WEAKNESS OF KNN:

- K-NN slow algorithm: K-NN might be very easy to implement but as dataset grows efficiency or speed of algorithm declines very fast.
- Curse of Dimensionality: KNN works well with small number of input variables but as the numbers of variables grow K-NN algorithm struggles to predict the output of new data point.
- 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 means 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 be 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. This might result in getting the less common class B wrongly classified.
- 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.

CODE:

```

1  # importing libraries
2
3  import pandas as pd
4  import numpy as np
5  import math
6  import operator
7
8
9  def get_train_test_split(dataset, split_ratio=0.8):
10     datalen = len(dataset)
11     shuffled = dataset.iloc[np.random.permutation(len(dataset))]
12     train_data = shuffled[:int(split_ratio * datalen)]
13     test_data = shuffled[int(split_ratio * datalen):]
14     return train_data, test_data
15
16 def euclidean_dist(p1, p2):
17     dist = 0.0
18     for i in range(len(p1)-1):
19         dist += pow((float(p1[i])-float(p2[i])),2)
20     dist = math.sqrt(dist)
21     return dist
22
23 def knn(training_dataset, test_value, k):
24     distances = {}

```

```

25     sort = {}
26
27     # length = test_value.shape[1]
28
29     # Calculating euclidean distance between each row of training
30 data and test data
31     for x in range(len(training_dataset)):
32         dist = euclidean_dist(test_value, training_dataset.iloc[x])
33
34         distances[x] = dist
35     # Sorting them on the basis of distance
36     sorted_d = sorted(distances.items(), key=operator.itemgetter(1))
37
38     neighbors = []
39
40     # Extracting top k neighbors
41     for x in range(k):
42         neighbors.append(sorted_d[x][0])
43     class_votes = {}
44
45     # Calculating the most freq class in the neighbors
46     for x in range(len(neighbors)):
47         response = training_dataset.iloc[neighbors[x]][-1]
48
49         if response in class_votes:
50             class_votes[response] += 1
51         else:
52             class_votes[response] = 1
53
54     sorted_votes = sorted(class_votes.items(),
55 key=operator.itemgetter(1), reverse=True)
56     return sorted_votes[0][0], neighbors
57
58 def cal_accuracy(data, test_data, K=5):
59     correct = 0
60     total = len(test_data)
61
62     for i in range(total):
63         pred_class, neigh = knn(data, test_data.iloc[i], K)
64         print("predicted class: {:16} | actual class:
65 {:16}".format(pred_class, test_data.iloc[i]['Name']))
66         if pred_class == test_data.iloc[i]['Name']:
67             correct += 1
68         else:
69             print("\t\t\tincorrect prediction", neigh)
70
71     return (correct/total) * 100
72
73 if __name__ == '__main__':
74
75     dataset = pd.read_csv('iris.csv')
76     print('first 5 rows of the dataset')
77     print(dataset.head(5))
78     train_data, test_data = get_train_test_split(dataset)
79     print('first 5 rows of the train set')
80     print(train_data.head(5))
81     print('first 5 rows of the test set')

```

```

82 print(test_data.head(5))
83 print(cal_accuracy(train_data, test_data, 5))
84

```

RESULTS AND OUTPUT:

```

first 5 rows of the dataset
  SepalLength SepalWidth PetalLength PetalWidth      Name
0          5.1         3.5         1.4         0.2  Iris-setosa
1          4.9         3.0         1.4         0.2  Iris-setosa
2          4.7         3.2         1.3         0.2  Iris-setosa
3          4.6         3.1         1.5         0.2  Iris-setosa
4          5.0         3.6         1.4         0.2  Iris-setosa

first 5 rows of the train set
  SepalLength SepalWidth PetalLength PetalWidth      Name
81          5.5         2.4         3.7         1.0  Iris-versicolor
33          5.5         4.2         1.4         0.2  Iris-setosa
110         6.5         3.2         5.1         2.0  Iris-virginica
22          4.6         3.6         1.0         0.2  Iris-setosa
116         6.5         3.0         5.5         1.8  Iris-virginica

first 5 rows of the test set
  SepalLength SepalWidth PetalLength PetalWidth      Name
72          6.3         2.5         4.9         1.5  Iris-versicolor
73          6.1         2.8         4.7         1.2  Iris-versicolor
111         6.4         2.7         5.3         1.9  Iris-virginica
4          5.0         3.6         1.4         0.2  Iris-setosa
83          6.0         2.7         5.1         1.6  Iris-versicolor

predicted class: Iris-virginica | actual class: Iris-versicolor
incorrect prediction [36, 44, 133, 27, 112]
predicted class: Iris-versicolor | actual class: Iris-versicolor
predicted class: Iris-virginica | actual class: Iris-virginica
predicted class: Iris-setosa | actual class: Iris-setosa
predicted class: Iris-virginica | actual class: Iris-versicolor
incorrect prediction [9, 123, 50, 133, 117]
predicted class: Iris-virginica | actual class: Iris-virginica
predicted class: Iris-setosa | actual class: Iris-setosa
predicted class: Iris-virginica | actual class: Iris-virginica
predicted class: Iris-virginica | actual class: Iris-virginica
predicted class: Iris-versicolor | actual class: Iris-versicolor
predicted class: Iris-virginica | actual class: Iris-virginica
predicted class: Iris-setosa | actual class: Iris-setosa
predicted class: Iris-virginica | actual class: Iris-virginica
predicted class: Iris-setosa | actual class: Iris-setosa
predicted class: Iris-virginica | actual class: Iris-versicolor
incorrect prediction [9, 50, 84, 117, 112]

80.0

```

FINDINGS AND LEARNINGS:

1. We have Implemented K-nearest-neighbors algorithm in python 3.
2. We have learned the nuances of the KNN-algorithm.
3. We have learnt about the applications, strengths and weaknesses of KNN-algorithm.

PROGRAM 10

AIM: Write a program in python language to implement model evaluation (Cross validation and generate confusion matrix).

INTRODUCTION AND THEORY:

CROSS VALIDATION

Cross-validation, sometimes called rotation estimation, or out-of-sample testing is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (called the validation dataset or testing set). The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem).

The general procedure is as follows (For K Fold Cross Validation):

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
4. Take the group as a hold out or test data set
5. Take the remaining groups as a training data set
6. Fit a model on the training set and evaluate it on the test set
7. Retain the evaluation score and discard the model
8. Summarize the skill of the model using the sample of model evaluation scores.

CONFUSION MATRIX

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).

It is a special kind of contingency table, with two dimensions ("actual" and "predicted"), and identical sets of "classes" in both dimensions (each combination of dimension and class is a variable in the contingency table).

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

CODE:

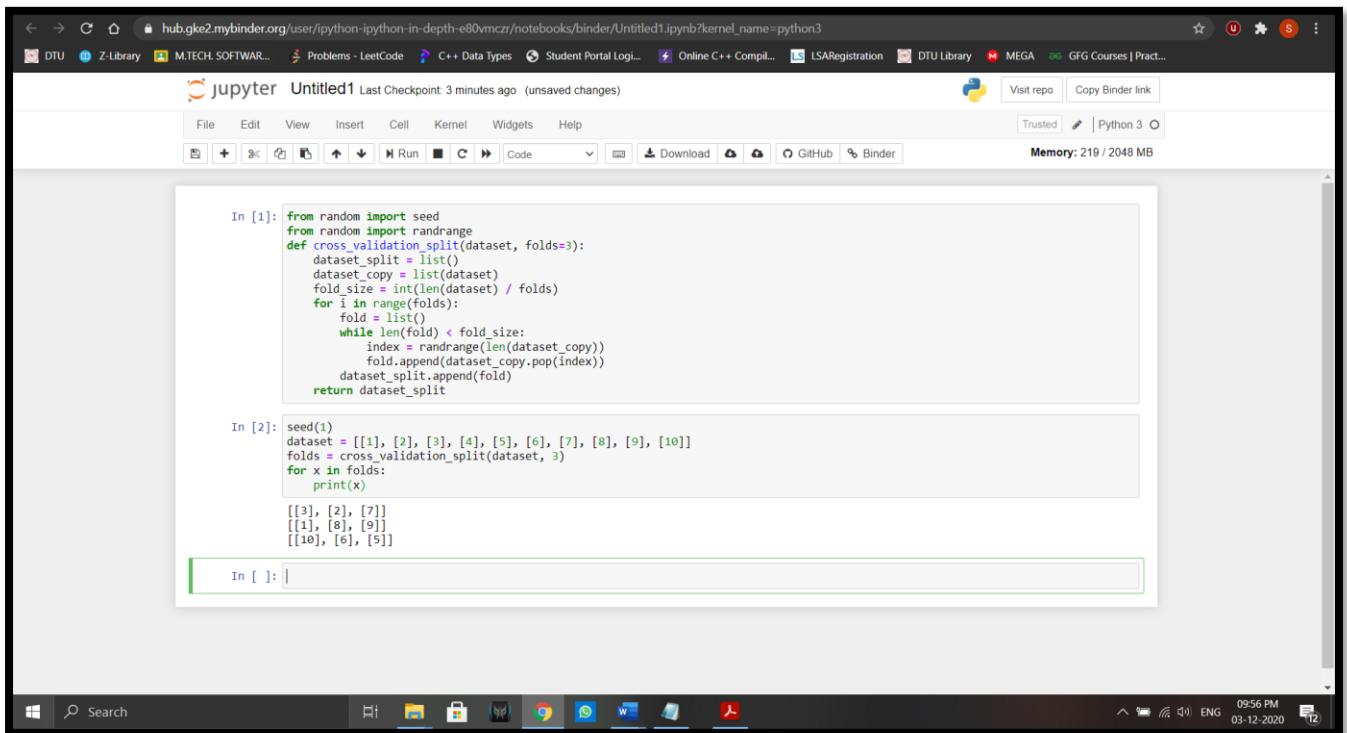
```

1  from random import seed
2  from random import randrange
3
4
5  def cross_validation_split(dataset, folds=3):
6      dataset_split = list()
7      dataset_copy = list(dataset)
8      fold_size = int(len(dataset) / folds)
9      for i in range(folds):
10         fold = list()
11         while len(fold) < fold_size:
12             index = randrange(len(dataset_copy))
13             fold.append(dataset_copy.pop(index))
14         dataset_split.append(fold)
15     return dataset_split
16
17     seed(1)
18     dataset = [[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]]
19     folds = cross_validation_split(dataset, 3)
20     for x in folds:
21         print(x)
22
23
24     def confusionmatrix(actual, predicted, normalize=False):
25         unique = sorted(set(actual))
26         matrix = [[0 for _ in unique] for _ in unique]
27         imap = {key: i for i, key in enumerate(unique)}
28
29         for p, a in zip(predicted, actual):
30             matrix[imap[p]][imap[a]] += 1
31
32         if normalize:
33             sigma = sum([sum(matrix[imap[i]]) for i in unique])
34             matrix = [row for row in map(lambda i: list(map(lambda j: j /
35 sigma, i)), matrix)]
36         return matrix
37
38
39     cm = confusionmatrix(
40         [1, 1, 2, 0, 1, 1, 2, 0, 0, 1], # actual
41         [0, 1, 1, 0, 2, 1, 2, 2, 0, 2] # predicted
42     )
43     print('actual : ', [1, 1, 2, 0, 1, 1, 2, 0, 0, 1])
44     print('predicted: ', [0, 1, 1, 0, 2, 1, 2, 2, 0, 2])
45     for x in cm:
46         print(x)

```

RESULTS AND OUTPUT:

K-FOLD CROSS VALIDATION



A screenshot of a Jupyter Notebook interface. The browser address bar shows a URL from hub.gke2.mybinder.org. The notebook title is 'Untitled1' and it indicates 'Last Checkpoint: 3 minutes ago (unsaved changes)'. The top menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with icons for file operations, running, and saving. The main area contains two code cells. The first cell, 'In [1]:', defines a function 'cross_validation_split' that takes a dataset and the number of folds as input. It uses random seed, randrange, and list operations to create a list of folds, each containing a subset of the dataset. The second cell, 'In [2]:', calls the function with a dataset of 10 elements and 3 folds, and prints the resulting folds. The output shows three folds: [[3], [2], [7]], [[1], [8], [9]], and [[10], [6], [5]]. The bottom status bar shows 'Memory: 219 / 2048 MB'.

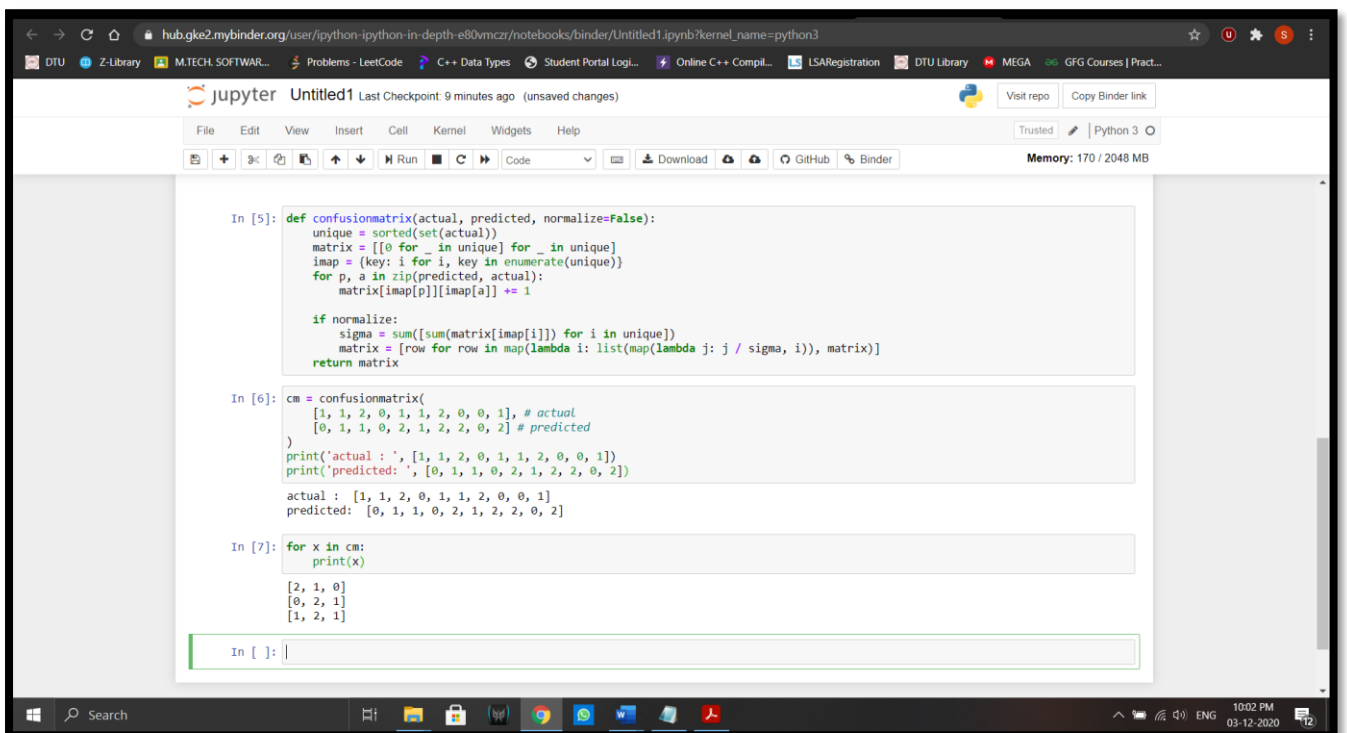
```
In [1]: from random import seed
from random import randrange
def cross_validation_split(dataset, folds=3):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / folds)
    for i in range(folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split

In [2]: seed(1)
dataset = [[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]]
folds = cross_validation_split(dataset, 3)
for x in folds:
    print(x)

[[3], [2], [7]]
[[1], [8], [9]]
[[10], [6], [5]]

In [ ]: |
```

CONFUSION MATRIX



A screenshot of a Jupyter Notebook interface. The browser address bar shows a URL from hub.gke2.mybinder.org. The notebook title is 'Untitled1' and it indicates 'Last Checkpoint: 9 minutes ago (unsaved changes)'. The top menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with icons for file operations, running, and saving. The main area contains three code cells. The first cell, 'In [5]:', defines a function 'confusionmatrix' that takes actual, predicted, and a normalize flag as input. It creates a matrix of zeros, increments the values at the intersection of predicted and actual classes, and then normalizes the matrix by row. The second cell, 'In [6]:', calls the function with actual values [1, 1, 2, 0, 1, 1, 2, 0, 0, 1] and predicted values [0, 1, 1, 0, 2, 1, 2, 2, 0, 2]. It prints the actual and predicted arrays, and the resulting confusion matrix. The third cell, 'In [7]:', iterates over the confusion matrix and prints each element. The output shows the confusion matrix as a 2x2 grid: [[2, 1, 0], [0, 2, 1], [1, 2, 1]]. The bottom status bar shows 'Memory: 170 / 2048 MB'.

```
In [5]: def confusionmatrix(actual, predicted, normalize=False):
        unique = sorted(set(actual))
        matrix = [[0 for _ in unique] for _ in unique]
        imap = {key: i for i, key in enumerate(unique)}
        for p, a in zip(predicted, actual):
            matrix[imap[p]][imap[a]] += 1

        if normalize:
            sigma = sum([sum(matrix[imap[i]]) for i in unique])
            matrix = [row for row in map(lambda i: list(map(lambda j: j / sigma, i)), matrix)]
        return matrix

In [6]: cm = confusionmatrix(
        [1, 1, 2, 0, 1, 1, 2, 0, 0, 1], # actual
        [0, 1, 1, 0, 2, 1, 2, 2, 0, 2] # predicted
    )
print('actual : ', [1, 1, 2, 0, 1, 1, 2, 0, 0, 1])
print('predicted: ', [0, 1, 1, 0, 2, 1, 2, 2, 0, 2])

actual : [1, 1, 2, 0, 1, 1, 2, 0, 0, 1]
predicted: [0, 1, 1, 0, 2, 1, 2, 2, 0, 2]

In [7]: for x in cm:
        print(x)

[2, 1, 0]
[0, 2, 1]
[1, 2, 1]

In [ ]: |
```


FINDINGS AND LEARNINGS:

1. What is cross validation and confusion matrix.
2. Cross validation helps correct over fitting.
3. Confusion Matrix helps identify skew and misclassification
4. How to code cross validation and confusion matrix in python.