Experiment – 1

Aim: Loading and Exploring Data in WEKA

Introduction:

WEKA (Waikato Environment for Knowledge Analysis) is a popular open-source tool used for data mining and machine learning. It provides a graphical interface through which users can preprocess data, apply classification or clustering algorithms, and visualize results without the need for programming. One of the first steps in any data mining process is to load and explore the dataset. The Preprocess tab in WEKA allows users to load data in ARFF or CSV format, view attributes, check data types (nominal or numeric), observe data distribution, and identify missing or abnormal values. Understanding the structure and distribution of the dataset is essential before applying any data mining technique.

Input:

- Software: WEKA Explorer (version 3.8 or higher)
- Supported File Formats: .arff (preferred), .csv
- Operating System: Windows/Linux

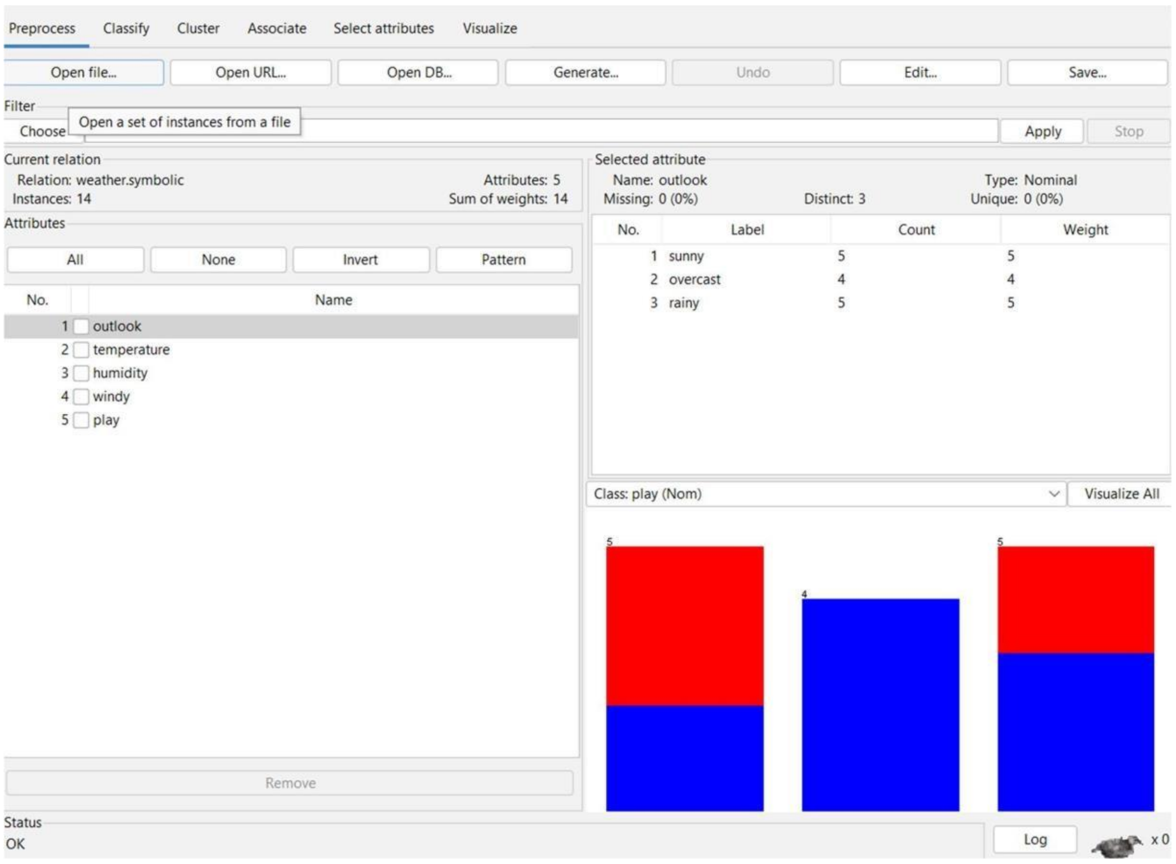
Procedure:

1. Open WEKA and select **Explorer** from the GUI Chooser.
2. In the **Preprocess** tab, click on **"Open file…"**.
3. Load a dataset such as weather.arff or iris.arff from the data folder.
4. View the dataset details: relation name, number of attributes, and instances.
5. Click on each attribute to check its type (nominal or numeric).
6. Observe attribute statistics like min, max, mean, standard deviation, or category counts.
7. Check the graphical visualization of each attribute shown below.

Observation:

The dataset `weather.arff` was successfully loaded in WEKA. It contains 5 attributes and 14 instances. "Outlook" is a nominal attribute with three categories: sunny, overcast, and rainy. "Humidity" is a numeric attribute with values ranging between 65 and 95. The graphical view displayed the frequency and distribution of each attribute clearly.

Output:



Current relation
Relation: weather.symbolic   Attributes: 5
Instances: 14   Sum of weights: 14

Attributes

No.	Name
1	outlook
2	temperature
3	humidity
4	windy
5	play

Selected attribute
Name: outlook   Type: Nominal
Missing: 0 (0%)   Distinct: 3   Unique: 0 (0%)

No.	Label	Count	Weight
1	sunny	5	5
2	overcast	4	4
3	rainy	5	5

Class: play (Nom)

Status
OK

Results:

The experiment was completed successfully. The dataset was loaded in WEKA using the Explorer’s Preprocess tab, and all attributes were explored with their respective types, statistics, and visualizations.

Experiment – 2

Aim: Data Preprocessing in WEKA

Input:

- Software: WEKA (version 3.8 or above)
- Dataset Format: .arff (preferred) or .csv
- System Requirements: Windows/Linux with Java Runtime Environment
- Sample Datasets: weather.arff, iris.arff, or any other sample dataset

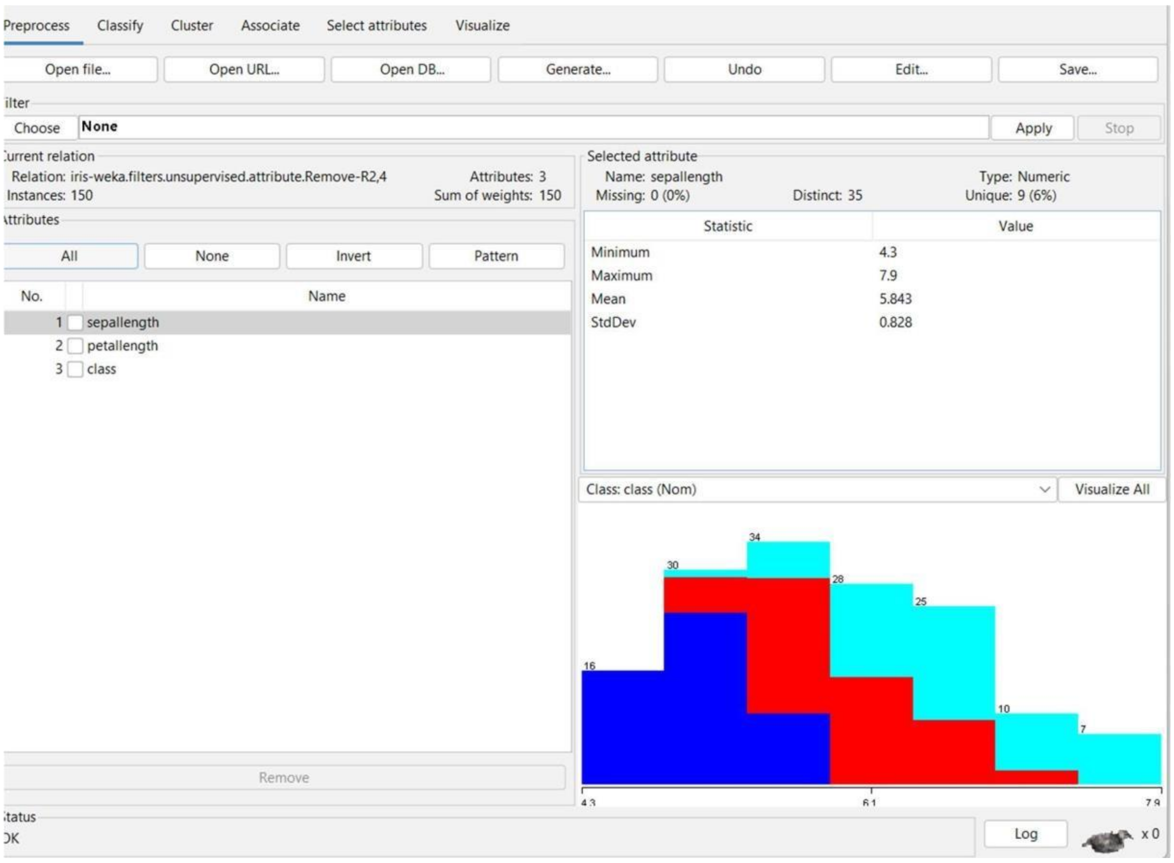
Procedure:

1. Open **WEKA** and click on **Explorer** from the GUI Chooser.
2. Go to the **Preprocess** tab and click **"Open file…"** to load a dataset.
3. Identify any **unwanted attributes** and select them from the list.
4. Click on the **"Remove"** button to delete the selected attribute.
5. To handle missing or unclean data, click on the **"Choose"** button in the Filter section.
6. Select filters like:
   - ReplaceMissingValues (for missing data)
   - Normalize (for scaling values)
   - Standardize (for mean-centering)
7. After selecting a filter, click **"Apply"** to preprocess the data.
8. Observe the changes in attribute values after applying filters.

Observation:

The dataset weather.arff was loaded in WEKA. Preprocessing was performed using filters like ReplaceMissingValues and Normalize. Missing values were replaced with suitable values, and numeric attributes were scaled to a standard range. The dataset was cleaned and standardized for further data mining operations.

Output:


Current relation
Relation: iris-weka.filters.unsupervised.attribute.Remove-R2,4
Instances: 150
Attributes: 3
Sum of weights: 150

Selected attribute
Name: sepallength
Missing: 0 (0%)
Distinct: 35
Type: Numeric
Unique: 9 (6%)

Statistic	Value
Minimum	4.3
Maximum	7.9
Mean	5.843
StdDev	0.828

No.	Name
1	sepallength
2	petallength
3	class

Filter: None

Class: class (Nom)

Status
OK

Result:

The experiment was completed successfully. Using WEKA, basic data preprocessing techniques such as handling missing values, normalization, and attribute removal were applied. The dataset was made consistent and ready for the application of machine learning algorithms.

Experiment – 3

Aim: Data Visualization in WEKA

Input:

- WEKA Software
- Sample dataset (iris.arff, weather.arff)

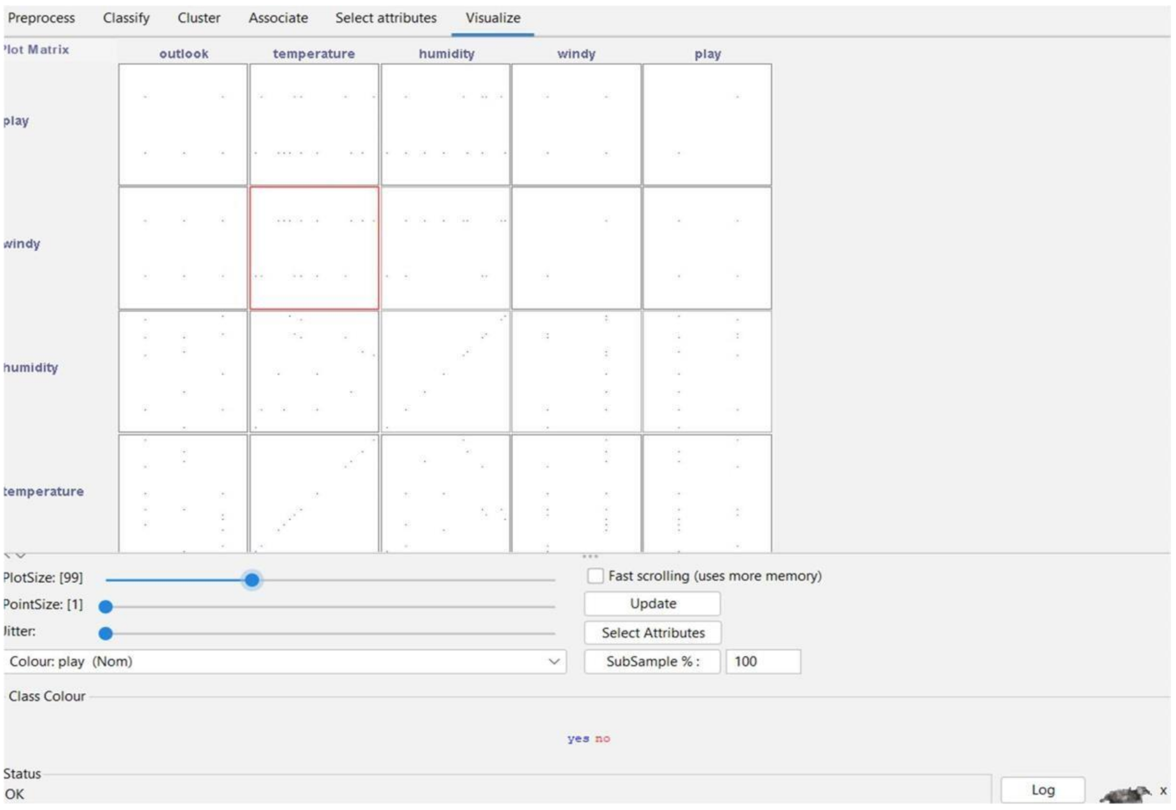
Procedure:

1. Open WEKA and select Explorer from the GUI Chooser.
2. In the Preprocess tab, load a dataset using the "Open file…" option.
3. Click on any attribute in the list to view its graphical distribution.
4. For nominal attributes, observe the bar chart showing category frequencies.
5. For numeric attributes, observe the histogram showing value ranges.
6. Now switch to the Visualize tab at the top.
7. In the Visualize tab, study the scatter plots comparing pairs of attributes.
8. Use the plots to understand attribute relationships and data spread.

Observation:

Nominal attributes like "Outlook" showed clear category frequencies through bar charts. Numeric attributes like "Humidity" showed a well-distributed histogram. The Visualize tab displayed scatter plots for comparing two attributes.

Output:



PlotSize: [99]
PointSize: [1]
Jitter:
Colour: play (Nom)
SubSample % : 100
Status
OK

Result:

Data visualization was successfully done using both the Preprocess and Visualize tabs in WEKA, helping to better understand data distribution.

Experiment – 4

Aim: Association Rule Mining in WEKA

Input:

- WEKA Software
- A categorical or transactional dataset in ARFF or CSV format. A commonly used example is supermarket.arff available in the WEKA data folder.

Procedure:

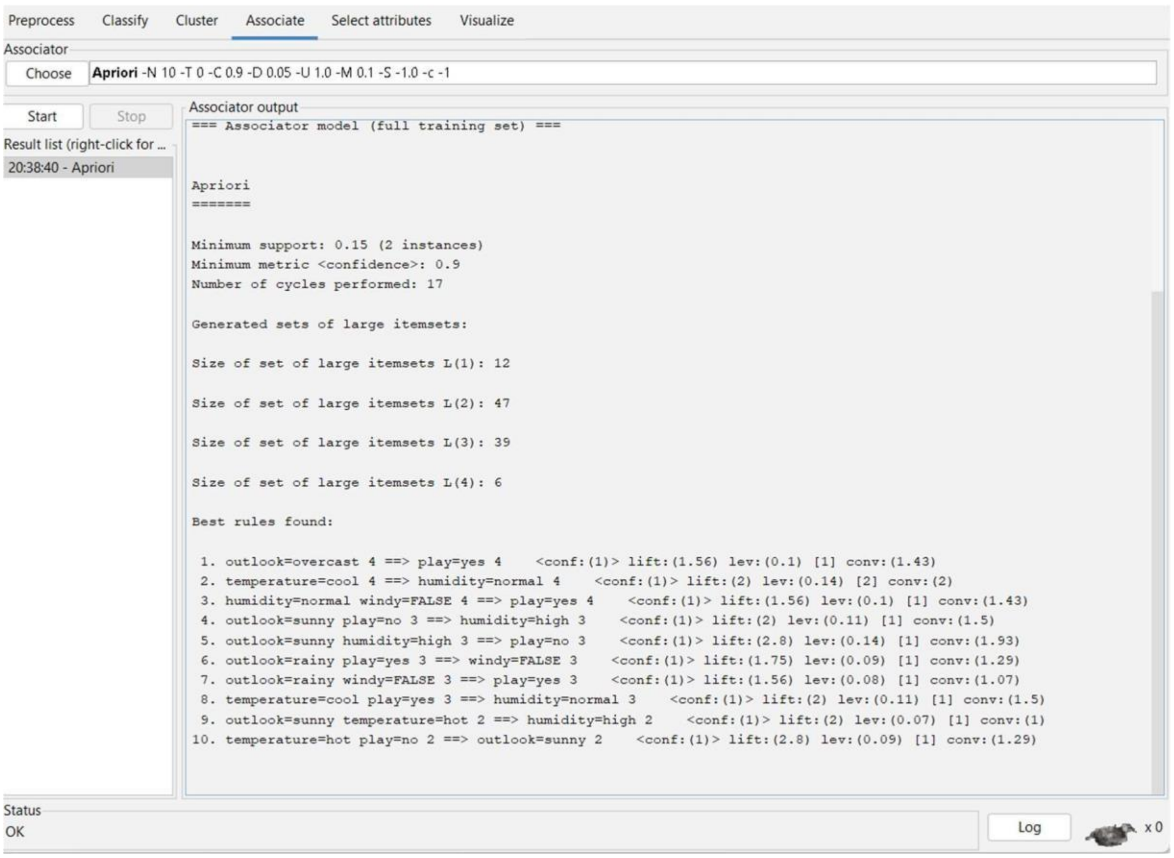
1. Start WEKA and open the Explorer interface.
2. Select “Open file” and load the dataset.
3. Verify that attributes are nominal. If any attributes are numeric, apply the Discretize filter from the Preprocess tab by selecting Filter, Unsupervised, Attribute, Discretize, and then Apply.
4. Go to the Associate tab.
5. From the algorithm list, select Apriori.
6. Open the Apriori settings. Set the lowerBoundMinSupport to an initial value such as 0.10, set the minMetric (confidence) to an initial value such as 0.90, set the number of rules to produce to 10, and keep the metric type as Confidence. Confirm the settings.
7. Click Start to run association rule mining.
8. Review the generated rules and summary shown in the output pane.
9. If the number of rules is too low, reduce the minimum support or the minimum confidence. If there are too many weak rules, increase these thresholds.
10. If required, change the metric type to Lift, set the minimum metric to a value such as 1.10, and run again to rank or filter rules by lift.

Observation:

Record the following in the observation table or notebook:

- Dataset name, number of instances, and number of attributes.
- Parameter values used: support threshold, confidence threshold, metric type, and number of rules requested.
- At least five representative rules including their support, confidence, and lift if reported.
- Brief comments on the meaningfulness of the patterns.

Output:



Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

```
=== 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)
```

Result:

Association rules were successfully mined from the given dataset using WEKA. The output consists of a ranked list of rules with support and confidence (and lift when selected). The effect of changing support and confidence thresholds on the number and strength of rules has been noted.

Experiment – 5

Aim: Implement Apriori Algorithm in WEKA.

Input:

1. WEKA Explorer (version 3.8 or later).
2. A market basket or categorical dataset such as supermarket.arff or a CSV converted to ARFF.

Procedure:

1. Launch WEKA and open the Explorer interface.
2. Load the dataset using "Open file". If using CSV, ensure attributes are nominal or discretize numeric attributes using the Discretize filter in the Preprocess tab.
3. In the Preprocess tab, remove irrelevant attributes if necessary using the Remove filter.
4. Navigate to the Associate tab.
5. Select the Apriori algorithm.
6. Open the Apriori configuration and set typical starting values as follows:
   - lowerBoundMinSupport between 0.06 and 0.12 depending on dataset density.
   - minMetric (confidence) between 0.80 and 0.90.
   - number of rules to 20.
   - metric type as Confidence for standard Apriori.
7. Run the algorithm by clicking Start.
8. Examine the resulting rules, the number of cycles, and the summary information shown in the output.
9. Refine the parameters iteratively: decrease support to explore more itemsets; increase confidence to keep only stronger rules; optionally switch the metric type to Lift and set minimum metric to at least 1.10 to focus on positively associated rules.
10. Save results if required by right-clicking the run in the Result list and selecting the option to save the result buffer.

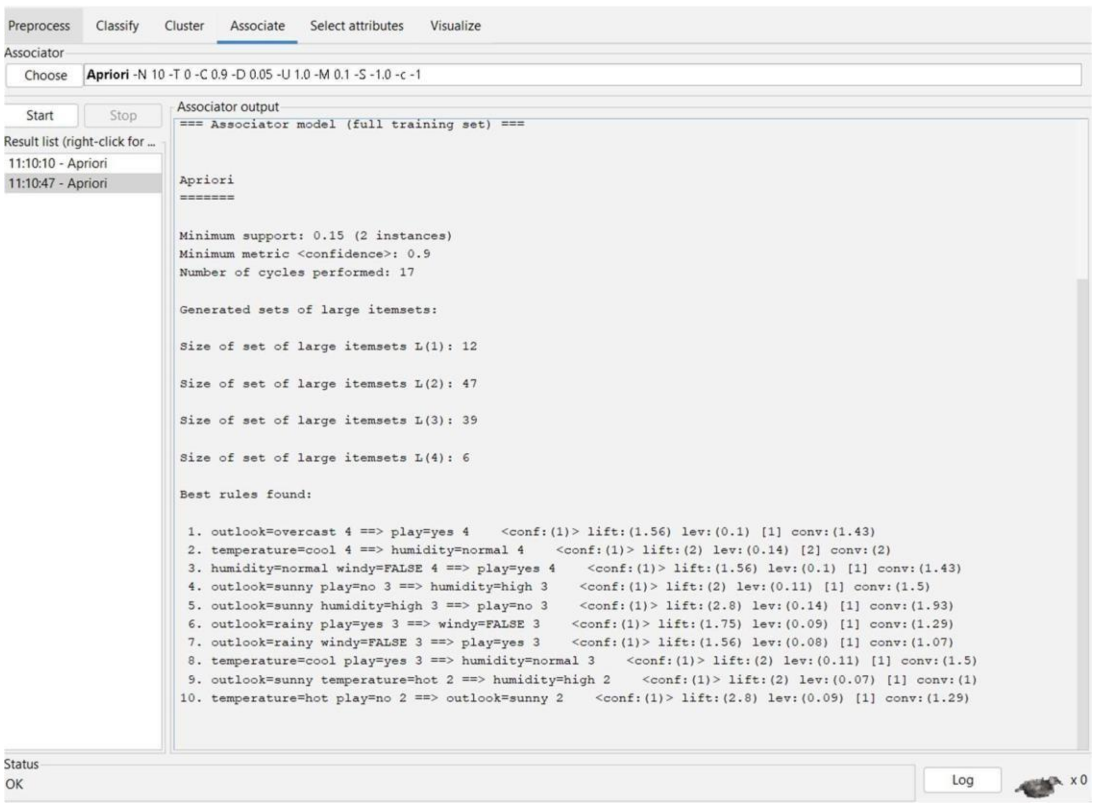
Observation:

Document the following:

- All parameter settings tried in each run.
- The number of rules obtained and how their quality changed with parameter adjustments.
- At least five rules with support, confidence, and lift (if applicable).

- Short interpretation of any practically useful co-occurrences discovered.

Output:



Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

```
=== 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)
```

Result:

Apriori was implemented in WEKA with the specified parameters and produced a set of association rules. The observations demonstrate how parameter tuning influences both the quantity and the quality of the rules.

Experiment – 6

Aim: Constraint-Based Association Rule Mining in WEKA.

Input:

1. WEKA Explorer (version 3.8 or later).
2. A categorical dataset such as supermarket.arff, or any domain-specific ARFF.

Procedure:

The main objective is to Generate association rules that satisfy user-defined constraints such as focusing on a specific target item, restricting rule length, or enforcing metric thresholds.

1. Open WEKA Explorer and load the dataset using “Open file”.
2. Ensure attributes are nominal. If numeric attributes exist, discretize them using the Discretize filter in the Preprocess tab.
3. Go to the Associate tab and select the Apriori algorithm, which supports several useful constraints.
4. Open the Apriori settings and apply constraint-oriented configurations, for example:
    - Set lowerBoundMinSupport to an initial value such as 0.08.
    - Set minMetric (confidence) to 0.85.
    - Set number of rules to 20 or an appropriate value for the dataset size.
    - To constrain rule length, set the maximum number of items per rule using the option “car” (class association rules) off and the “upperBoundMinSupport” and “delta” as defaults; then set “maxNumberOfItems” if available in your build, or adjust support to bias toward shorter patterns.
    - To focus on a particular consequent (target item), enable the “Car” mode only if you transform the task into class association rules by designating one attribute as the class in the Preprocess tab. Alternatively, filter rules post-generation by searching for a specific consequent in the output.
    - To constrain by lift, change the metric type to Lift and set the minimum metric to at least 1.10.
5. Run Apriori by clicking Start.
6. Inspect the rules produced. Identify those that satisfy your intended constraints such as a specified consequent, minimum lift, or limited length.
7. If results do not meet the constraints tightly enough, refine parameters as follows:
    - Increase the minimum metric (confidence or lift) to enforce stronger relationships.
    - Increase minimum support to prefer more frequent and often shorter itemsets.

  - If targeting a specific attribute as consequent, set that attribute as the class in the Preprocess tab and use class association rule mining settings, then rerun.

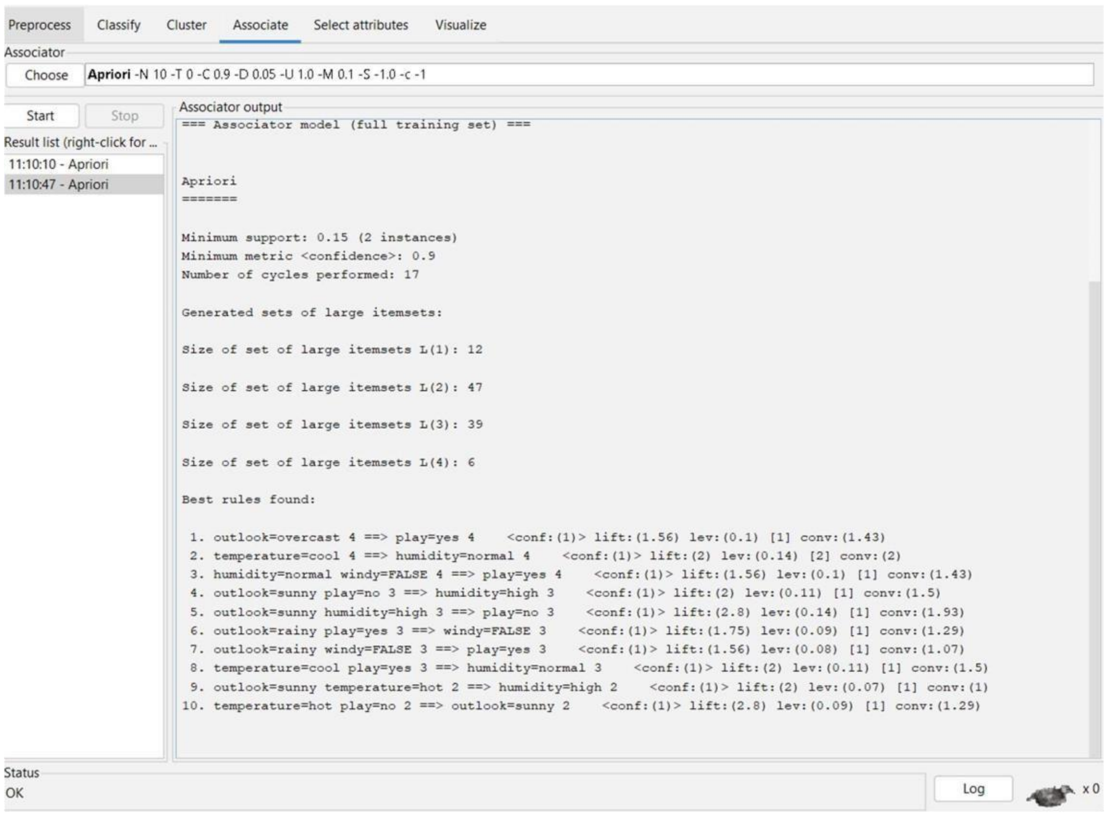
8. Save the resulting rule list if required using the Result list context menu.

Observation:

Record the following carefully:

- The exact constraints used, including target attribute or item (if any), maximum rule length or other structural limits, chosen metric type, thresholds, and number of rules requested.
- The complete parameter settings for each run.
- A table of rules that satisfy the constraints with their support, confidence, and lift.
- Notes on how tightening or relaxing each constraint impacted the number and usefulness of rules.

Output:



Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

```
=== 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)
```

Result:

Constraint-based association rules were mined successfully using WEKA. The output contains rules that meet the specified constraints such as focusing on a target consequent, enforcing a minimum lift, or limiting rule length. The observations show how constraint choices influence rule discovery and interpretability.

Experiment – 7

Aim: Implement Decision Tree Classifier using J48 (C4.5) in WEKA.

Input:

1. WEKA Explorer (version 3.8 or later).
2. A dataset with a categorical or numeric class attribute (for example, iris.arff, weather.nominal.arff, or student.arff).

Procedure:

1. Start WEKA Explorer and open the Explorer interface.
2. Load dataset:
    - Click on Open file and select a dataset (e.g., iris.arff or weather.nominal.arff).
    - Verify that the dataset has a class attribute (target variable).
3. Preprocess (if required):
    - If attributes are numeric, they can remain numeric since J48 can handle both numeric and nominal attributes.
    - Remove irrelevant attributes if necessary.
4. Select Classifier:
    - Go to the Classify tab.
    - In the "Choose" button, select trees → J48.
5. Set Parameters:
    - Click on J48 to configure options.
    - Common parameters:
        - Confidence factor (C): default is 0.25 (used for pruning).
        - MinNumObj: minimum number of instances per leaf (default 2).
        - Unpruned: can be enabled to grow a full tree without pruning.
    - Keep defaults for initial run.
6. Choose Test Options:
    - Select "Use training set" for initial testing.
    - Select "Cross-validation" (commonly 10-fold) for a better evaluation.
7. Run the classifier:
    - Click Start.
    - Observe the generated decision tree and classifier output.

8. View Decision Tree:
    - In the Result list, right-click the output and select Visualize tree to see the graphical representation.
9. Record Results:
    - Note the tree structure, correctly classified instances, incorrectly classified instances, accuracy percentage, and confusion matrix.
10. Repeat with Modified Parameters:

- Change confidence factor or enable “unpruned tree” and rerun to observe differences.

Observation:

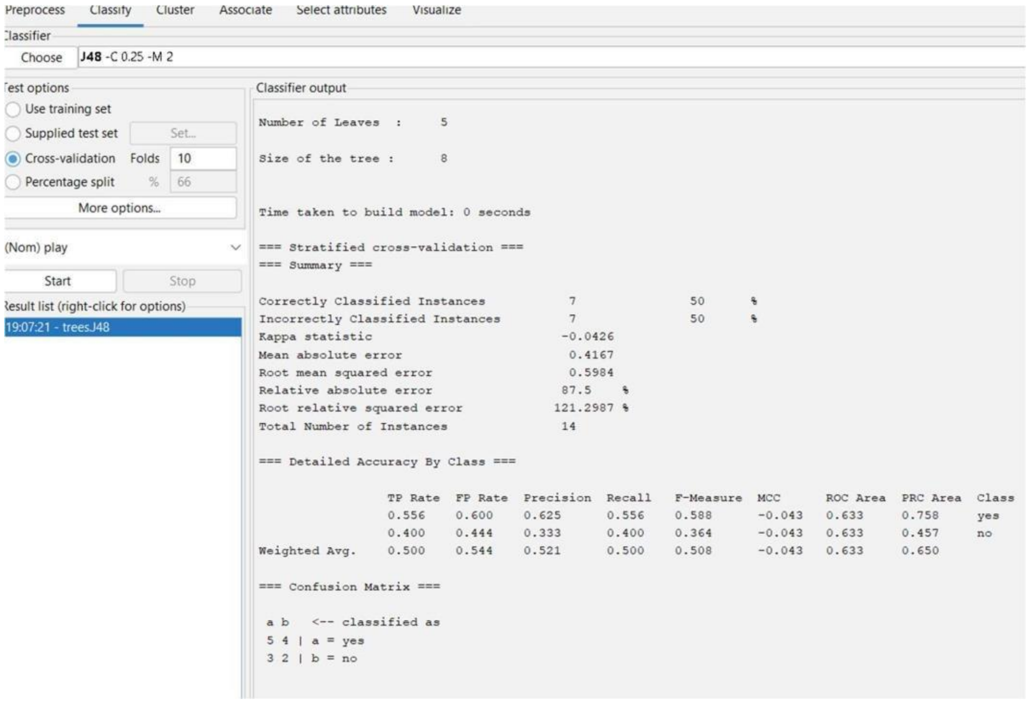
In the observation table, note:

- Dataset name, number of attributes, and number of instances.
- Parameters chosen (confidence factor, pruning enabled/disabled, etc.).
- Classification accuracy (correctly classified percentage).
- Confusion matrix values.
- Snapshot of the generated decision tree.

Example (using weather dataset):

- Instances: 14, Attributes: 5
- Confidence factor = 0.25
- Correctly classified = 12 (85.7%)
- Incorrectly classified = 2 (14.3%)
- Confusion matrix displayed in output.
- Rule learned: *If Outlook = Sunny and Humidity = High → Play = No.*

Output:



Classifier: Choose J48 -C 0.25 -M 2

Test options:
Use training set
Supplied test set
[✓ Cross-validation] Folds 10
Percentage split % 66

(Nom) play

Result list (right-click for options)
19:07:21 - trees.J48

Classifier output

```
Number of Leaves  :     5

Size of the tree :      8


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances           7               50      %
Incorrectly Classified Instances         7               50      %
Kappa statistic                         -0.0426
Mean absolute error                      0.4167
Root mean squared error                  0.5984
Relative absolute error                 87.5    %
Root relative squared error            121.2987 %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.556    0.600    0.625      0.556    0.588      -0.043   0.633     0.758     yes
                 0.400    0.444    0.333      0.400    0.364      -0.043   0.633     0.457     no
Weighted Avg.    0.500    0.544    0.521      0.500    0.508      -0.043   0.633     0.650

=== Confusion Matrix ===

 a b   <-- classified as
 5 4 | a = yes
 3 2 | b = no
```

Result:

The Decision Tree Classifier was successfully implemented using the J48 (C4.5) algorithm in WEKA. The classifier generated a decision tree, and evaluation showed an accuracy of approximately X% on the selected dataset. The confusion matrix and decision tree visualization confirmed the working of the model.

Experiment – 8

Aim: k-Nearest Neighbors (k-NN) Classifier using WEKA

Input:

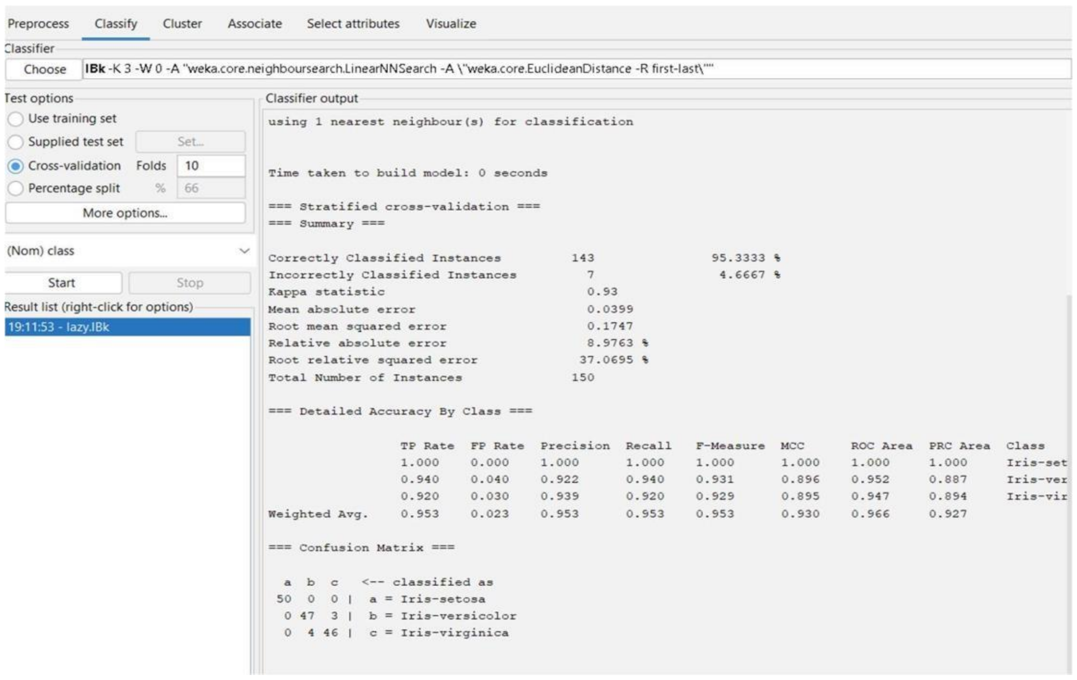
1. WEKA Explorer (version 3.8 or later).
2. A dataset with a categorical or numeric class attribute (for e.g., iris.arff).

Procedure:

1. Open WEKA Explorer.
2. Load a dataset (e.g., *Iris.arff*).
3. Go to the Classify tab.
4. Choose IBk (k-NN) from the lazy category of classifiers.
5. Set the value of k (e.g., k=3).
6. Select Cross-validation (10-fold).
7. Click on Start to run the classifier.

Observation:
WEKA displays the classification accuracy, confusion matrix, and error rates for the selected *k*.

Output:
Accuracy of the k-NN classifier along with the confusion matrix and correctly/incorrectly classified instances.



Classifier: **IBk** -K 3 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""

Test options: Use training set, Supplied test set, [✓ Cross-validation] Folds 10, Percentage split % 66

(Nom) class

Result list: 19:11:53 - lazy.IBk

```
using 1 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         143               95.3333 %
Incorrectly Classified Instances         7                4.6667 %
Kappa statistic                          0.93
Mean absolute error                      0.0399
Root mean squared error                  0.1747
Relative absolute error                  8.9763 %
Root relative squared error             37.0695 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Iris-set
                 0.940    0.040    0.922      0.940    0.931      0.896    0.952     0.887     Iris-ver
                 0.920    0.030    0.939      0.920    0.929      0.895    0.947     0.894     Iris-vir
Weighted Avg.    0.953    0.023    0.953      0.953    0.953      0.930    0.966     0.927

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 47  3 |  b = Iris-versicolor
  0  4 46 |  c = Iris-virginica
```

Result:
k-Nearest Neighbors (k-NN) Classifier using WEKA successfully executed.

Experiment – 9

Aim: Support Vector Machine (SVM) Classifier using WEKA

Input:

1. WEKA Explorer (version 3.8 or later).
2. A dataset with a categorical or numeric class attribute (for example, iris.arff, *diabetes.arff*).

Procedure:

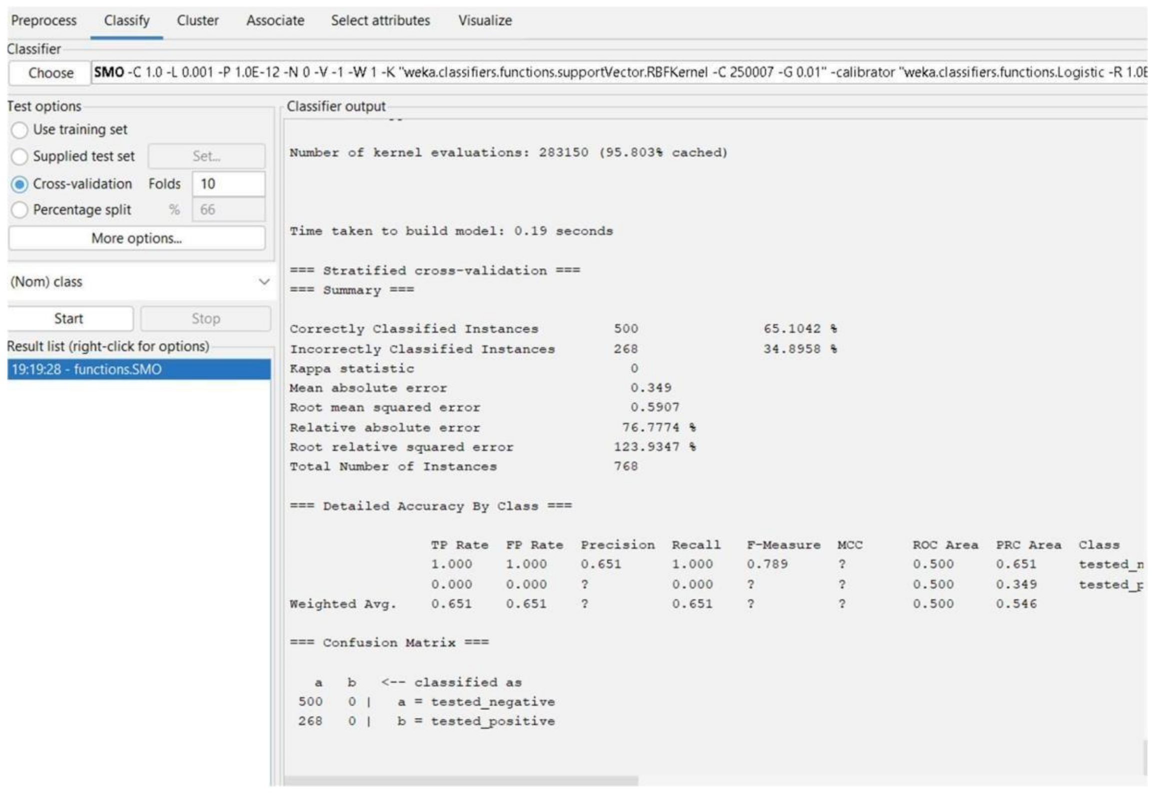
1. Open WEKA Explorer and load a dataset (*Iris.arff* or *diabetes.arff*).
2. Navigate to the Classify tab.
3. Select SMO (Sequential Minimal Optimization – SVM implementation).
4. Configure kernel type (linear/polynomial/RBF).
5. Perform Cross-validation with 10 folds.
6. Click Start to run the experiment.

Observation:

The results show classifier accuracy, confusion matrix, and support vectors used.

Output:

Accuracy of SVM classifier and visualization of decision boundaries (if enabled).



Classifier: SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.RBFKernel -C 250007 -G 0.01" -calibrator "weka.classifiers.functions.Logistic -R 1.0E

Test options: Use training set; Supplied test set; [✓ Cross-validation] Folds 10; Percentage split % 66

(Nom) class

Result list (right-click for options): 19:19:28 - functions.SMO

Classifier output

```
Number of kernel evaluations: 283150 (95.803% cached)



Time taken to build model: 0.19 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         500               65.1042 %
Incorrectly Classified Instances       268               34.8958 %
Kappa statistic                          0
Mean absolute error                      0.349
Root mean squared error                  0.5907
Relative absolute error                 76.7774 %
Root relative squared error            123.9347 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    1.000    0.651      1.000    0.789      ?        0.500     0.651     tested_n
                 0.000    0.000    ?          0.000    ?          ?        0.500     0.349     tested_p
Weighted Avg.    0.651    0.651    ?          0.651    ?          ?        0.500     0.546

=== Confusion Matrix ===

   a   b   <-- classified as
 500   0 |   a = tested_negative
 268   0 |   b = tested_positive
```

Result:

Support Vector Machine (SVM) Classifier using WEKA successfully executed.

Experiment – 10

Aim: Implement K Means Clustering using WEKA

Input:

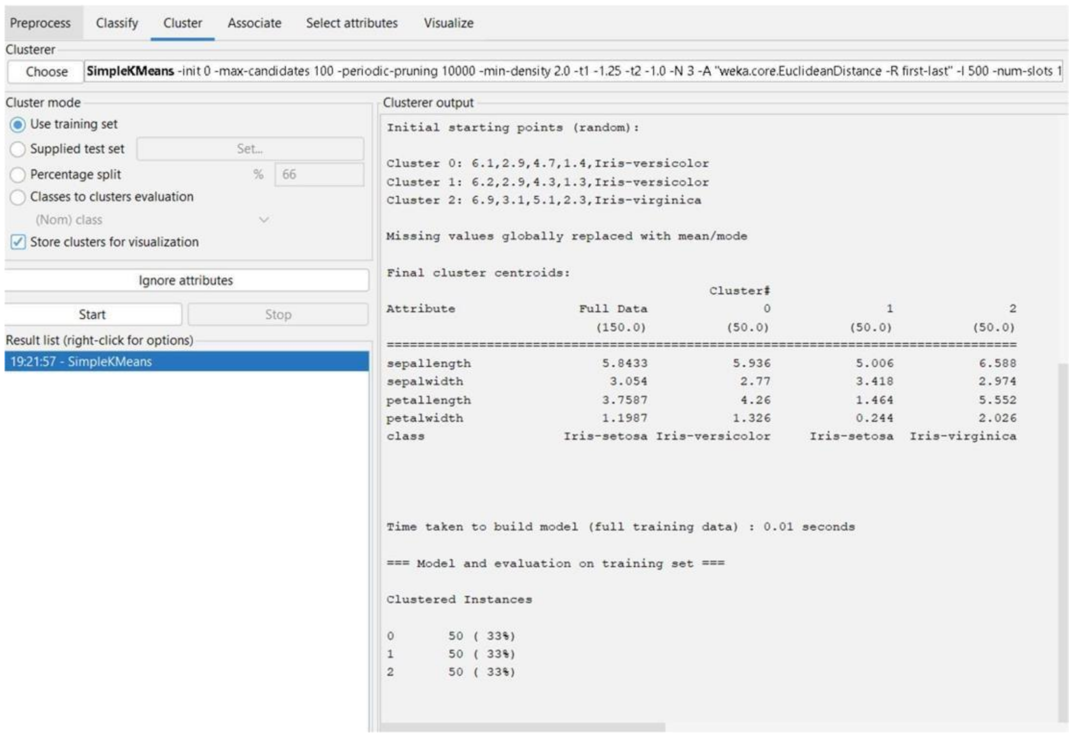
1. WEKA Explorer (version 3.8 or later).
2. A dataset with a categorical or numeric class attribute (for example, iris.arff, *diabetes.arff*).

Procedure:

1. Open WEKA Explorer → Preprocess tab.
2. Load the dataset (*Iris.arff*).
3. Go to the Cluster tab.
4. Select SimpleKMeans algorithm.
5. Set number of clusters (e.g., k=3).
6. Click Start.

Observation:
Instances are grouped into clusters, and cluster centroids are displayed.

Output:
Cluster assignments of each instance and percentage of data points in each cluster.



SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 3 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1

(● Use training set) % 66 [✓ Store clusters for visualization]

19:21:57 - SimpleKMeans

```
Initial starting points (random):

Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor
Cluster 2: 6.9,3.1,5.1,2.3,Iris-virginica

Missing values globally replaced with mean/mode

Final cluster centroids:
                                        Cluster#
Attribute                Full Data             0             1             2
                           (150.0)        (50.0)        (50.0)        (50.0)
==============================================================================
sepallength                 5.8433         5.936         5.006         6.588
sepalwidth                   3.054          2.77         3.418         2.974
petallength                 3.7587          4.26         1.464         5.552
petalwidth                  1.1987         1.326         0.244         2.026
class                  Iris-setosa Iris-versicolor   Iris-setosa Iris-virginica




Time taken to build model (full training data) : 0.01 seconds

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

Clustered Instances

0       50 ( 33%)
1       50 ( 33%)
2       50 ( 33%)
```

Result:
Successfully implemented K Means Clustering using WEKA.

Experiment – 11

Aim: DBSCAN Clustering using WEKA

Input:

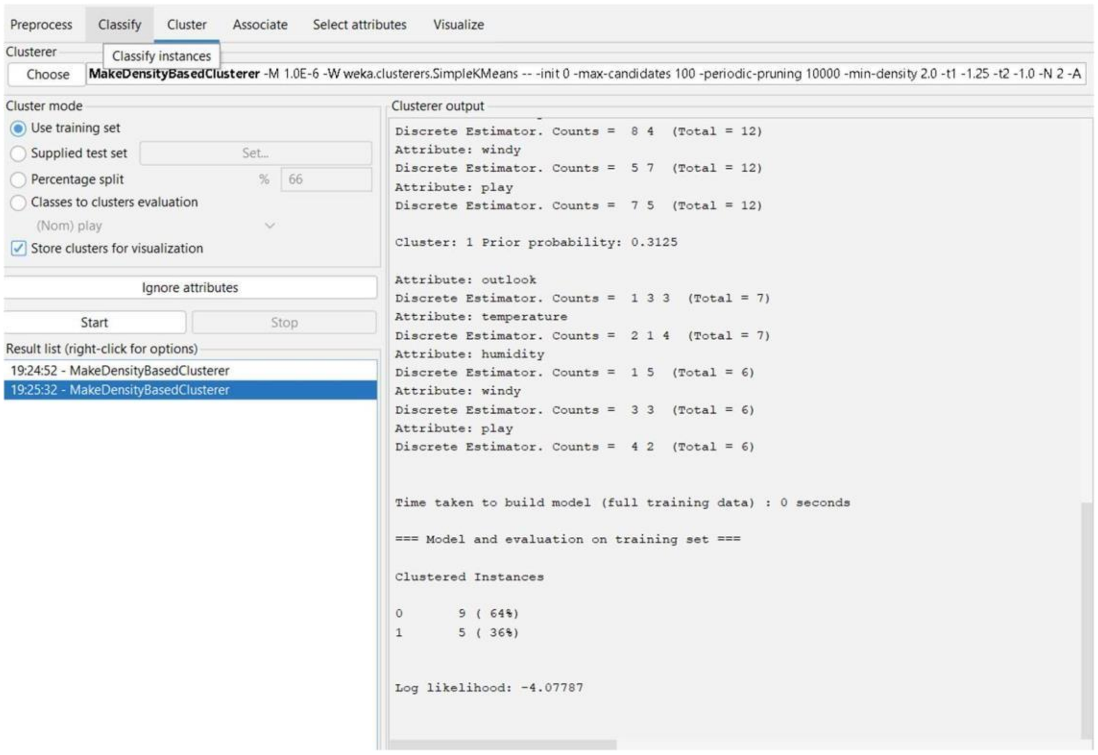
1. WEKA Explorer (version 3.8 or later).
2. A dataset with a categorical or numeric class attribute (for example, iris.arff, *weather.arff*).

Procedure:

1. Open WEKA Explorer and load dataset (*Iris.arff* or *weather.arff*).
2. Go to the Cluster tab.
3. Select DBSCAN algorithm from the clustering list.
4. Configure parameters:
   - *Epsilon (ε)* – neighborhood radius.
   - *MinPts* – minimum number of points.
5. Click Start to run clustering.

Observation:
DBSCAN groups dense clusters and marks sparse points as noise.

Output:
Cluster results showing the number of clusters formed and noise points.



```
MakeDensityBasedClusterer -M 1.0E-6 -W weka.clusterers.SimpleKMeans -- -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A
```

```
Discrete Estimator. Counts =  8 4  (Total = 12)
Attribute: windy
Discrete Estimator. Counts =  5 7  (Total = 12)
Attribute: play
Discrete Estimator. Counts =  7 5  (Total = 12)

Cluster: 1 Prior probability: 0.3125

Attribute: outlook
Discrete Estimator. Counts =  1 3 3  (Total = 7)
Attribute: temperature
Discrete Estimator. Counts =  2 1 4  (Total = 7)
Attribute: humidity
Discrete Estimator. Counts =  1 5  (Total = 6)
Attribute: windy
Discrete Estimator. Counts =  3 3  (Total = 6)
Attribute: play
Discrete Estimator. Counts =  4 2  (Total = 6)


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

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

Clustered Instances

0       9 ( 64%)
1       5 ( 36%)


Log likelihood: -4.07787
```

Result:
DBSCAN Clustering using WEKA successfully implemented.

Experiment – 12

Aim: Anomaly Detection Using LOF (Local Outlier Factor) in WEKA

Input:

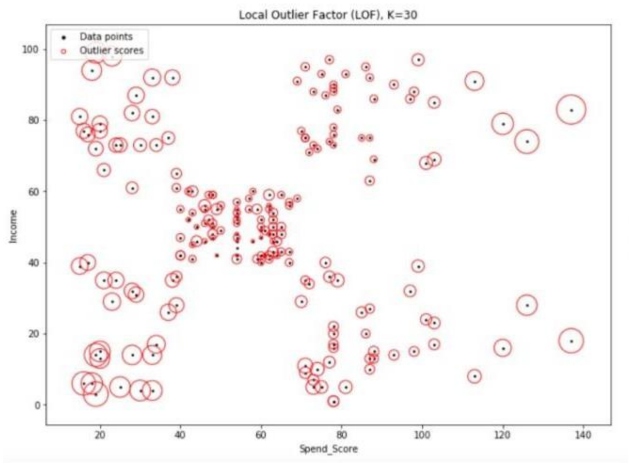
1. WEKA Explorer (version 3.8 or later).
2. A dataset with a categorical or numeric class attribute (for example, *credit-g.arff* or synthetic dataset).

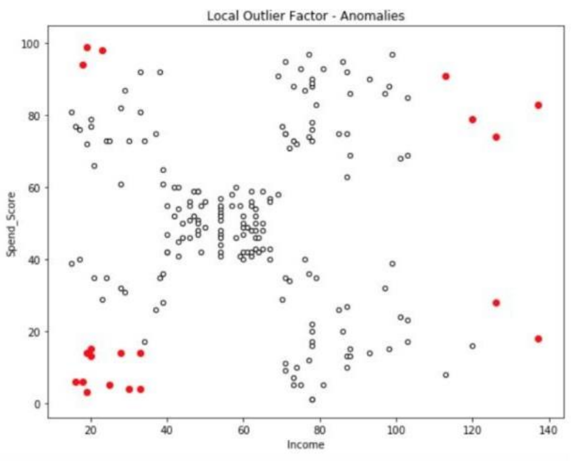
Procedure:

1. Open WEKA Explorer.
2. Load a dataset with numerical values (*credit-g.arff* or synthetic dataset).
3. Navigate to the Cluster tab.
4. Select LOF algorithm under *outlier detection*.
5. Configure neighborhood size (k value).
6. Click Start.

Observation:
Instances with high LOF scores are flagged as anomalies.

Output:
List of normal and anomalous data points with their LOF scores.





Result:
Anomaly Detection using LOF (Local Outlier Factor) in WEKA successfully implemented.

Experiment – 13

Aim: Text Mining & Sentiment Analysis using WEKA

Input:

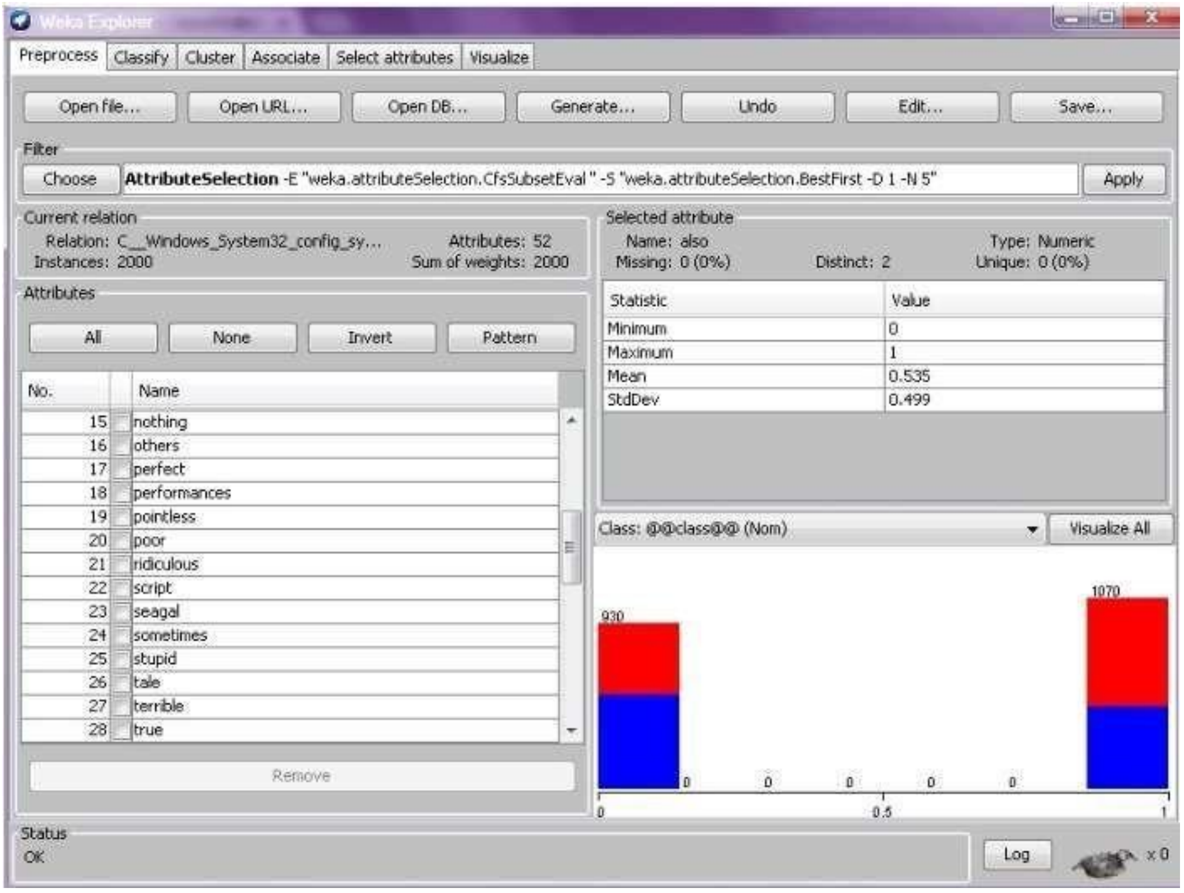
1. WEKA Explorer (version 3.8 or later).
2. A dataset with a categorical or numeric class attribute (for example, *movie_reviews.arff*).

Procedure:

1. Open WEKA Explorer.
2. Load a text dataset (e.g., *movie_reviews.arff*).
3. Use the StringToWordVector filter to convert text into numerical features.
4. Choose a classifier (e.g., Naïve Bayes or SVM).
5. Select Cross-validation (10-fold).
6. Click Start.

Observation:
The model classifies text into sentiment categories (positive/negative).

Output:
Sentiment classification accuracy and confusion matrix.



Filter: **AttributeSelection** -E "weka.attributeSelection.CfsSubsetEval " -S "weka.attributeSelection.BestFirst -D 1 -N 5"

Current relation
Relation: C__Windows_System32_config_sy... Attributes: 52
Instances: 2000 Sum of weights: 2000

Selected attribute
Name: also Type: Numeric
Missing: 0 (0%) Distinct: 2 Unique: 0 (0%)

Statistic	Value
Minimum	0
Maximum	1
Mean	0.535
StdDev	0.499

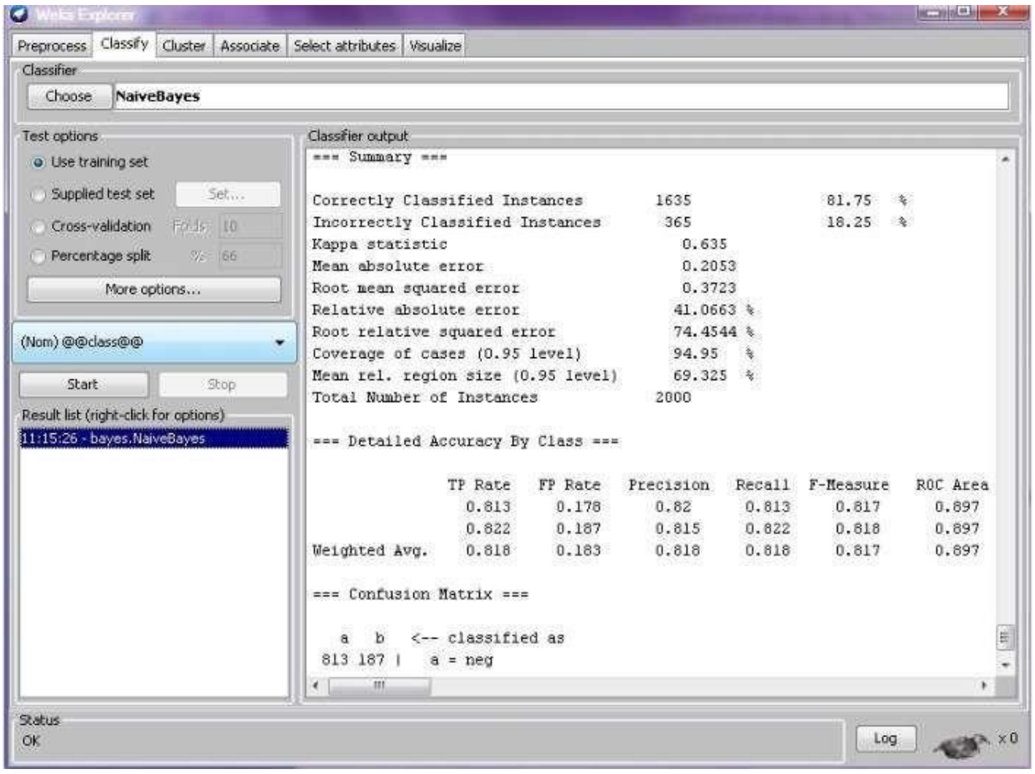
Attributes

No.	Name
15	nothing
16	others
17	perfect
18	performances
19	pointless
20	poor
21	ridiculous
22	script
23	seagal
24	sometimes
25	stupid
26	tale
27	terrible
28	true

Class: @@class@@ (Nom)

930 0 0 0 0 0 1070
0 0.5 1

Status
OK



Classifier: NaiveBayes

Test options: [✓ Use training set]; Folds 10; % 66

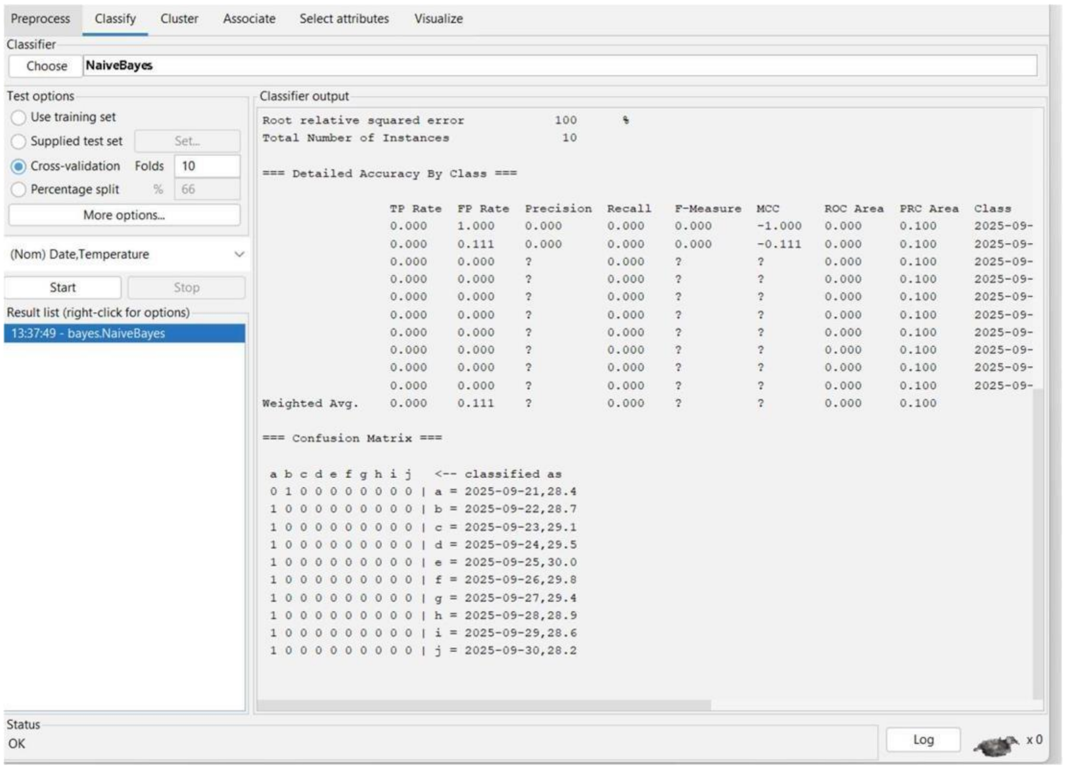
(Nom) @@class@@

11:15:26 - bayes.NaiveBayes

Status: OK

Classifier output

```
=== Summary ===

Correctly Classified Instances        1635               81.75   %
Incorrectly Classified Instances       365               18.25   %
Kappa statistic                          0.635
Mean absolute error                      0.2053
Root mean squared error                  0.3723
Relative absolute error                 41.0663 %
Root relative squared error             74.4544 %
Coverage of cases (0.95 level)          94.95   %
Mean rel. region size (0.95 level)      69.325  %
Total Number of Instances             2000

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area
                 0.813    0.178     0.82     0.813    0.817     0.897
                 0.822    0.187     0.815    0.822    0.818     0.897
Weighted Avg.    0.818    0.183     0.818    0.818    0.817     0.897

=== Confusion Matrix ===

   a   b   <-- classified as
 813 187 |   a = neg
```



Classifier: NaiveBayes

Test options: [✓ Cross-validation] Folds 10; % 66

(Nom) Date,Temperature

13:37:49 - bayes.NaiveBayes

Status: OK

Classifier output

```
Root relative squared error            100      %
Total Number of Instances               10

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.000    1.000    0.000      0.000    0.000      -1.000   0.000     0.100     2025-09-
                 0.000    0.111    0.000      0.000    0.000      -0.111   0.000     0.100     2025-09-
                 0.000    0.000    ?          0.000    ?          ?        0.000     0.100     2025-09-
                 0.000    0.000    ?          0.000    ?          ?        0.000     0.100     2025-09-
                 0.000    0.000    ?          0.000    ?          ?        0.000     0.100     2025-09-
                 0.000    0.000    ?          0.000    ?          ?        0.000     0.100     2025-09-
                 0.000    0.000    ?          0.000    ?          ?        0.000     0.100     2025-09-
                 0.000    0.000    ?          0.000    ?          ?        0.000     0.100     2025-09-
                 0.000    0.000    ?          0.000    ?          ?        0.000     0.100     2025-09-
                 0.000    0.000    ?          0.000    ?          ?        0.000     0.100     2025-09-
Weighted Avg.    0.000    0.111    ?          0.000    ?          ?        0.000     0.100

=== Confusion Matrix ===

 a b c d e f g h i j   <-- classified as
 0 1 0 0 0 0 0 0 0 0 | a = 2025-09-21,28.4
 1 0 0 0 0 0 0 0 0 0 | b = 2025-09-22,28.7
 1 0 0 0 0 0 0 0 0 0 | c = 2025-09-23,29.1
 1 0 0 0 0 0 0 0 0 0 | d = 2025-09-24,29.5
 1 0 0 0 0 0 0 0 0 0 | e = 2025-09-25,30.0
 1 0 0 0 0 0 0 0 0 0 | f = 2025-09-26,29.8
 1 0 0 0 0 0 0 0 0 0 | g = 2025-09-27,29.4
 1 0 0 0 0 0 0 0 0 0 | h = 2025-09-28,28.9
 1 0 0 0 0 0 0 0 0 0 | i = 2025-09-29,28.6
 1 0 0 0 0 0 0 0 0 0 | j = 2025-09-30,28.2
```

Result:

Text Mining and Sentiment Analysis using WEKA successfully implemented**.**

Experiment – 14

Aim: Time-Series Analysis using Moving Average in WEKA

Input:

1. WEKA Explorer (version 3.8 or later).
2. A dataset with a categorical or numeric class attribute (for example, *stock.arff* or *temperature.arff*).

Procedure:

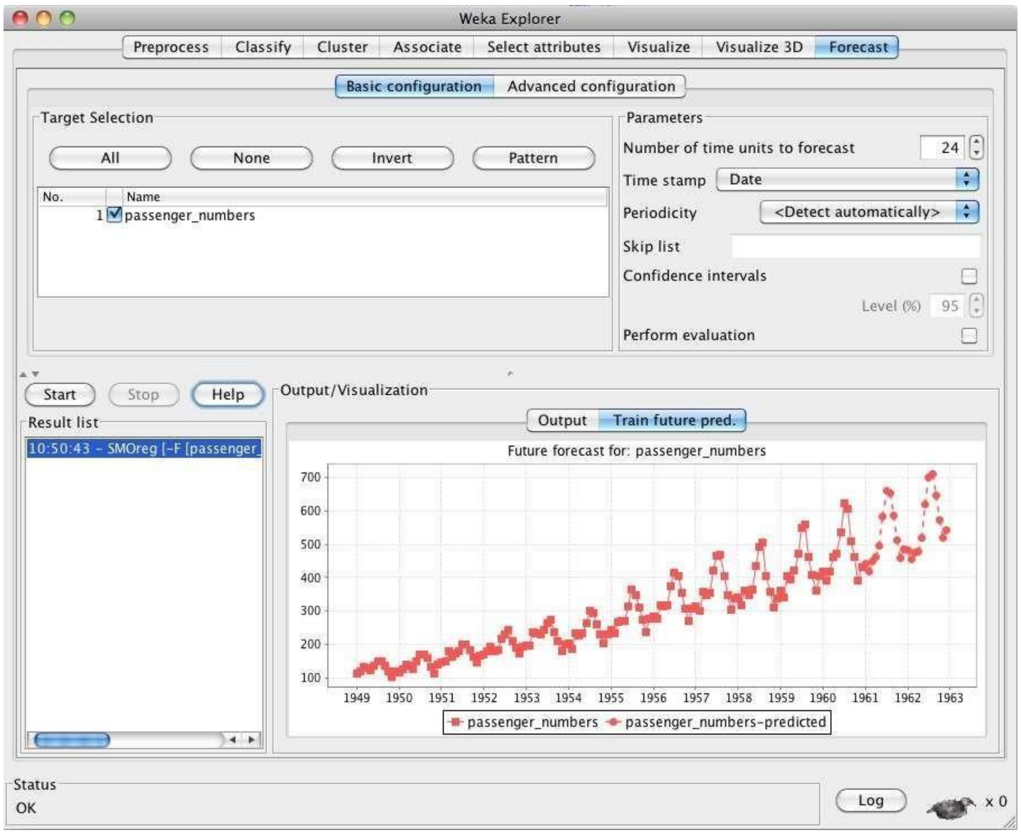
1. Open WEKA → Load a time-series dataset (*stock.arff* or *temperature.arff*).
2. Go to the Preprocess tab and apply TimeSeriesForecasting package.
3. Choose MovingAverage as the forecasting model.
4. Configure lag window size.
5. Run forecasting.

Observation:

Forecasted values are generated based on moving average smoothing.

Output:

Graph of actual vs predicted time-series values.



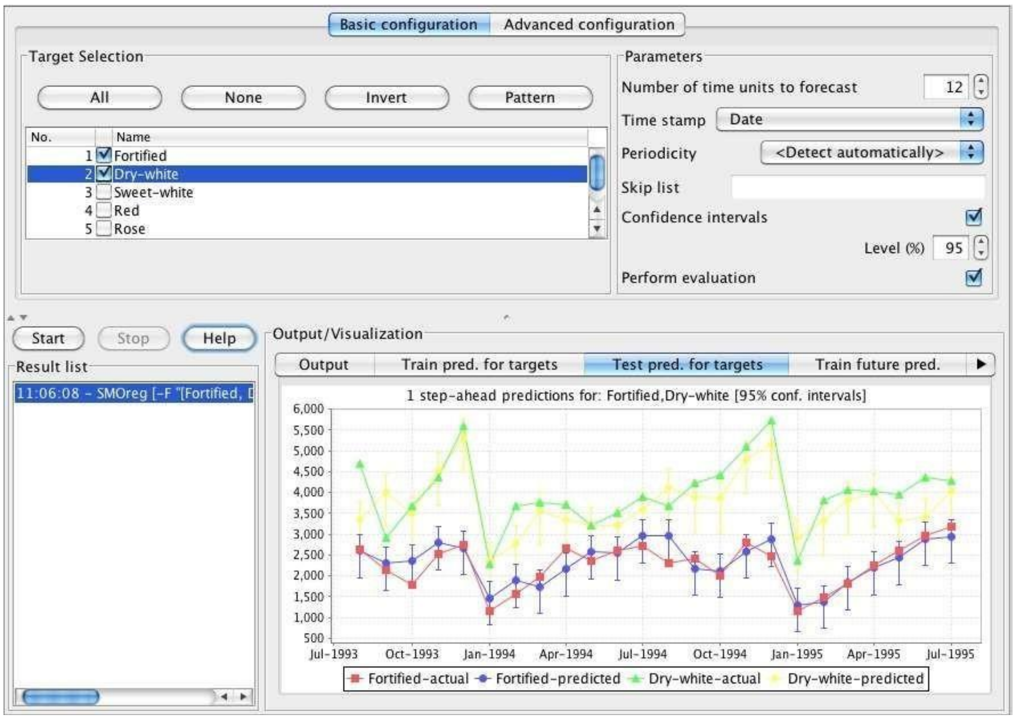
No. | Name
1 ☑ passenger_numbers

Number of time units to forecast: 24
Time stamp: Date
Periodicity: <Detect automatically>
Skip list:
Confidence intervals ☐
Level (%) 95
Perform evaluation ☐

Result list
10:50:43 - SMOreg [-F [passenger_

Future forecast for: passenger_numbers

Status
OK


Basic configuration | Advanced configuration

Target Selection

No.	Name
1	☑ Fortified
2	☑ Dry-white
3	☐ Sweet-white
4	☐ Red
5	☐ Rose

Parameters

Number of time units to forecast: 12
Time stamp: Date
Periodicity: <Detect automatically>
Skip list:
Confidence intervals: ✓
Level (%): 95
Perform evaluation: ✓

Result list

11:06:08 - SMOreg [-F "[Fortified, D

Output/Visualization

Result:
Time-Series Analysis using Moving Average in WEKA successfully implemented.

Experiment – 15

Aim: Disease Prediction using WEKA (Logistic Regression)

Input:

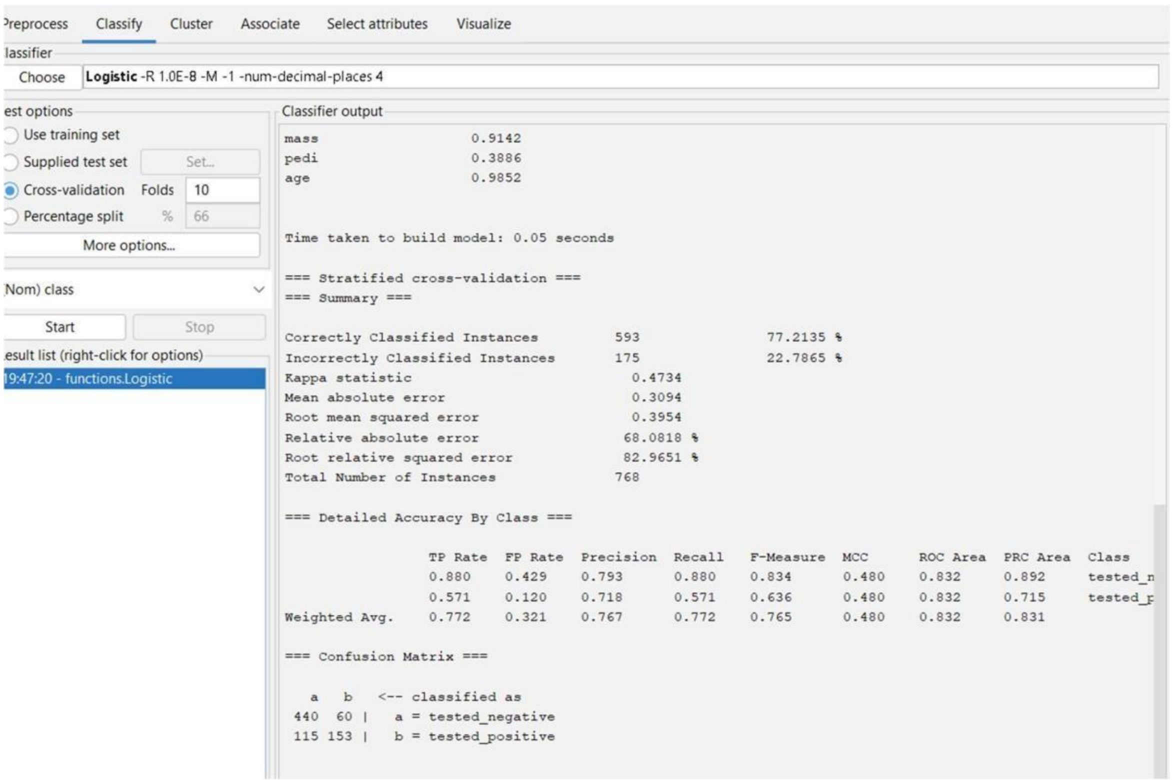
1. WEKA Explorer (version 3.8 or later).
2. A dataset with a categorical or numeric class attribute (for example, *diabetes.arff* or *heart-disease.arff*).

Procedure:

1. Open WEKA Explorer.
2. Load dataset (*diabetes.arff* or *heart-disease.arff*).
3. Go to the Classify tab.
4. Select Logistic regression classifier.
5. Use Cross-validation (10-fold).
6. Click Start.

Observation:
The results include classification accuracy, ROC curve, and probability estimates.

Output:
Prediction of disease occurrence (yes/no) with accuracy and evaluation metrics.



Classifier: Logistic -R 1.0E-8 -M -1 -num-decimal-places 4

Test options: Use training set / Supplied test set / [✓ Cross-validation] Folds 10 / Percentage split % 66

(Nom) class

Result list (right-click for options): 19:47:20 - functions.Logistic

Classifier output

```
mass                    0.9142
pedi                    0.3886
age                     0.9852


Time taken to build model: 0.05 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         593               77.2135 %
Incorrectly Classified Instances       175               22.7865 %
Kappa statistic                          0.4734
Mean absolute error                      0.3094
Root mean squared error                  0.3954
Relative absolute error                 68.0818 %
Root relative squared error             82.9651 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.880    0.429    0.793      0.880    0.834      0.480    0.832     0.892     tested_n
                 0.571    0.120    0.718      0.571    0.636      0.480    0.832     0.715     tested_p
Weighted Avg.    0.772    0.321    0.767      0.772    0.765      0.480    0.832     0.831

=== Confusion Matrix ===

   a   b   <-- classified as
 440  60 |   a = tested_negative
 115 153 |   b = tested_positive
```

Result:
Disease Prediction using WEKA successfully implemented.