Practical No. 05

Aim: Organize and Prepare the data needed for data mining using pre-processing techniques.

Theory: Steps to open Weka-

1. From the Start menu, in the All Programs option, select Weka 3.6.9.
2. The below Weka GUI Chooser window appears. In this window, click on the Explorer option to open the Weka Explorer.

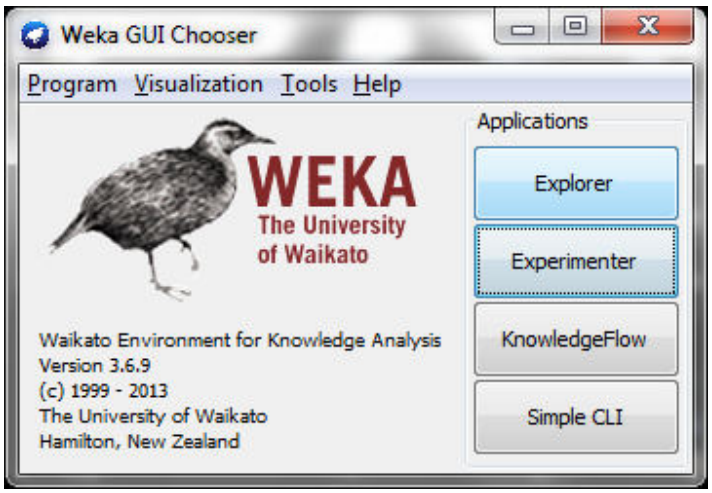


Figure 10. Weka GUI Chooser window

3. The Weka Explorer window shown in the Figureure below appears. In this window there are various options namely, Preprocessing, Classify, Cluster, Associate, Select attributes and visualize.

Steps to Organize and Prepare the data needed for data mining using pre-processing techniques-

Step 1: In order to perform preprocessing, click on the Preprocess Tab.

Step 2: In this tab pane, we have to first click on the Open file button to select the Data that is to be preprocessed.

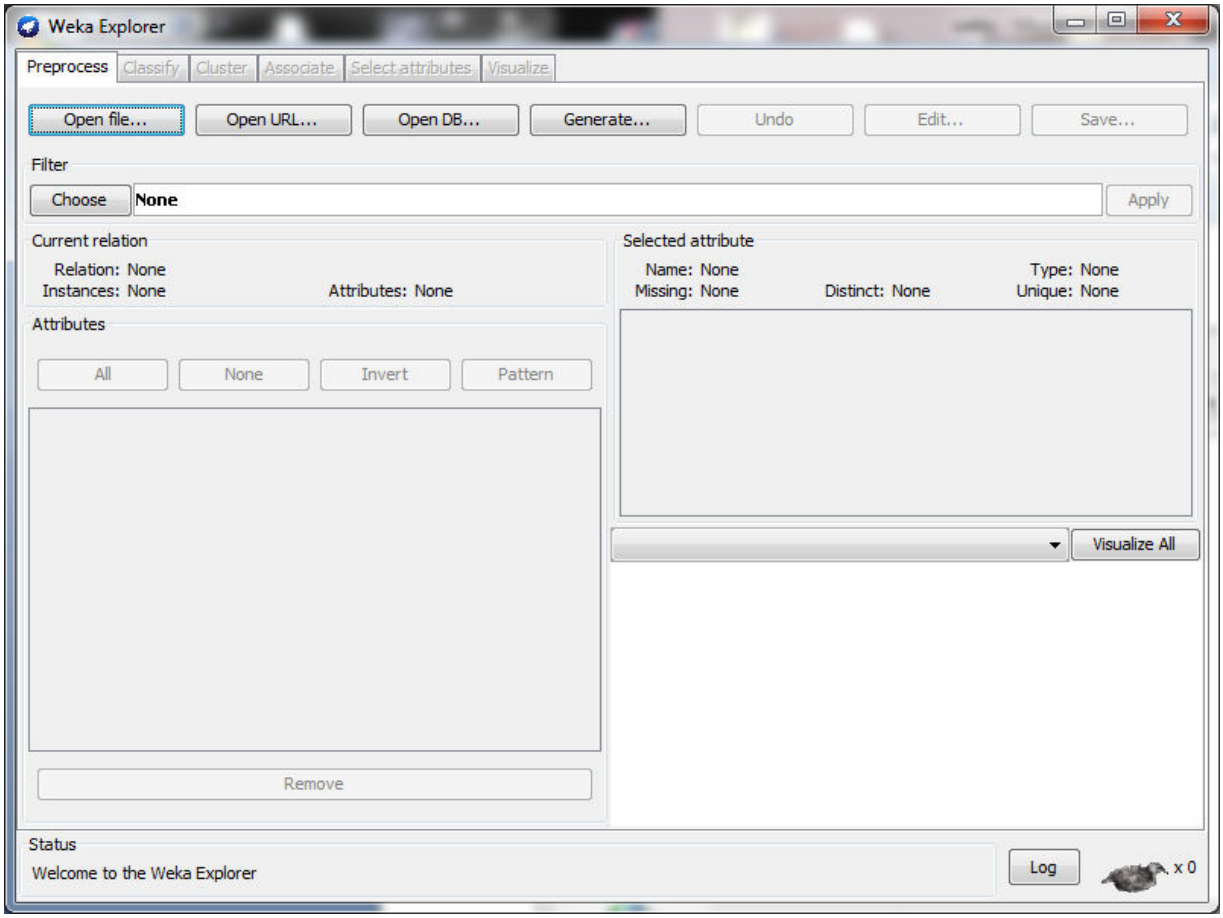


Figure 10.1 Weka explorer window- Preprocessing.

Step 3: An Open dialog box appears, in which we have to go to the following path:

C:\Program Files (x86)\Weka-3-6\data

And select the desired data that you want to Preprocess. In the Figure 10.2, Weather.numeric data is selected. And then click on Open.

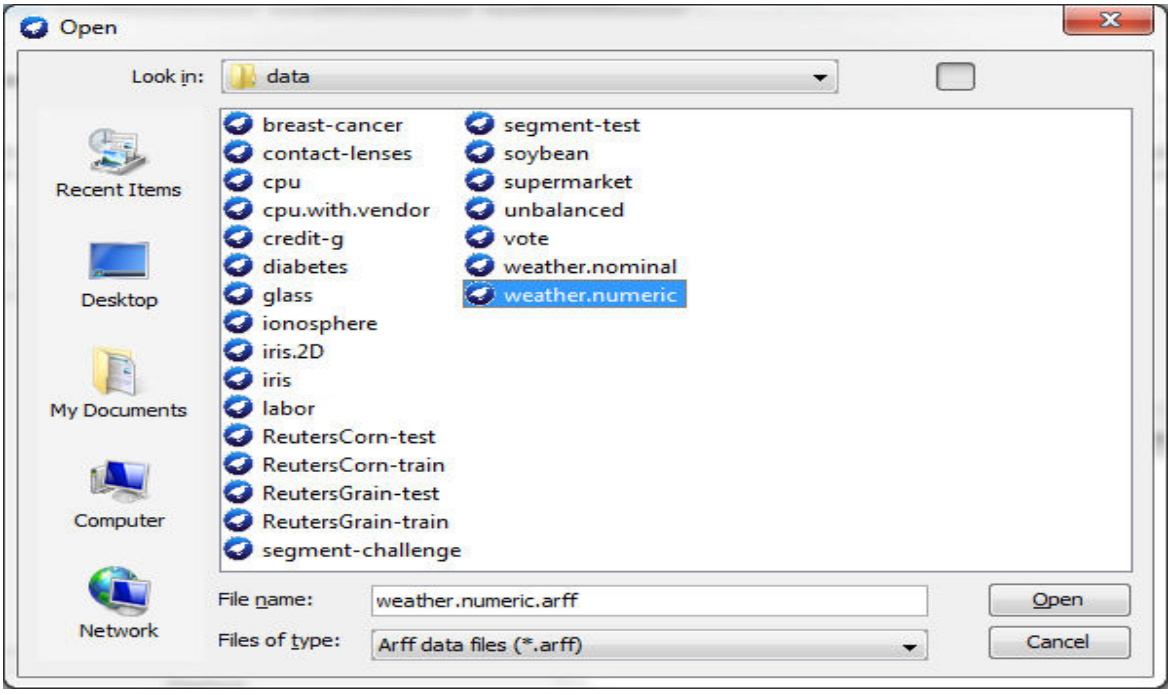


Figure 10.2 Open dialog box

Step 4: After selecting the data to be preprocessed is selected, click on the Choose option in the Filter panel in order to select the attribute. As shown in the Figure 10.3, expand the filters option in the weka tree view then further expand the supervised option and select the AddClassification attribute.

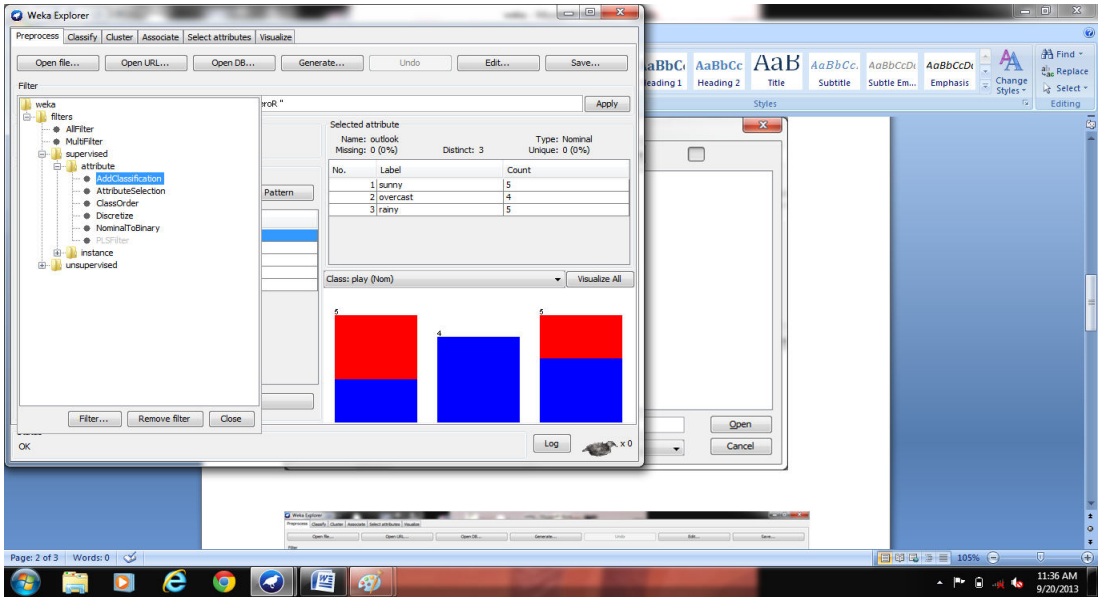


Figure 10.3 Filter panel - Selecting Filter attribute

Step 5: Now click on the Apply button. We will get to see the graph changes in the right end of the window.

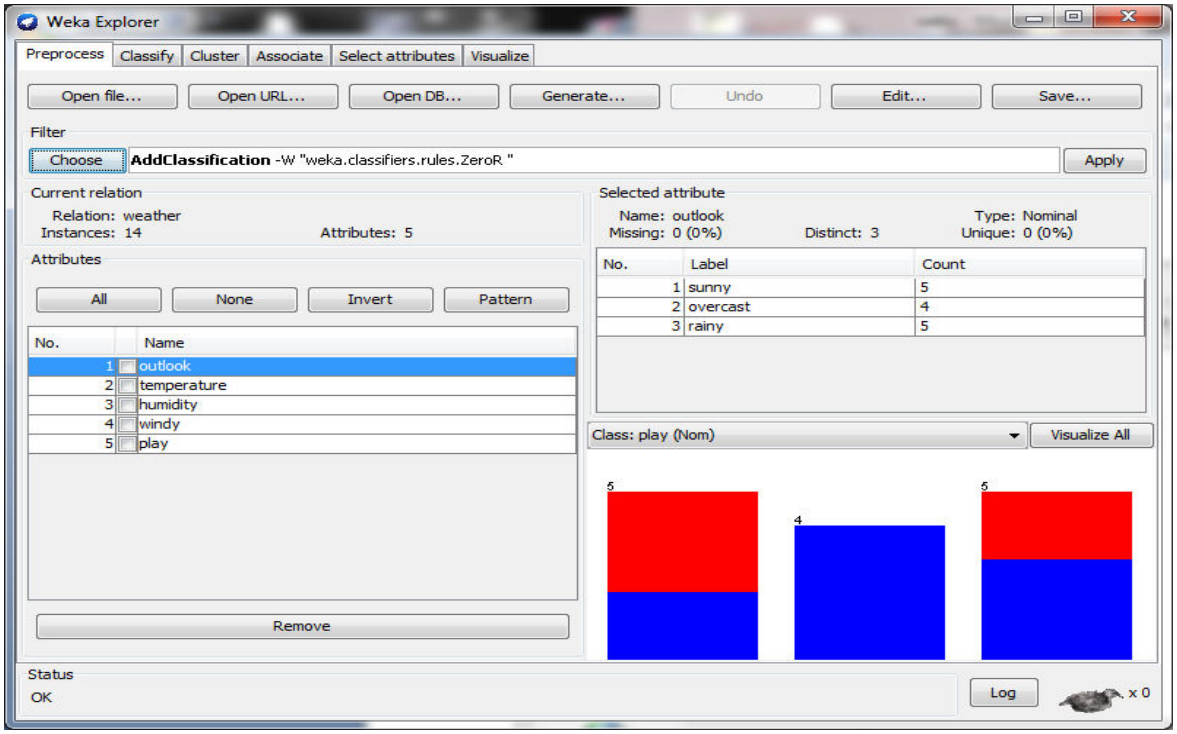


Figure 10.4 Applying Chosen Filter

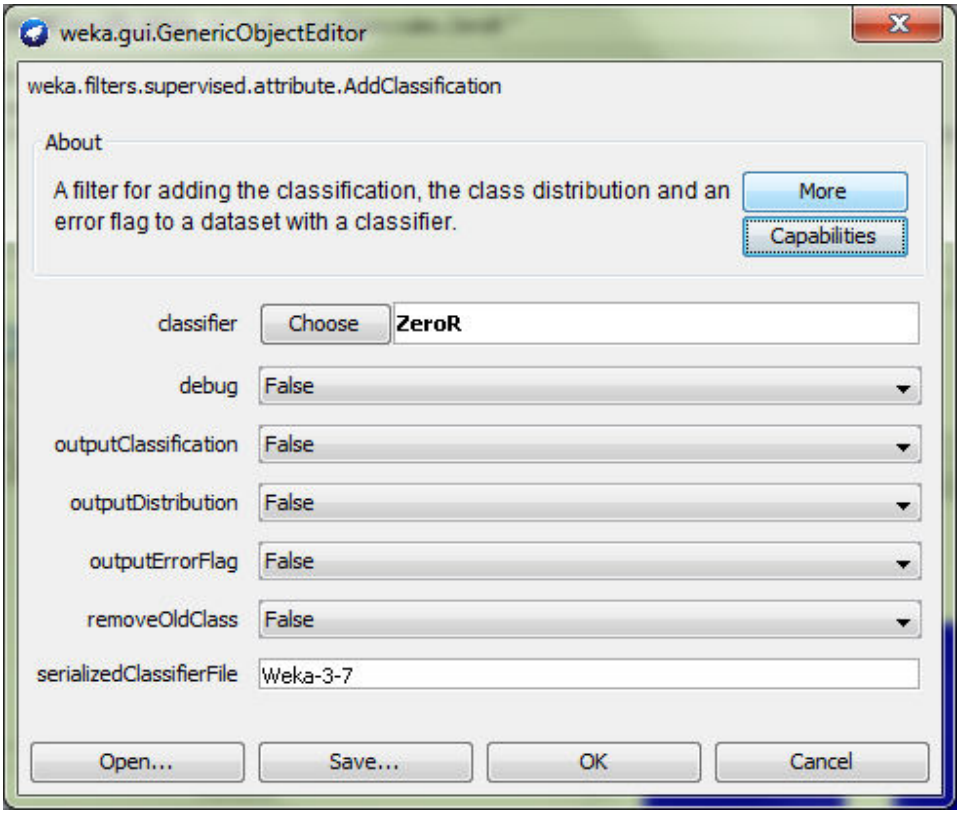


Figure 10.5 Generic Object Editor

Properties of AddClassification:

NAME

weka.filters.supervised.attribute.**AddClassification**

SYNOPSIS

A filter for adding the classification, the class distribution and an error flag to a dataset with a classifier. The classifier is either trained on the data itself or provided as serialized model.

OPTIONS

debug -- Turns on output of debugging information.

outputDistribution -- Whether to add attributes with the distribution for all classes (for numeric classes this will be identical to the attribute output with 'outputClassification').

serializedClassifierFile -- A file containing the serialized model of a trained classifier.

removeOldClass -- Whether to remove the old class attribute.

outputClassification -- Whether to add an attribute with the actual classification.

classifier -- The classifier to use for classification.

outputErrorFlag -- Whether to add an attribute indicating whether the classifier output a wrong classification (for numeric classes this is the numeric difference).

Step 6: Now, try selecting a few attributes of your choice, as in below Figureure, we have selected the temperature and humidity attributes.

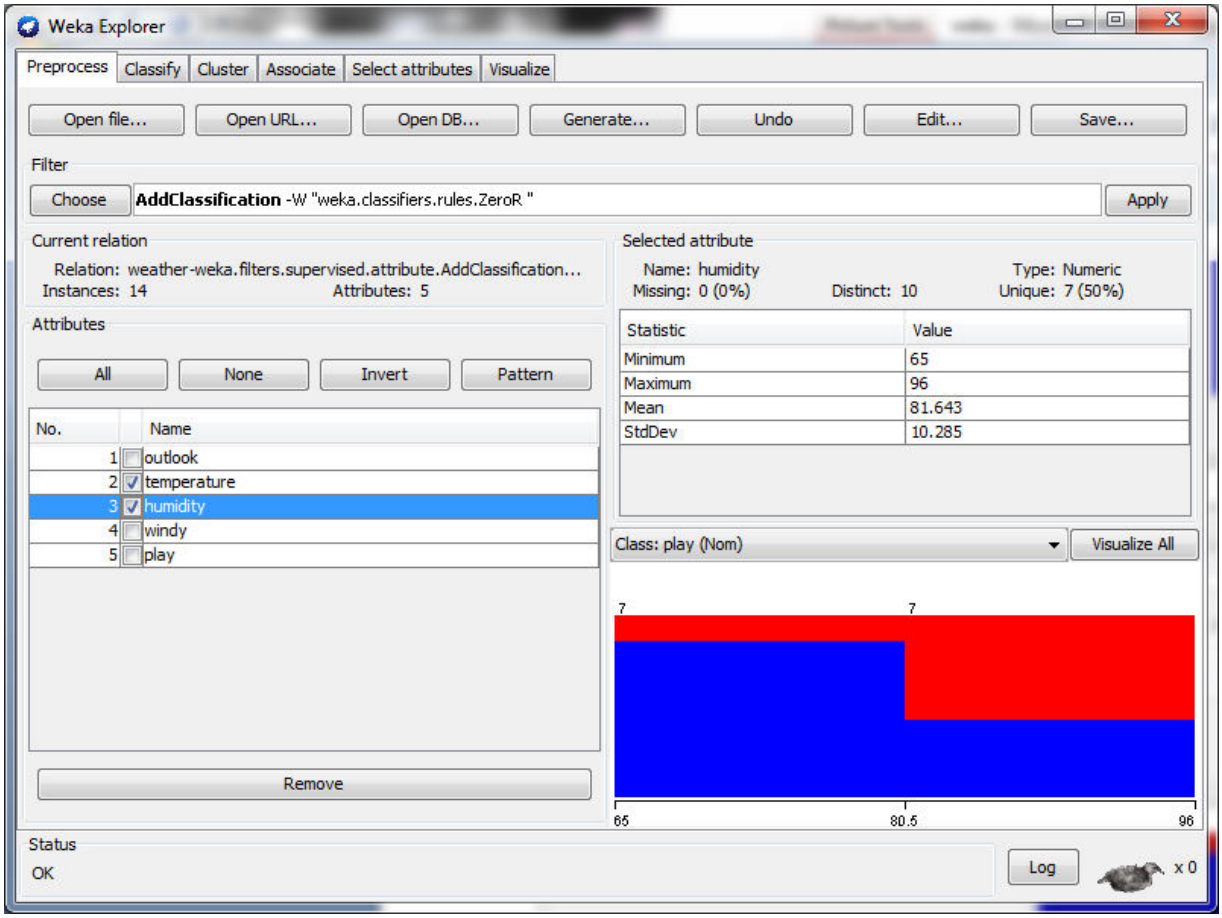


Figure 10.6 Visualization of Preprocessed data

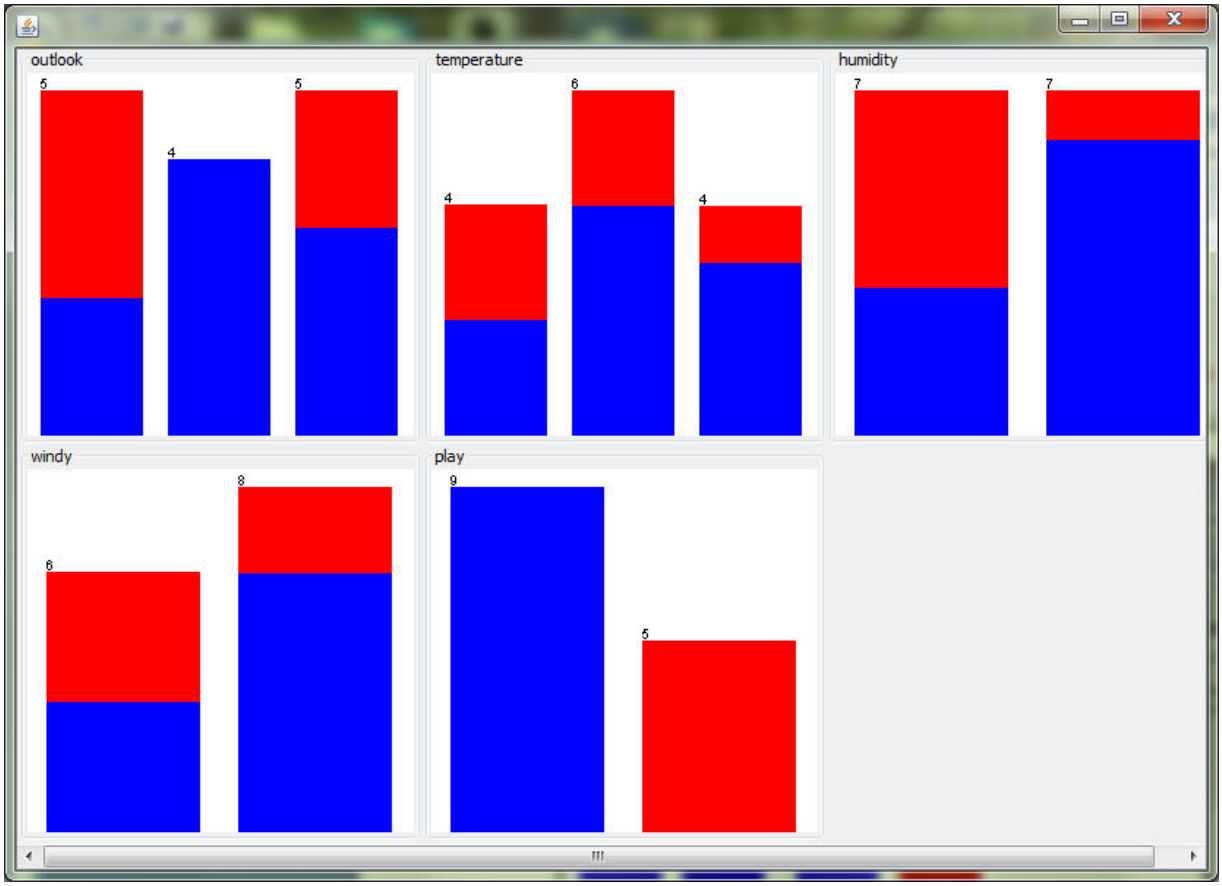


Figure 14.7 Visualization of all the Attributes

Practical No. 6

Aim: Perform exploratory analysis of the data to be used for mining.

Requirements:

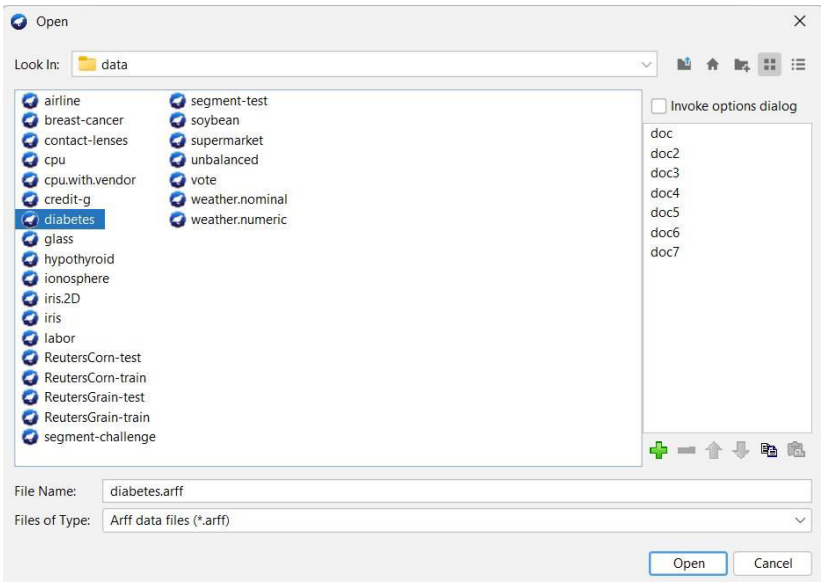
- Weka software (version 3.8 or later)
- A dataset (in .arff or .csv format)

Step 1: Load the Data

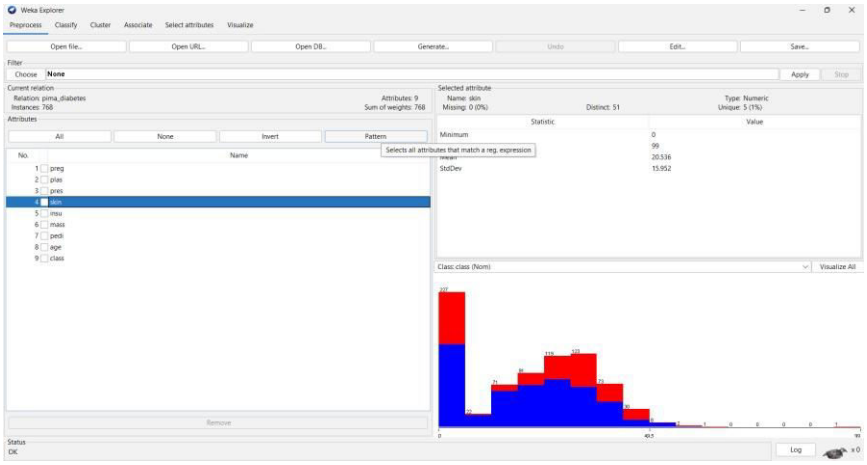
1. **Open Weka:**
    - Launch Weka on your machine.

2. **Select the Explorer tab:**
    - In the Weka main window, click on the "Explorer" button.
3. **Open the Dataset:**
    - In the Explorer window, click the "Open file..." button on the top left.
        - Navigate to the directory where your dataset is stored. ○

    Select the dataset (preferably in .arff format) and click "Open."

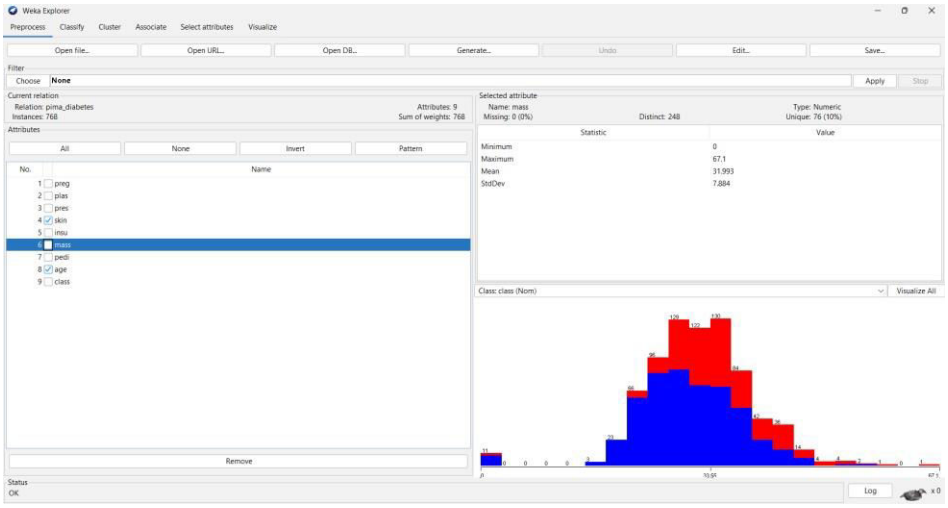


4. **View the Dataset:** ○ Once the dataset is loaded, you will see various tabs like "Preprocess," "Classify," etc. ○ Under the "Preprocess" tab, you can view the attributes and instances of the dataset.

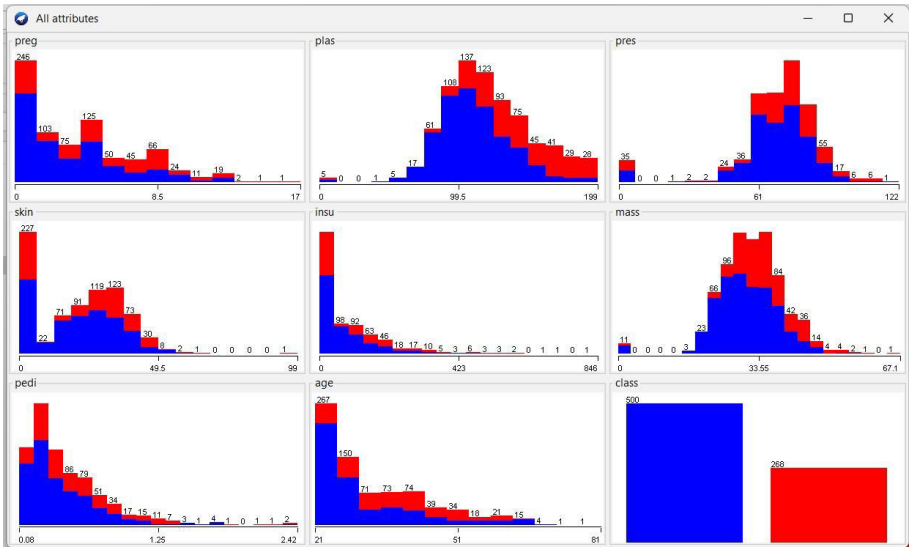


Step 2: Exploratory Data Analysis (EDA)

1. **Attribute Selection:**
   - In the "Preprocess" tab, view all the attributes listed in the Attributes panel. ○ You can select/deselect attributes by checking/unchecking them if needed.



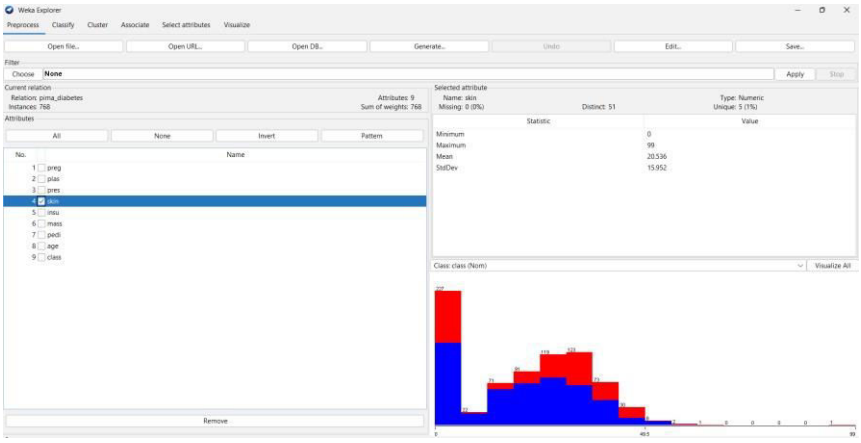
   - Click on the "Visualize All" button to get a scatterplot matrix of all attributes.



2. **Summary Statistics:**
    - At the bottom of the "Preprocess" window, Weka will show summary statistics such as the number of instances, missing values, distinct values, etc.

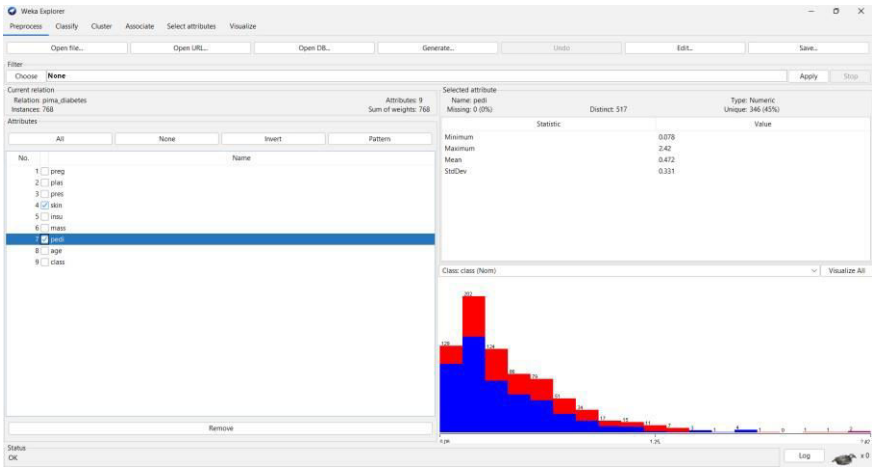
Step 3: Visualize Individual Attributes

1. **Visualize Individual Attributes:**
    - Select any attribute by clicking on it in the "Attributes" panel.
    - Click on the "Visualize" button. ○ Weka will display a histogram for that attribute, which provides insights into the distribution.



2. **Explore Correlations:**

- In the scatterplot matrix (under the "Visualize All" button), you can analyze the correlation between various attributes.
- Right-click on any point to zoom into specific parts of the graph.

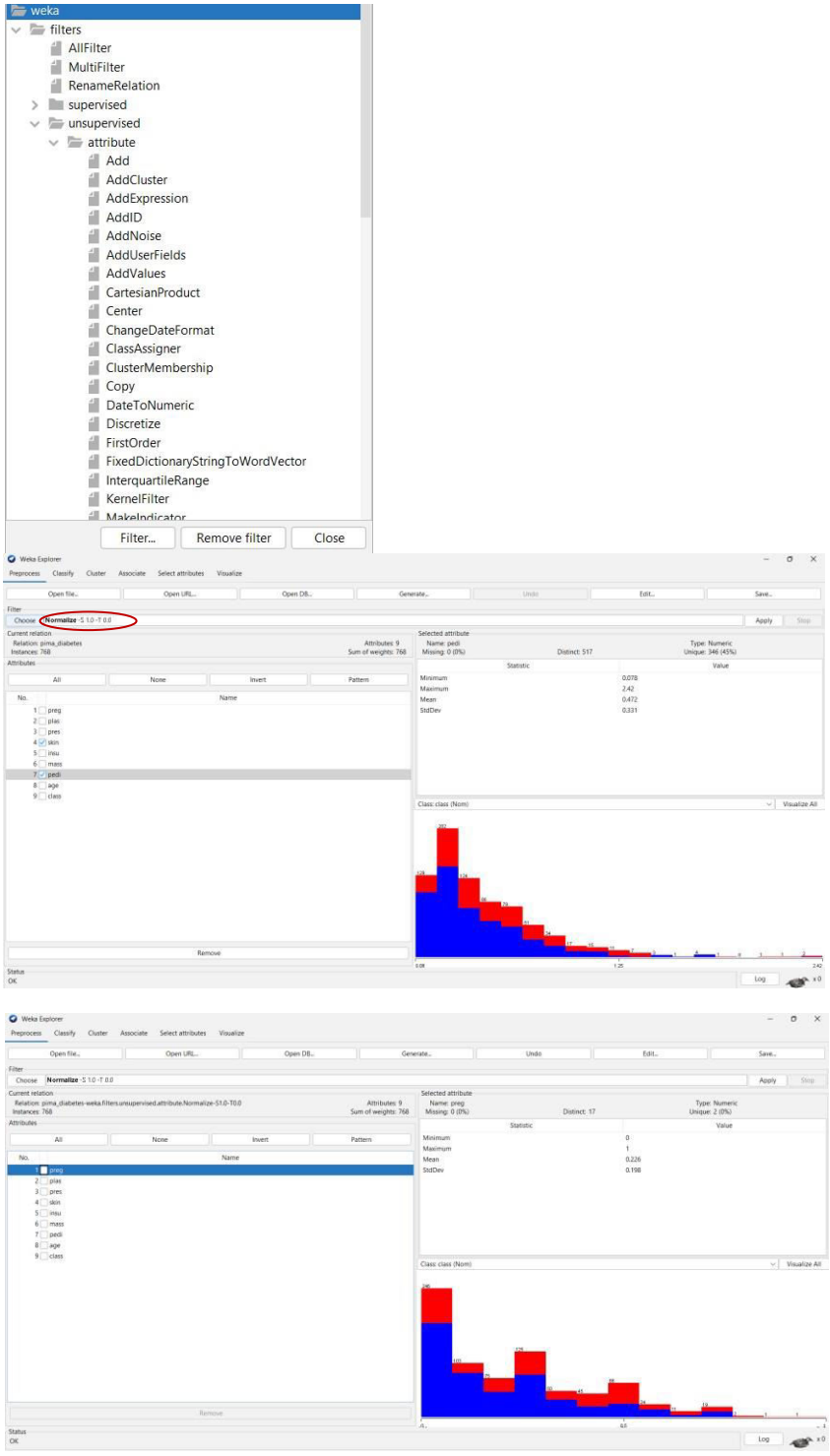


Step 4: Handle Missing Values (if any)

1. **Detect Missing Values:**
   - In the summary statistics for each attribute, you can check if any attribute contains missing values.
2. **Impute Missing Data:**
   - To handle missing values, click on the "Filter" button in the "Preprocess" tab.
     - Select a filter such as ReplaceMissingValues to automatically handle missing values.

Step 5: Data Transformation (if necessary)

1. **Normalize/Standardize Data:**
   - If your data requires normalization or standardization, click on "Filter" in the "Preprocess" tab.
   - Choose a filter like Normalize or Standardize from the list and apply it to your dataset.



2. **Discretization (optional):**
    - If you need to convert continuous attributes into discrete ones, apply a discretization filter such as Discretize to specific attributes.

Step 6: Save the Preprocessed Data

1. **Save Data for Mining:**
    - After performing the necessary preprocessing, save the modified dataset by clicking on the "Save" button in the "Preprocess" tab. o Save it as a new .arff file for further mining experiments.

Practical No. 7

Aim: Using open source tools Implement Classifiers

Step 1: In this Practical, select the Classify tab in the Weka Explorer and then click on the Choose button in the Classifier panel. Expand the classifier folder in the Weka tree view, in that expand the trees folder. Now select the J48 Algorithm and then click on Start. The output of classification of the selected data is obtained in the Classifier output pane.

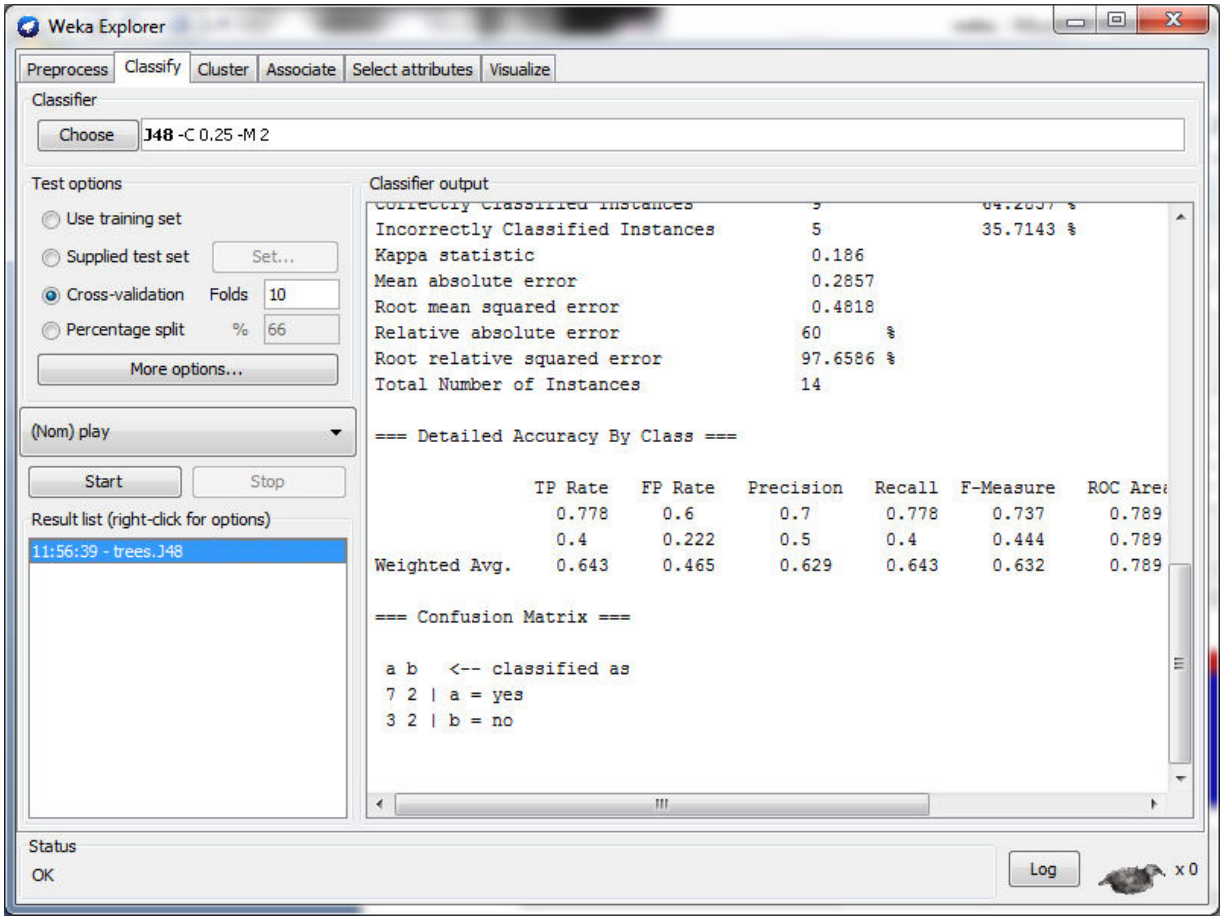


Figure 12.1 Weka Explorer- Classify

Output:-

=== Run information ===

Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2
Relation: weather-weka.filters.supervised.attribute.AddClassification-
Wweka.classifiers.rules.ZeroR
Instances: 14
Attributes: 5
outlook
temperature
humidity
windy
play
Test mode:10-fold cross-validation
=== Classifier model (full training set) ===
J48 pruned tree
---
outlook = sunny
| humidity <= 75: yes (2.0)
| humidity > 75: no (3.0)
outlook = overcast: yes (4.0)
outlook = rainy
| windy = TRUE: no (2.0)
| windy = FALSE: yes (3.0)
Number of Leaves : 5
Size of the tree : 8
Time taken to build model: 0 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances 9 64.2857 %

Incorrectly Classified Instances 5 35.7143 %
Kappa statistic 0.186
Mean absolute error 0.2857
Root mean squared error 0.4818
Relative absolute error 60 %
Root relative squared error 97.6586 %
Total Number of Instances 14

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.778	0.6	0.7	0.778	0.737	0.789	yes
	0.4	0.222	0.5	0.4	0.444	0.789	no
Weighted Avg.	0.643	0.465	0.629	0.643	0.632	0.789	

=== Confusion Matrix ===

a	b	<-- classified as
7	2	a = yes
3	2	b = no

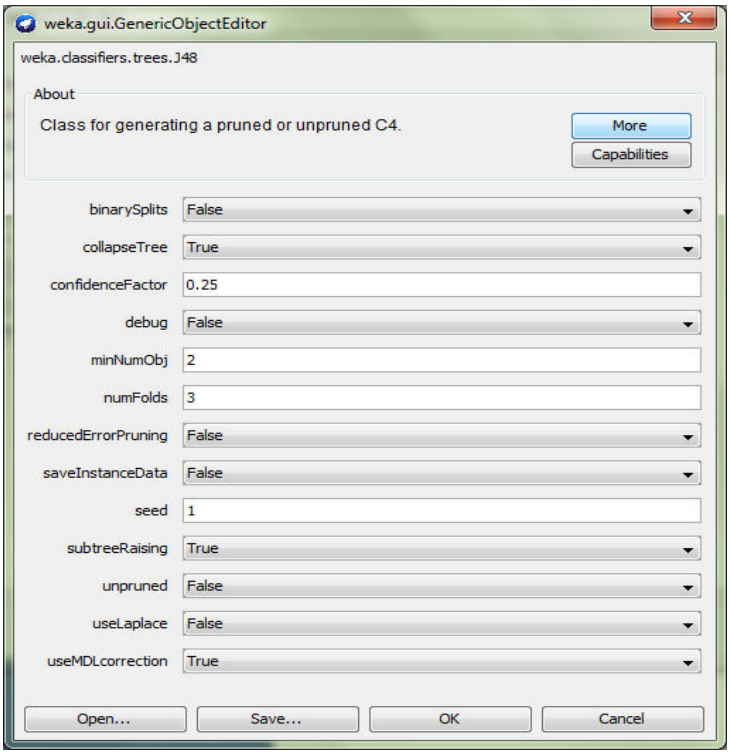


Figure 12.2 Generic Object Editor

Properties of J48 Algorithm:

NAME

weka.classifiers.trees.**J48**

SYNOPSIS

Class for generating a pruned or unpruned C4.5 decision tree. For more information, see

Ross Quinlan (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA.

OPTIONS

debug -- If set to true, classifier may output additional info to the console.

minNumObj -- The minimum number of instances per leaf.

confidenceFactor -- The confidence factor used for pruning (smaller values incur more pruning).

binarySplits -- Whether to use binary splits on nominal attributes when building the trees.

seed -- The seed used for randomizing the data when reduced-error pruning is used.

numFolds -- Determines the amount of data used for reduced-error pruning. One fold is used for pruning, the rest for growing the tree.

saveInstanceData -- Whether to save the training data for visualization.

unpruned -- Whether pruning is performed.

subtreeRaising -- Whether to consider the subtree raising operation when pruning.

collapseTree -- Whether parts are removed that do not reduce training error.

useMDLcorrection -- Whether MDL correction is used when finding splits on numeric attributes.

useLaplace -- Whether counts at leaves are smoothed based on Laplace.

reducedErrorPruning -- Whether reduced-error pruning is used instead of C.4.5 pruning.

Step 2: Now, right click on tree.J48 option in the Result list pane and select the Visualize tree option.

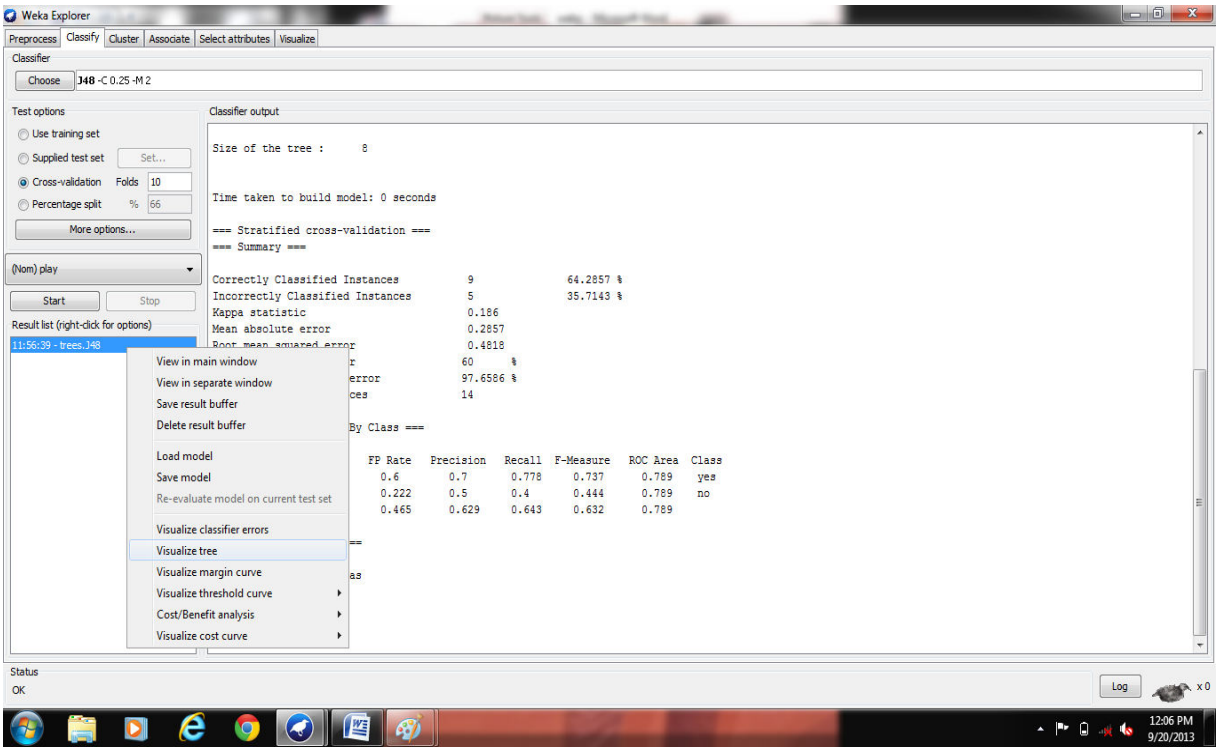


Figure 12.3 Classify- Result list pane

In Figure 12.3, the Weka Classifier Tree Visualizer window displays the obtained Tree view of the selected Data.

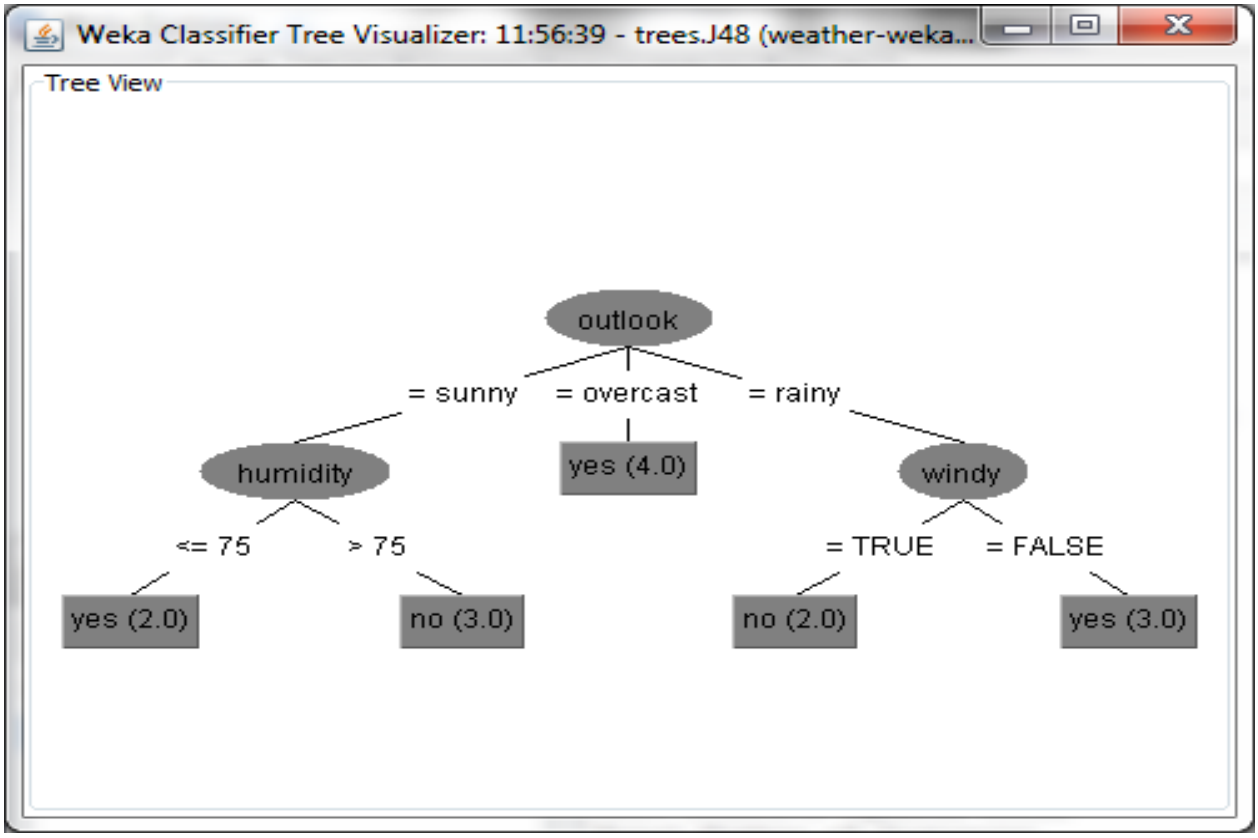


Figure 12.4 Weka Classifier Tree Visualizer

Practical No. 8

Aim: Using open source tools Implement Association Mining Algorithm.

Step 1: First open the Weka Explorer and select the Associate tab pane.

Step2: Now click on the Choose button in the Associator Panel and expand the associations tree view in the weka folder so as to select the algorithm for Association Rule Mining.

As shown in Figure 14.1, we have selected the Apriori algorithm.

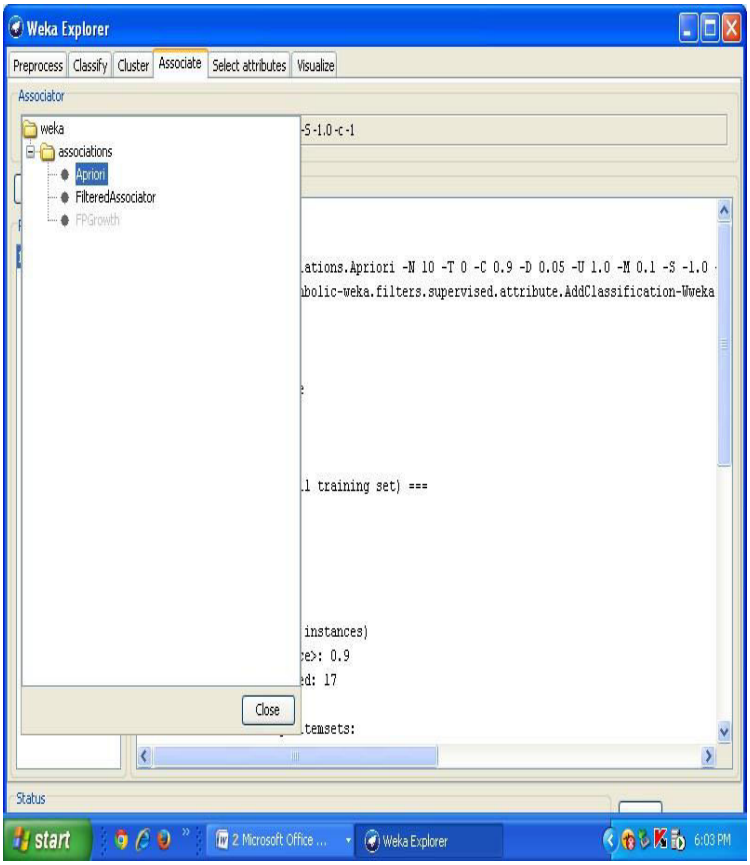


Figure 14.1 Weka Explorer- Selecting the Apriori algorithm

Step 3: After selecting the Algorithm, click on the Start button.

The output will be displayed in the Associator output section.

Output:

=== Run information ===

Scheme: weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

Relation: weather.symbolic-weka.filters.supervised.attribute.AddClassificationWweka.classifiers.rules.ZeroR-weka.filters.supervised.attribute.Discretize-Rfirstlast

Instances: 14

Attributes: 5

outlook

temperature

humidity

windy play

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

Apriori

=======

Minimum support: 0.15 (2 instances)

Minimum metric <confidence>: 0.9

Number of cycles performed: 17 Generated

sets of large itemsets:

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

of set of large itemsets L(2): 47

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

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

rules found:

1. outlook=overcast 4 ==> play=yes 4 <conf:(1)> lift:(1.56) lev:(0.1) [1] conv:(1.43)

2. temperature=cool 4 ==> humidity=normal 4 <conf:(1)> lift:(2) lev:(0.14) [2] conv:(2)

3. humidity=normal windy=FALSE 4 ==> play=yes 4 <conf:(1)> lift:(1.56) lev:(0.1) [1] conv:(1.43)

4. outlook=sunny play=no 3 ==> humidity=high 3 <conf:(1)> lift:(2) lev:(0.11) [1] conv:(1.5)

5. outlook=sunny humidity=high 3 ==> play=no 3 <conf:(1)> lift:(2.8) lev:(0.14) [1] conv:(1.93)

6. outlook=rainy play=yes 3 ==> windy=FALSE 3 <conf:(1)> lift:(1.75) lev:(0.09) [1] conv:(1.29)

7. outlook=rainy windy=FALSE 3 ==> play=yes 3 <conf:(1)> lift:(1.56) lev:(0.08) [1] conv:(1.07)

8. temperature=cool play=yes 3 ==> humidity=normal 3 <conf:(1)> lift:(2) lev:(0.11) [1] conv:(1.5)

9. outlook=sunny temperature=hot 2 ==> humidity=high 2 <conf:(1)> lift:(2) lev:(0.07) [1] conv:(1)

10. temperature=hot play=no 2 ==> outlook=sunny 2 <conf:(1)> lift:(2.8) lev:(0.09) [1] conv:(1.29)

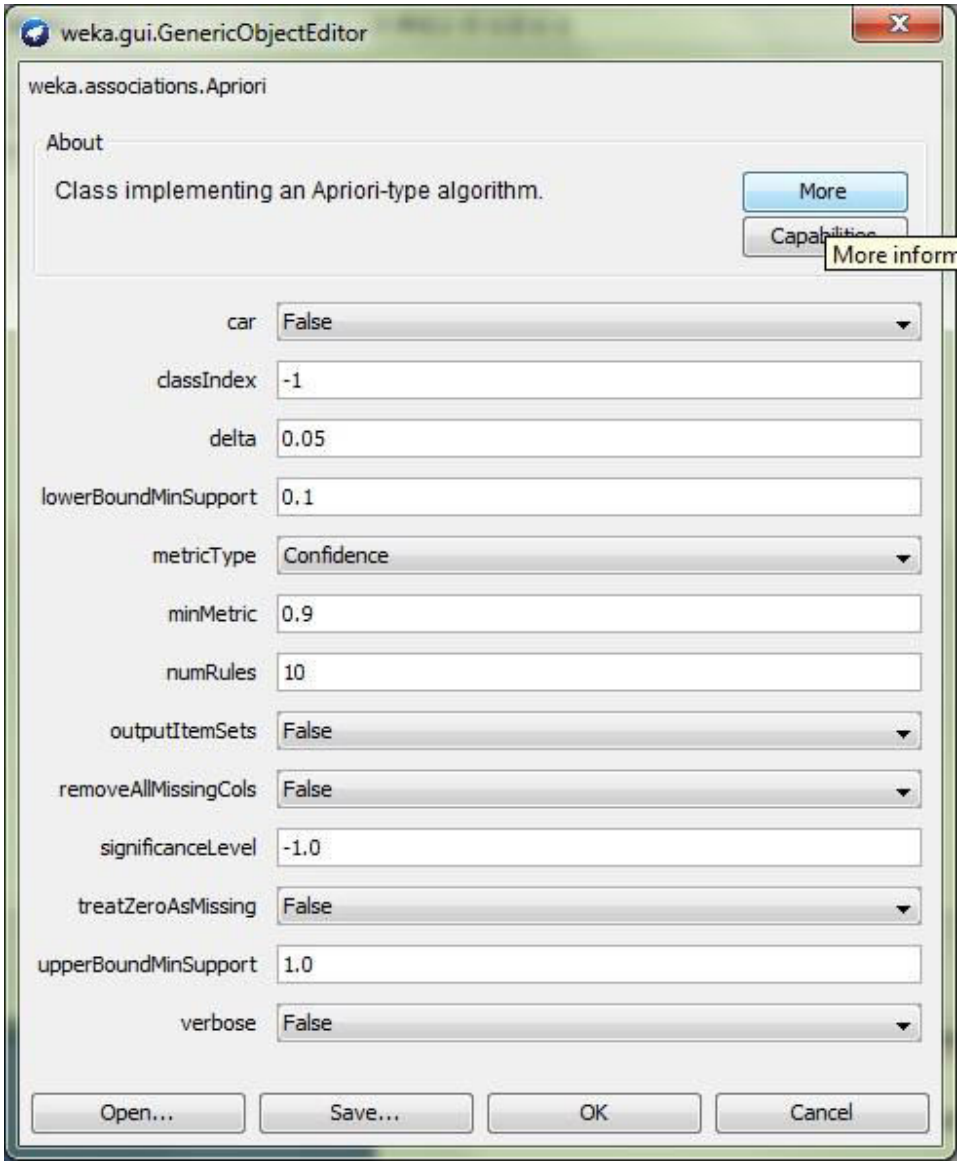


Figure 14.2 Generic Object Editor

Properties of Apriori:

NAME

weka.associations.Apriori

SYNOPSIS

Class implementing an Apriori-type algorithm. Iteratively reduces the minimum support until it finds the required number of rules with the given minimum confidence.

The algorithm has an option to mine class association rules. It is adapted as explained in the second reference.

For more information see:

R. Agrawal, R. Srikant: Fast Algorithms for Mining Association Rules in Large Databases. In: 20th International Conference on Very Large Data Bases, 478-499, 1994.

Bing Liu, Wynne Hsu, Yiming Ma: Integrating Classification and Association Rule Mining. In: Fourth International Conference on Knowledge Discovery and Data Mining, 80-86, 1998.

OPTIONS

upperBoundMinSupport -- Upper bound for minimum support. Start iteratively

decreasing minimum support from this value. removeAllMissingCols --

Remove columns with all missing values. lowerBoundMinSupport -- Lower

bound for minimum support. outputItemSets -- If enabled the itemsets are

output as well.

significanceLevel -- Significance level. Significance test (confidence metric only).

minMetric -- Minimum metric score. Consider only rules with scores higher than this value.

classIndex -- Index of the class attribute. If set to -1, the last attribute is taken as class attribute.

treatZeroAsMissing -- If enabled, zero (that is, the first value of a nominal) is treated in the same way as a missing value. numRules -- Number of rules to find.

delta -- Iteratively decrease support by this factor. Reduces support until min support is reached or required number of rules has been generated. verbose -- If enabled the algorithm will be run in verbose mode.

metricType -- Set the type of metric by which to rank rules. Confidence is the proportion of the examples covered by the premise that are also covered by the consequence (Class association rules can only be mined using confidence). Lift is confidence divided by the proportion of all examples that are covered by the consequence. This is a measure of the importance of the association that is independent of support. Leverage is the proportion of additional examples covered by both the premise and consequence above those expected if the premise and consequence were independent of each other. The total number of examples that this represents is presented in brackets following the leverage. Conviction is another measure of departure from independence. Conviction is given by P(premise)P(!consequence) / P(premise, !consequence).

car -- If enabled class association rules are mined instead of (general) association rules.

Practical No. 9

Aim: Using open source tools implement Clustering Algorithm.

Step 1: To perform clustering, open the Weka Explorer and select the Cluster tab.

Step 2: In the Clusterer panel, click the Choose button and expand the clusterer option in the weka tree view. And select the SimpleKMeans algorithm to cluster the data.

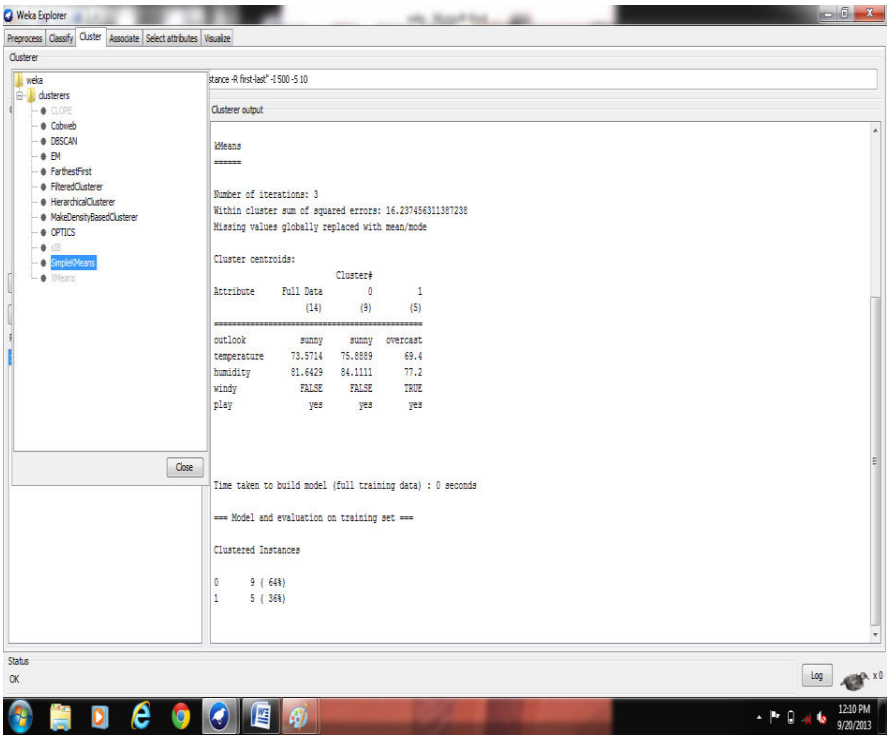


Figure 13.1 Weka Explorer- Cluster pane

Step 3: Now on clicking the start button, we will get to see the output in the Clusterer output window.

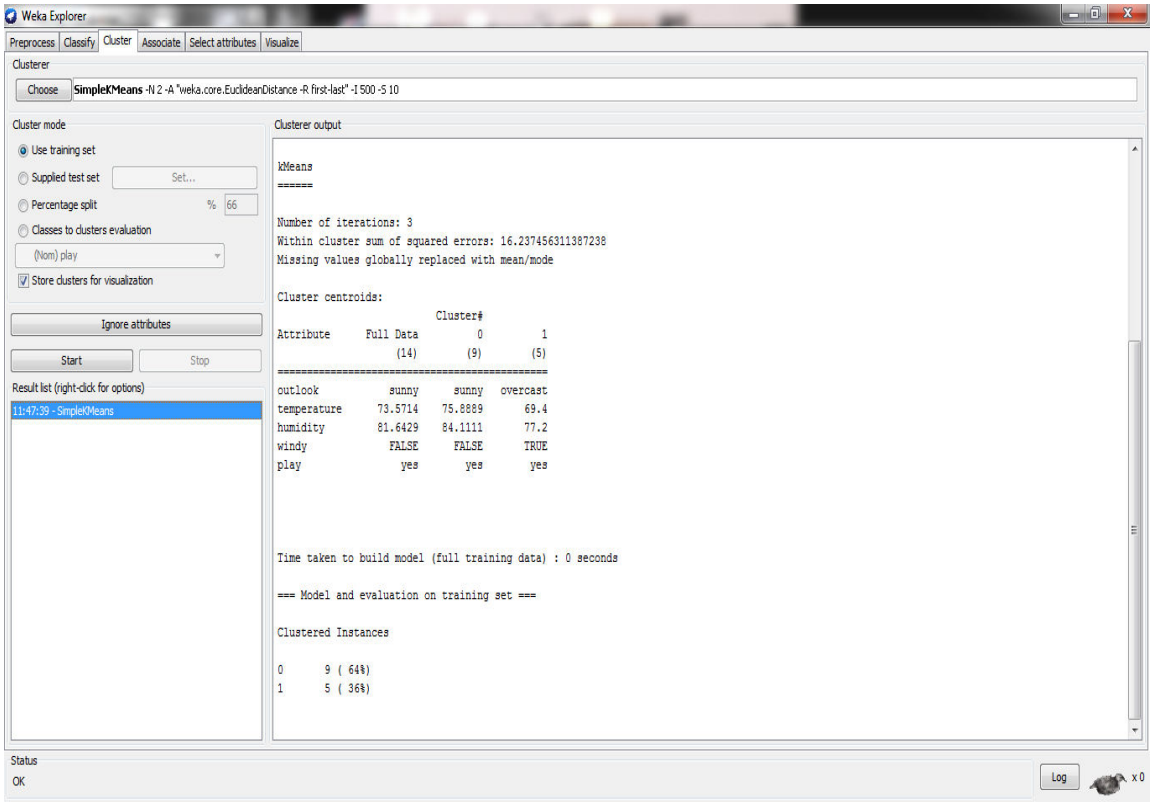


Figure 13.2 Clusterer output on applying SimpleKMeans

Output:-

=== Run information ===

Scheme:weka.clusterers.SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

Relation: weather-weka.filters.supervised.attribute.AddClassification-Wweka.classifiers.rules.ZeroR

Instances: 14

Attributes: 5

outlook

temperature

humidity

windy

play

Test mode:evaluate on training data

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

kMeans

======

Number of iterations: 3
Within cluster sum of squared errors: 16.237456311387238
Missing values globally replaced with mean/mode
Cluster centroids:

		Cluster#		
Attribute	Full Data	0	1	
	(14)	(9)	(5)	
outlook	sunny	sunny	overcast	
temperature	73.5714	75.8889	69.4	
humidity	81.6429	84.1111	77.2	
windy	FALSE	FALSE	TRUE	
play	yes	yes	yes	

Time taken to build model (full training data) : 0 seconds
=== Model and evaluation on training set ===
Clustered Instances
0 9 (64%)
1 5 (36%)

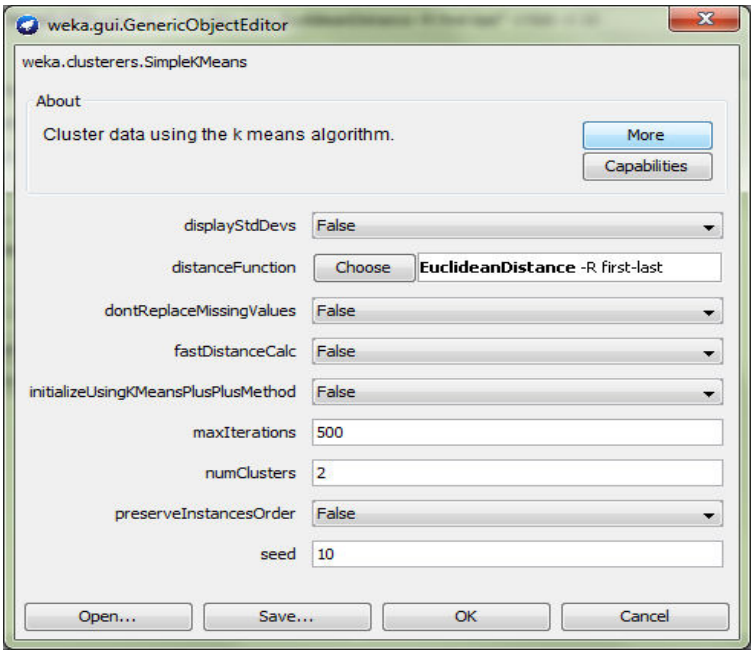


Figure 13.3 Clusterer output on applying SimpleKMeans

Properties of SimpleKMeans

NAME

weka.clusterers.SimpleKMeans

SYNOPSIS

Cluster data using the k means algorithm. Can use either the Euclidean distance (default) or the Manhattan distance. If the Manhattan distance is used, then centroids are computed as the component-wise median rather than mean. For more information see:

D. Arthur, S. Vassilvitskii: k-means++: the advantages of carefull seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, 1027-1035, 2007.

OPTIONS

distanceFunction -- The distance function to use for instances comparison (default: weka.core.EuclideanDistance).

seed -- The random number seed to be used.

preserveInstancesOrder -- Preserve order of instances.

numClusters -- set number of clusters

dontReplaceMissingValues -- Replace missing values globally with mean/mode.

maxIterations -- set maximum number of iterations

initializeUsingKMeansPlusPlusMethod -- Initialize cluster centers using the probabilistic farthest first method of the k-means++ algorithm

displayStdDevs -- Display std deviations of numeric attributes and counts of nominal attributes.

fastDistanceCalc -- Uses cut-off values for speeding up distance calculation, but suppresses also the calculation and output of the within cluster sum of squared errors/sum of distances.

Step 4: Now right click the SimpleKMeans option in the Result list pane. And select Visualize cluster assignments.

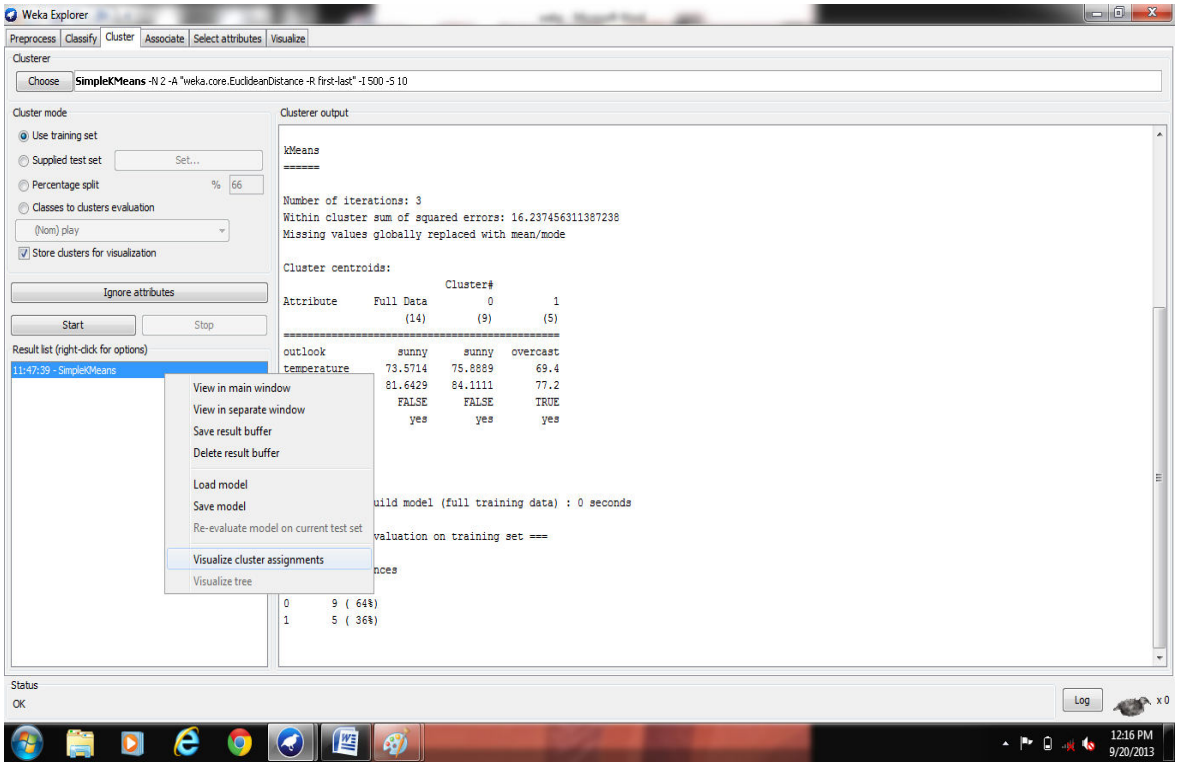


Figure 13.4 Cluster - Result list pane

The Figure 13.5 - Weka Clusterer Visualize window, displays the graph changes due to clustering.

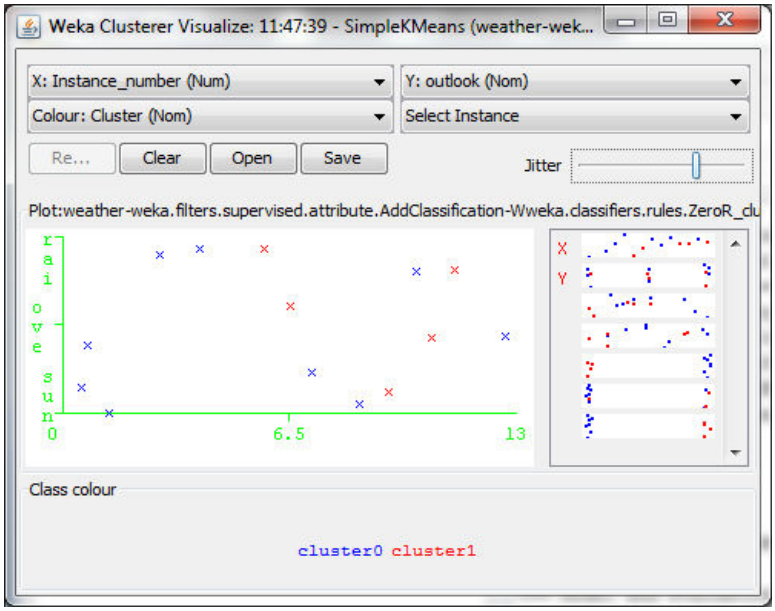


Figure 13.5 Weka Clusterer Visualize window

Practical No. 10

Aim: Implementation of any one classifier using Python.

Step 1: Open Google Colab

1. **Access Google Colab:**
    - **Open your web browser and go to Google Colab.**
2. **Create a New Notebook:**
    - **In the Google Colab interface, click on "File" > "New notebook" to create a new notebook for your code.**
    - **A new notebook interface will appear, where you can write and execute Python code.**

Step 2: Upload the .csv File

1. **Access the Files Panel:**
    - **On the left-hand side of the Colab interface, you will see a vertical toolbar.**
    - **Click on the folder icon to open the file system panel.**
2. **Upload the File:**
    - **In the Files panel, click on the "Upload" button.**
    - **Select your .csv file from your local system and upload it.**
    - **The uploaded file will now appear in the file list in Colab.**

Step 3: Mount Google Drive (Optional)

1. **If your .csv file is on Google Drive:**
    - **Run the following code in a Colab code cell to mount your Google Drive:**

```
from google.colab import drive
drive.mount('/content/drive')
```

2. **Access the File:**
    - **After running the code, a link will appear asking you to authorize Google Drive access.**
    - **Once authorized, you will see your Google Drive mounted under /content/drive/MyDrive/, where you can access your .csv file.**

Step 4: Read the CSV File in Code

1. **Read the Uploaded CSV:**

- **If you uploaded the file directly to Colab, you can use the file name directly in the pd.read_csv() function.**

Example:

df = pd.read_csv('your_file.csv')

- **If the file is in Google Drive, provide the full path to the file:**

df = pd.read_csv('/content/drive/MyDrive/path_to_your_file.csv')

Step 5: Write and Run Code on Colab

1. **Write Code in Code Cells:**
    - **Write Python code in the cells of your notebook.**

2. **Execute Code Cells:**
    - **To execute the code in a cell, press Shift + Enter or click on the "Play" icon next to the cell.**

Step 6: Save and Export Work

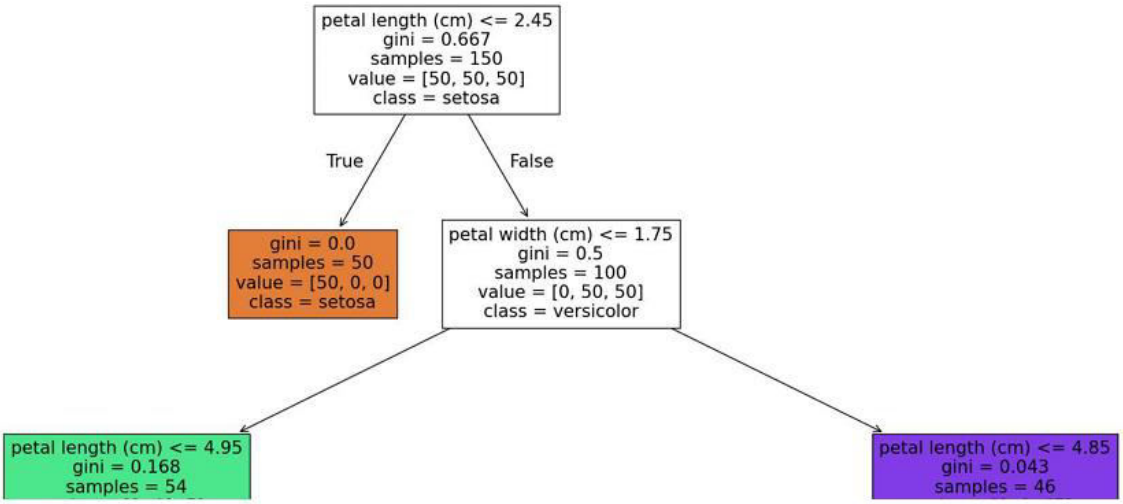
1. **Save Notebook:**
    - **To save your work, click on "File" > "Save" or "Save a copy in Drive."**
2. **Download Notebook:**
    - **You can download the notebook as a .ipynb or .py file by clicking "File" > "Download" > Select format.**

Code:

```
from matplotlib import pyplot as plt
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
# Prepare the data data
iris = datasets.load_iris()
X = iris.data
y = iris.target
# Fit the classifier with default hyper-parameters
clf = DecisionTreeClassifier(random_state=1234)
model = clf.fit(X, y)
text_representation = tree.export_text(clf)
print(text_representation)
with open("decistion_tree.log", "w") as fout:
    fout.write(text_representation)
    fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf,
                   feature_names=iris.feature_names,
                   class_names=iris.target_names,
                   filled=True)
```

Output:

```
|--- feature_2 <= 2.45
|   |--- class: 0
|--- feature_2 >  2.45
|   |--- feature_3 <= 1.75
|   |   |--- feature_2 <= 4.95
|   |   |   |--- feature_3 <= 1.65
|   |   |   |   |--- class: 1
|   |   |   |--- feature_3 >  1.65
|   |   |   |   |--- class: 2
|   |   |--- feature_2 >  4.95
|   |   |   |--- feature_3 <= 1.55
|   |   |   |   |--- class: 2
|   |   |   |--- feature_3 >  1.55
|   |   |   |   |--- feature_0 <= 6.95
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_0 >  6.95
|   |   |   |   |   |--- class: 2
|   |--- feature_3 >  1.75
|   |   |--- feature_2 <= 4.85
|   |   |   |--- feature_1 <= 3.10
|   |   |   |   |--- class: 2
|   |   |   |--- feature_1 >  3.10
|   |   |   |   |--- class: 1
|   |   |--- feature_2 >  4.85
|   |   |   |--- class: 2
```



Practical No. 11

Aim: Implementation of any one clustering algorithm using Python.

Step 1: Open Google Colab

1. **Access Google Colab:**
    - **Open your web browser and go to Google Colab.**
2. **Create a New Notebook:**
    - **In the Google Colab interface, click on "File" > "New notebook" to create a new notebook for your code.**
    - **A new notebook interface will appear, where you can write and execute Python code.**

Step 2: Upload the .csv File

1. **Access the Files Panel:**
    - **On the left-hand side of the Colab interface, you will see a vertical toolbar.**
    - **Click on the folder icon to open the file system panel.**
2. **Upload the File:**
    - **In the Files panel, click on the "Upload" button.**
    - **Select your .csv file from your local system and upload it.**
    - **The uploaded file will now appear in the file list in Colab.**

Step 3: Mount Google Drive (Optional)

1. **If your .csv file is on Google Drive:**
    - **Run the following code in a Colab code cell to mount your Google Drive:**

```
from google.colab import drive
drive.mount('/content/drive')
```

2. **Access the File:**
    - **After running the code, a link will appear asking you to authorize Google Drive access.**
    - **Once authorized, you will see your Google Drive mounted under /content/drive/MyDrive/, where you can access your .csv file.**

Step 4: Read the CSV File in Code

1. **Read the Uploaded CSV:**

- **If you uploaded the file directly to Colab, you can use the file name directly in the pd.read_csv() function.**

Example:

df = pd.read_csv('your_file.csv')

- **If the file is in Google Drive, provide the full path to the file:**

df = pd.read_csv('/content/drive/MyDrive/path_to_your_file.csv')

Step 5: Write and Run Code on Colab

1. **Write Code in Code Cells:**
    - **Write Python code in the cells of your notebook.**

2. **Execute Code Cells:**
    - **To execute the code in a cell, press Shift + Enter or click on the "Play" icon next to the cell.**

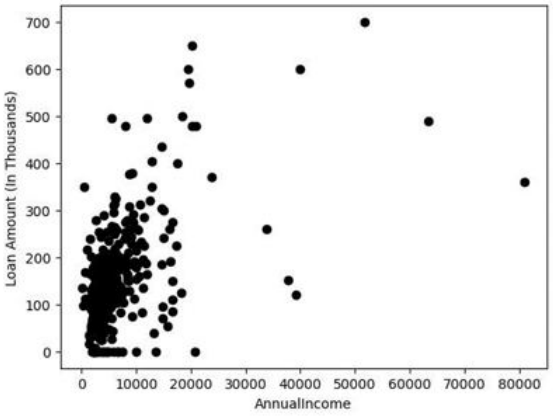
Step 6: Save and Export Work

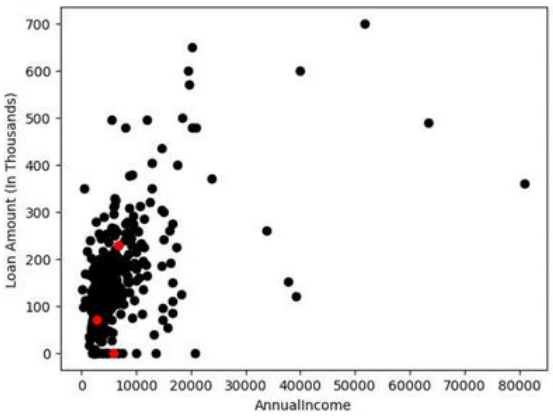
1. **Save Notebook:**
    - **To save your work, click on "File" > "Save" or "Save a copy in Drive."**
2. **Download Notebook:**
    - **You can download the notebook as a .ipynb or .py file by clicking "File" > "Download" > Select format.**

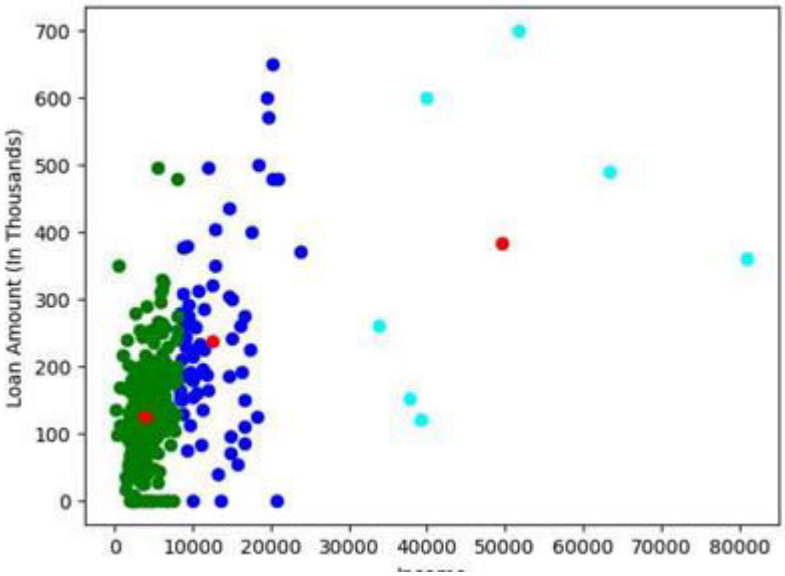
Code:

```
#import libraries
import pandas as pd
import numpy as np
import random as rd
import matplotlib.pyplot as plt
data = pd.read_csv('CreditRisk.csv')
data.head()
X = data[["LoanAmount","ApplicantIncome"]]
#Visualise data points
plt.scatter(X["ApplicantIncome"],X["LoanAmount"],c='black')
plt.xlabel('AnnualIncome')
plt.ylabel('Loan Amount (In Thousands)')
plt.show()
K=3
# Select random observation as centroids
Centroids = (X.sample(n=K))
plt.scatter(X["ApplicantIncome"],X["LoanAmount"],c='black')
plt.scatter(Centroids["ApplicantIncome"],Centroids["LoanAmount"],c='red')
plt.xlabel('AnnualIncome')
plt.ylabel('Loan Amount (In Thousands)')
plt.show()# Step 3 - Assign all the points to the closest cluster centroid
# Step 4 - Recompute centroids of newly formed clusters
# Step 5 - Repeat step 3 and 4
diff = 1
j=0
while(diff!=0):
    XD=X
    i=1
    for index1,row_c in Centroids.iterrows():
        ED=[]
        for index2,row_d in XD.iterrows():
            d1=(row_c["ApplicantIncome"]-row_d["ApplicantIncome"])**2
            d2=(row_c["LoanAmount"]-row_d["LoanAmount"])**2
            d=np.sqrt(d1+d2)
            ED.append(d)
        X[i]=ED
        i=i+1
    C=[]
    for index,row in X.iterrows():
        min_dist=row[1]
        pos=1
        for i in range(K):
            if row[i+1] < min_dist:
                min_dist = row[i+1]
                pos=i+1
        C.append(pos)
    X["Cluster"]=C
    Centroids_new = X.groupby(["Cluster"]).mean()[["LoanAmount","ApplicantIncome"]]
    if j == 0:
        diff=1
        j=j+1
    else:
        diff = (Centroids_new['LoanAmount'] - Centroids['LoanAmount']).sum() + (Centroids_new['ApplicantIncome'] - Centr
        print(diff.sum())
    Centroids = X.groupby(["Cluster"]).mean()[["LoanAmount","ApplicantIncome"]]
color=['blue','green','cyan']
for k in range(K):
    data=X[X["Cluster"]==k+1]
    plt.scatter(data["ApplicantIncome"],data["LoanAmount"],c=color[k])
plt.scatter(Centroids["ApplicantIncome"],Centroids["LoanAmount"],c='red')
plt.xlabel('Income')
plt.ylabel('Loan Amount (In Thousands)')
plt.show()
```

Output:


Loan Amount (In Thousands)
AnnualIncome
0
100
200
300
400
500
600
700
0
10000
20000
30000
40000
50000
60000
70000
80000


Loan Amount (In Thousands)
AnnualIncome
0
100
200
300
400
500
600
700
0
10000
20000
30000
40000
50000
60000
70000
80000


Loan Amount (In Thousands)
0
100
200
300
400
500
600
700
0
10000
20000
30000
40000
50000
60000
70000
80000

Practical No. 12

Aim: Implementation of any one association mining algorithm using Python.

Step 1: Open Google Colab

1. **Access Google Colab:**
    - **Open your web browser and go to Google Colab.**
2. **Create a New Notebook:**
    - **In the Google Colab interface, click on "File" > "New notebook" to create a new notebook for your code.**
    - **A new notebook interface will appear, where you can write and execute Python code.**

Step 2: Upload the .csv File

1. **Access the Files Panel:**
    - **On the left-hand side of the Colab interface, you will see a vertical toolbar.**
    - **Click on the folder icon to open the file system panel.**
2. **Upload the File:**
    - **In the Files panel, click on the "Upload" button.**
    - **Select your .csv file from your local system and upload it.**
    - **The uploaded file will now appear in the file list in Colab.**

Step 3: Mount Google Drive (Optional)

1. **If your .csv file is on Google Drive:**
    - **Run the following code in a Colab code cell to mount your Google Drive:**

```
from google.colab import drive
drive.mount('/content/drive')
```

2. **Access the File:**
    - **After running the code, a link will appear asking you to authorize Google Drive access.**
    - **Once authorized, you will see your Google Drive mounted under /content/drive/MyDrive/, where you can access your .csv file.**

Step 4: Read the CSV File in Code

1. **Read the Uploaded CSV:**

- **If you uploaded the file directly to Colab, you can use the file name directly in the pd.read_csv() function.**

Example:

df = pd.read_csv('your_file.csv')

- **If the file is in Google Drive, provide the full path to the file:**

df = pd.read_csv('/content/drive/MyDrive/path_to_your_file.csv')

Step 5: Write and Run Code on Colab

1. **Write Code in Code Cells:**
    - **Write Python code in the cells of your notebook.**

2. **Execute Code Cells:**
    - **To execute the code in a cell, press Shift + Enter or click on the "Play" icon next to the cell.**

Step 6: Save and Export Work

1. **Save Notebook:**
    - **To save your work, click on "File" > "Save" or "Save a copy in Drive."**
2. **Download Notebook:**
    - **You can download the notebook as a .ipynb or .py file by clicking "File" > "Download" > Select format.**

Code:

```
!pip install mlxtend
import warnings
warnings.filterwarnings('ignore')
# Import necessary libraries
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Create a sample dataset
data = {
    'bread': [1, 1, 1, 0, 1],
    'milk': [1, 1, 0, 1, 1],
    'cheese': [1, 0, 0, 1, 1],
    'apples': [0, 1, 1, 1, 0],
    'bananas': [0, 0, 1, 0, 1]
}

# Convert the data to a DataFrame
df = pd.DataFrame(data)

# Convert integer values to boolean
df = df.astype(bool)

# Display the dataset
print("Sample Dataset:")
print(df)

# Apply the Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)

# Display the frequent itemsets
print("\nFrequent Itemsets:")
print(frequent_itemsets)

# Generate association rules with a minimum confidence of 0.7
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

# Display the association rules
print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

Output:

```
Sample Dataset:
   bread   milk  cheese  apples  bananas
0   True   True    True   False    False
1   True   True   False    True    False
2   True  False   False    True     True
3  False   True    True    True    False
4   True   True    True   False     True

Frequent Itemsets:
   support        itemsets
0      0.8         (bread)
1      0.8          (milk)
2      0.6        (cheese)
3      0.6        (apples)
4      0.6   (milk, bread)
5      0.6  (milk, cheese)

Association Rules:
  antecedents consequents  support  confidence    lift
0      (milk)     (bread)      0.6        0.75  0.9375
1     (bread)      (milk)      0.6        0.75  0.9375
2      (milk)    (cheese)      0.6        0.75  1.2500
3    (cheese)      (milk)      0.6        1.00  1.2500
```