

EXPERIMENT: 1

AIM: Demonstration of preprocessing on some datasets eg. Student.arff/
labor.arff/Iris/ loan/ etc.

DESCRIPTION:

Steps to get Started with WEKA:

1. Open C drive.
2. Then open program files in that go to weka-3-8
3. (NOTE: search for folder named weka-version-number).
4. Select Explorer.



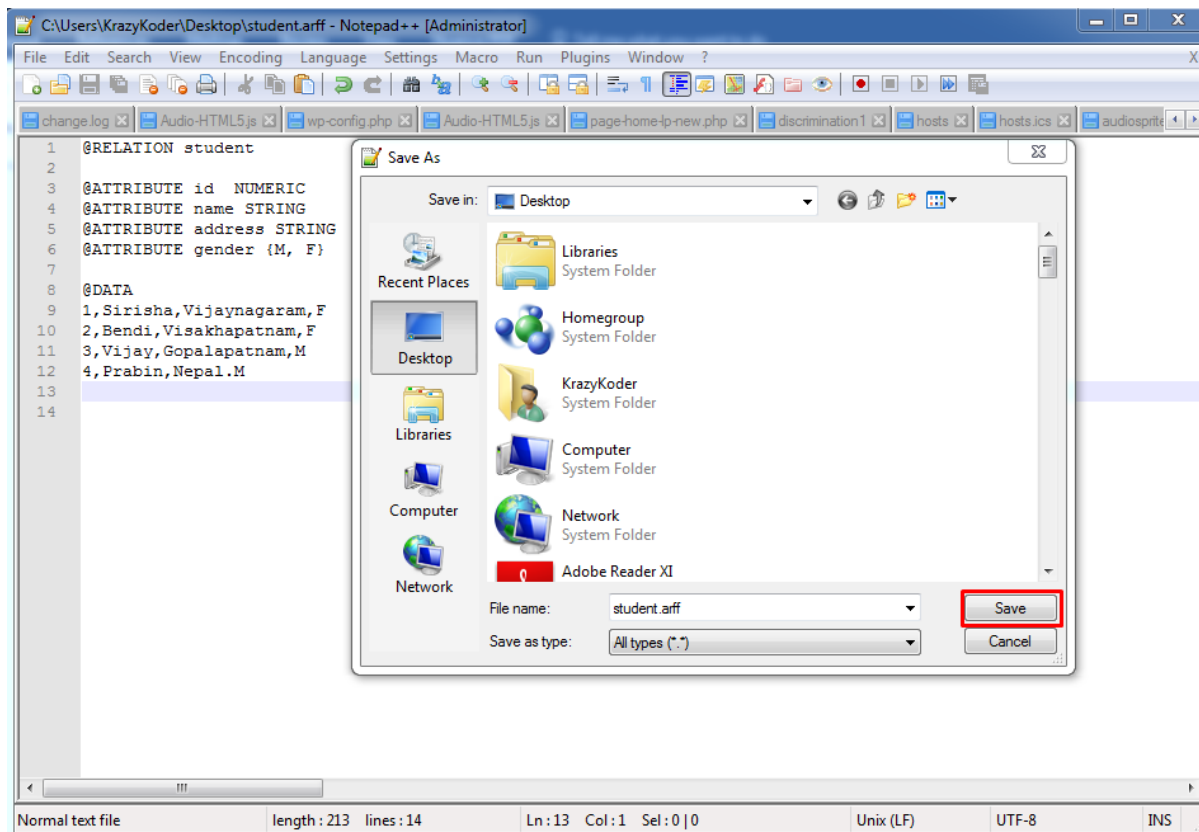
Steps To Create Own Data Set with weka attribute relation file format(.arff):

1. Open notepad
2. Type information as shown below in notepad;
 - @ RELATION = To make table
 - @ ATTRIBUTE = To define columns(Note: each attribute should contain @ ATTRIBUTE at beginning then instances/tuples are added by @ DATA (Note: each tuple in one row))

Example:

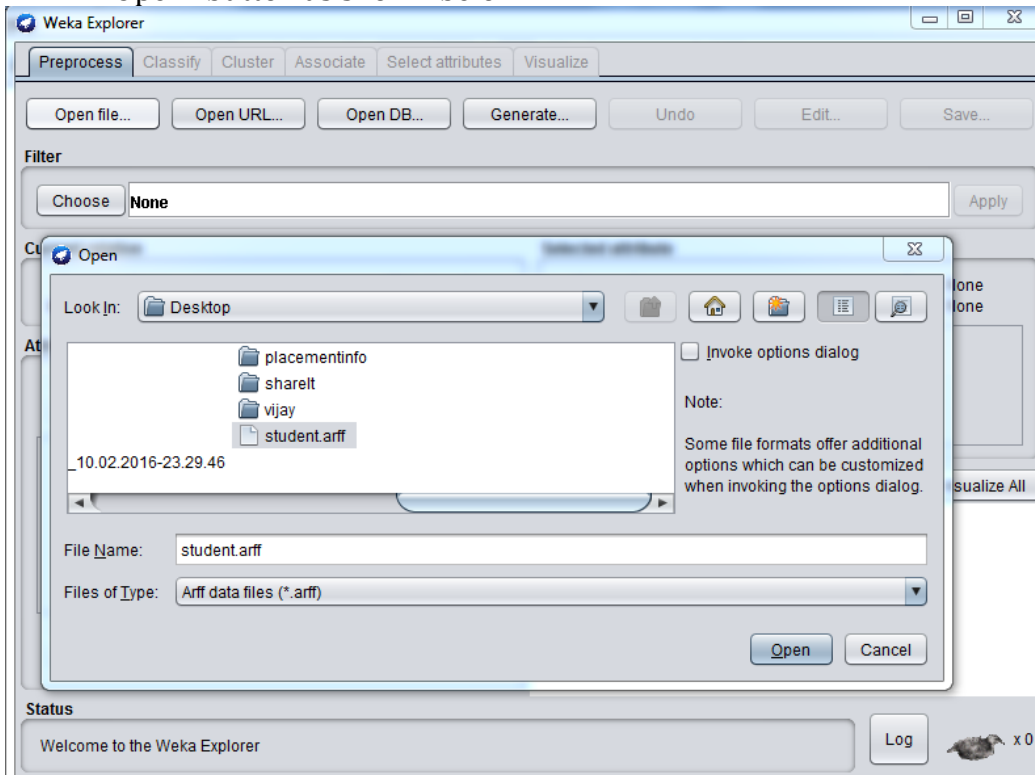
```
@RELATION student
@ATTRIBUTE id NUMERIC
@ATTRIBUTE name STRING
@ATTRIBUTE address STRING
@ATTRIBUTE gender {M, F}
@DATA
1,Sirisha,Vijaynagaram,F
2,Bendi,Visakhapatnam,F
3,Vijay,Gopalapatnam,M
4,Prabin,Nepal,M
```

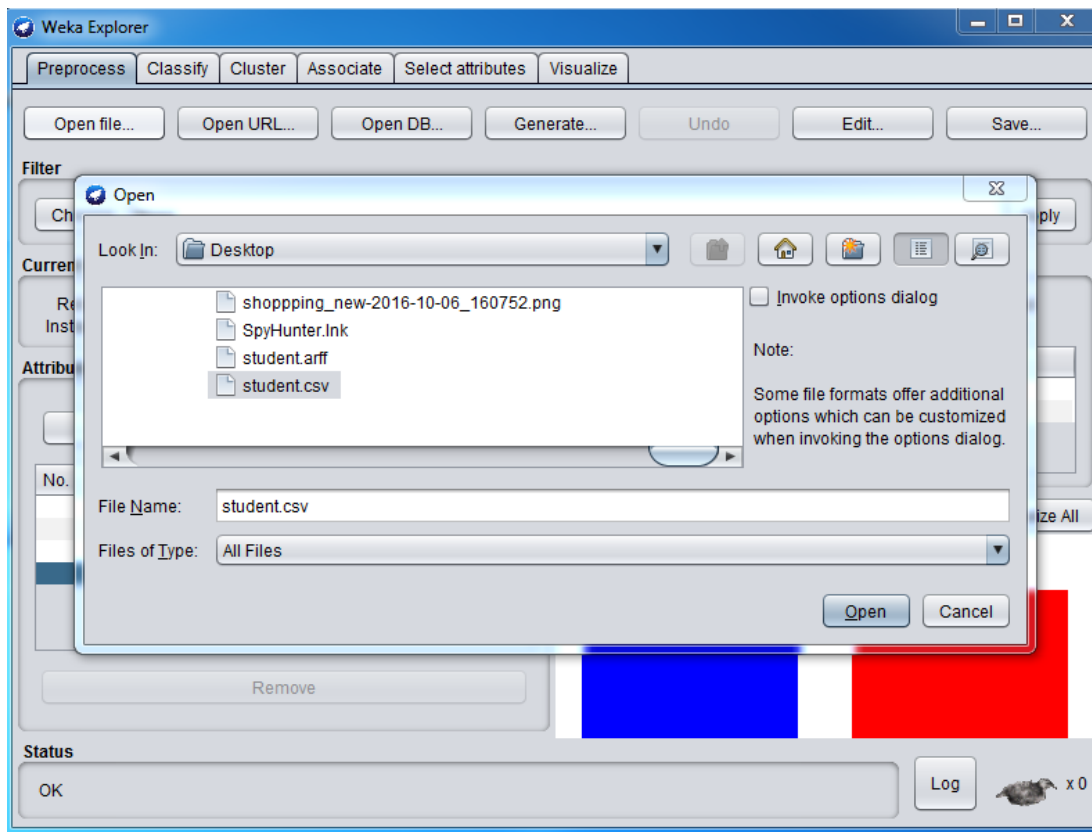
3. Save File with arff extension (Eg: **student.arff**) ;



Loading Dataset in weka (.arff or .csv):

1. Open weka Explorer
2. Click "Open File.." at top left
3. Select arff/csv file you want to load (Eg: student.arff/bank-data.csv) Then press "Open" button as shown below:





Note: The same process is continued for loading all other datasets in later cases.

IRIS DATA SET:

Number of Instances:150

Number of attributes:4

Missing Values?: No

@RELATION iris

@ATTRIBUTE sepallength REAL

@ATTRIBUTE sepalwidth REAL

@ATTRIBUTE petallength REAL

@ATTRIBUTE petalwidth REAL

@ATTRIBUTE class {Iris-setosa, Iris-versicolor, Iris-virginica}

@DATA

5.1, 3.5, 1.4, 0.2, Iris-setosa

4.9, 3.0, 1.4, 0.2, Iris-setosa

4.7, 3.2, 1.3, 0.2, Iris-setosa

4.6, 3.1, 1.5, 0.2, Iris-setosa

5.0, 3.6, 1.4, 0.2, Iris-setosa

7.0, 3.2, 4.7, 1.4, Iris-versicolor

6.4, 3.2, 4.5, 1.5, Iris-versicolor
6.9, 3.1, 4.9, 1.5, Iris-versicolor
5.5, 2.3, 4.0, 1.3, Iris-versicolor
6.5, 2.8, 4.6, 1.5, Iris-versicolor
6.3, 3.3, 6.0, 2.5, Iris-virginica
5.8, 2.7, 5.1, 1.9, Iris-virginica
7.1, 3.0, 5.9, 2.1, Iris-virginica
6.3, 2.9, 5.6, 1.8, Iris-virginica
6.5, 3.0, 5.8, 2.2, Iris-virginica

LABOR DATA SET:

Number of Instances:57

Number of attributes:17

Missing Values?: Yes

@RELATION 'labor-data'

@ATTRIBUTE 'duration' real

@ATTRIBUTE 'wage-increase-first-year' real

@ATTRIBUTE 'wage-increase-second-year' real

@ATTRIBUTE 'wage-increase-third-year' real

@ATTRIBUTE 'cost-of-living-adjustment' {'none','tcf','tc'}

@ATTRIBUTE 'working-hours' real

@ATTRIBUTE 'pension' {'none','ret_allw','empl_contr'}

@ATTRIBUTE 'standby-pay' real

@ATTRIBUTE 'shift-differential' real

@ATTRIBUTE 'education-allowance' {'yes','no'}

@ATTRIBUTE 'statutory-holidays' real

@ATTRIBUTE 'vacation' {'below_average','average','generous'}

@ATTRIBUTE 'longterm-disability-assistance' {'yes','no'}

@ATTRIBUTE 'contribution-to-dental-plan' {'none','half','full'}

@ATTRIBUTE 'bereavement-assistance' {'yes','no'}

@ATTRIBUTE 'contribution-to-health-plan' {'none','half','full'}

@ATTRIBUTE 'class' {'bad','good'}

@DATA

1,5,?,?,40,?,?,2,?,11,'average',?,?, 'yes',?, 'good'

2,4.5,5.8,?,?,35,'ret_allw',?,?, 'yes',11,'below_average',?, 'full',?, 'full', 'good'
?,?,?,?,38,'empl_contr',?,5,?,11,'generous','yes','half','yes','half','good'
3,3.7,4,5,'tc',?,?,?, 'yes',?,?,?, 'yes',?, 'good'
3,4.5,4.5,5,?,40,?,?,?,12,'average',?, 'half', 'yes', 'half', 'good'
2,2,2.5,?,?,35,?,?,6,'yes',12,'average',?,?,?, 'good'
3,4,5,5,'tc',?, 'empl_contr',?,?,?,12,'generous','yes','none','yes','half','good'
3,6.9,4.8,2.3,?,40,?,?,3,?,12,'below_average',?,?,?, 'good'
2,3,7,?,?,38,?,12,25,'yes',11,'below_average','yes','half','yes',?, 'good'
1,5.7,?,?, 'none',40,'empl_contr',?,4,?,11,'generous','yes','full',?,?, 'good'
3,3.5,4,4.6,'none',36,?,?,3,?,13,'generous',?,?, 'yes', 'full', 'good'
2,6.4,6.4,?,?,38,?,4,?,15,?,?, 'full',?,?, 'good'
2,3.5,4,?, 'none',40,?,?,2,'no',10,'below_average','no','half',?, 'half', 'bad'
3,3.5,4,5.1,'tcf',37,?,?,4,?,13,'generous',?, 'full', 'yes', 'full', 'good'
1,3,?,?, 'none',36,?,?,10,'no',11,'generous',?,?,?, 'good'
2,4.5,4,?, 'none',37,'empl_contr',?,?,?,11,'average',?, 'full', 'yes',?, 'good'
1,2.8,?,?,?,35,?,?,2,?,12,'below_average',?,?,?, 'good'
1,2.1,?,?, 'tc',40,'ret_allw',2,3,'no',9,'below_average','yes','half',?, 'none', 'bad'
1,2,?,?, 'none',38,'none',?,?, 'yes',11,'average','no','none','no','none', 'bad'
2,4,5,?, 'tcf',35,?,13,5,?,15,'generous',?,?,?, 'good'
2,4.3,4.4,?,?,38,?,4,?,12,'generous',?, 'full',?, 'full', 'good'
2,2.5,3,?,?,40,'none',?,?,?,11,'below_average',?,?,?, 'bad'
3,3.5,4,4.6,'tcf',27,?,?,?,?,?,?,?, 'good'
2,3.5,4,?, 'none',40,?,?,2,'no',10,'below_average','no','half',?, 'half', 'bad'
3,3.5,4,5.1,'tcf',37,?,?,4,?,13,'generous',?, 'full', 'yes', 'full', 'good'
1,3,?,?, 'none',36,?,?,10,'no',11,'generous',?,?,?, 'good'
2,4.5,4,?, 'none',37,'empl_contr',?,?,?,11,'average',?, 'full', 'yes',?, 'good'
1,2.8,?,?,?,35,?,?,2,?,12,'below_average',?,?,?, 'good'
2,3.5,4,?, 'none',40,?,?,2,'no',10,'below_average','no','half',?, 'half', 'bad'
3,3.5,4,5.1,'tcf',37,?,?,4,?,13,'generous',?, 'full', 'yes', 'full', 'good'
1,3,?,?, 'none',36,?,?,10,'no',11,'generous',?,?,?, 'good'
2,4.5,4,?, 'none',37,'empl_contr',?,?,?,11,'average',?, 'full', 'yes',?, 'good'
1,2.8,?,?,?,35,?,?,2,?,12,'below_average',?,?,?, 'good'
2,3.5,4,?, 'none',40,?,?,2,'no',10,'below_average','no','half',?, 'half', 'bad'
3,3.5,4,5.1,'tcf',37,?,?,4,?,13,'generous',?, 'full', 'yes', 'full', 'good'
1,3,?,?, 'none',36,?,?,10,'no',11,'generous',?,?,?, 'good'
2,4.5,4,?, 'none',37,'empl_contr',?,?,?,11,'average',?, 'full', 'yes',?, 'good'

Preprocessing in Weka

Pre-processing tools in WEKA are called “filters”. WEKA contains filters for discretization, normalization, resampling, ATTRIBUTE selection, transformation and combination of ATTRIBUTES . Some techniques, such as association rule mining, can only be performed on categorical data. This requires performing discretization on numeric or continuous ATTRIBUTES.

The weka.filters package is concerned with classes that transform datasets – by removing or adding ATTRIBUTES, resampling the dataset, removing examples and so on. This package offers useful support for data preprocessing, which is an important step in machine learning. All filters offer the options -i for specifying the input dataset, and -o for specifying the output dataset. If any of these parameters is not given, this specifies standard input resp. output for use within pipes. Other parameters are specific to each filter and can be found out via -h, as with any other class. The weka.filters package is organized into supervised and unsupervised filtering, both of which are again subdivided into instance and ATTRIBUTE filtering.

Steps for preprocessing on dataset labor.arff

1. Loading the data: We can load the dataset into weka by clicking on open file button in preprocessing interface and selecting the appropriate file.
2. Once the data is loaded, weka will recognize the attributes and during the scan of the data weka will compute some basic strategies on each attribute. The left panel in the above figure shows the list of recognized attributes while the top panel indicates the names of the base relation or table and the current working relation (which are same initially).
3. Clicking on an attribute in the left panel will show the basic statistics on the attributes for the categorical attributes the frequency of each attribute value is shown, while for continuous attributes we can obtain min, max, mean, standard deviation and deviation etc.,
4. The visualization in the right button panel in the form of cross-tabulation across two attributes.
(Note:we can select another attribute using the dropdown list)
5. Selecting or filtering attributes
6. **Filter example: Removing an attribute:**

When we need to remove an attribute,we can do this by using the attribute filters in weka. Following are steps to follow:

- a. In the filter model panel,click on choose button,This will show a popup window with a list of available filters.
 - b. Scroll down the list and select the “weka.filters.unsupervised.attribute.remove” filters.
 - c. Next click the textbox immediately to the right of the choose button.In the resulting dialog box enter the index of the attribute to be filtered out.
 - d. Make sure that invert selection option is set to false.The click OK now in the filter box.you will see “Remove-R-7”.
 - e. Click the apply button to apply filter to this data.This will remove the attribute and create new working relation.
 - f. Save the new working relation as an arff file by clicking save button on the top(button)panel.(labor.arff)
7. **Filter example: Discretization**

Sometimes association rule mining can only be performed on categorical data. This requires performing discretization on numeric or continuous attributes. In the following example let us discretize duration attribute. Steps to follow:

- Let us divide the values of duration attribute into three bins(intervals).
- From loaded labor.arff dataset in weka; Select the duration attribute.
- Activate filter-dialog box and select "WEKA.filters.unsupervised.attribute.discretize" from the list.
- To change the defaults for the filters,click on the box immediately to the right of the choose button.
- We enter the index for the attribute to be discretized.In this case the attribute is duration So we must enter '1' corresponding to the duration attribute.
- Enter '3' as the number of bins.Leave the remaining field values as they are.
- Click OK button.
- Click apply in the filter panel.This will result in a new working relation with the selected attribute partition into 3 bin.
- Save the new working relation in a file called labor-data-discretized.arff

Before Discretization:

The screenshot shows the Weka Explorer interface. The 'Filter' dialog box is open, and 'duration' is selected as the attribute to be discretized. The 'Selected attribute' panel displays the following statistics:

Statistic	Value
Minimum	1
Maximum	3
Mean	2.161
StdDev	0.708

A histogram below the statistics shows the distribution of values across three bins. The x-axis is labeled 1, 2, 3. The y-axis represents frequency. The first bin (1) has a frequency of 10, the second bin (2) has a frequency of 27, and the third bin (3) has a frequency of 19.

After Discretization:

The screenshot shows the Weka Explorer interface after discretization. The 'Filter' dialog box is open, and 'Discretize - B 3 - M - 1.0 - R first-last' is selected. The 'Selected attribute' panel displays the following statistics for the discretized 'duration' attribute:

No.	Label	Count	Weight
1	'[-inf-1.666667]	10	10.0
2	'[1.666667-2.333333]	27	27.0
3	'[2.333333-inf]	19	19.0

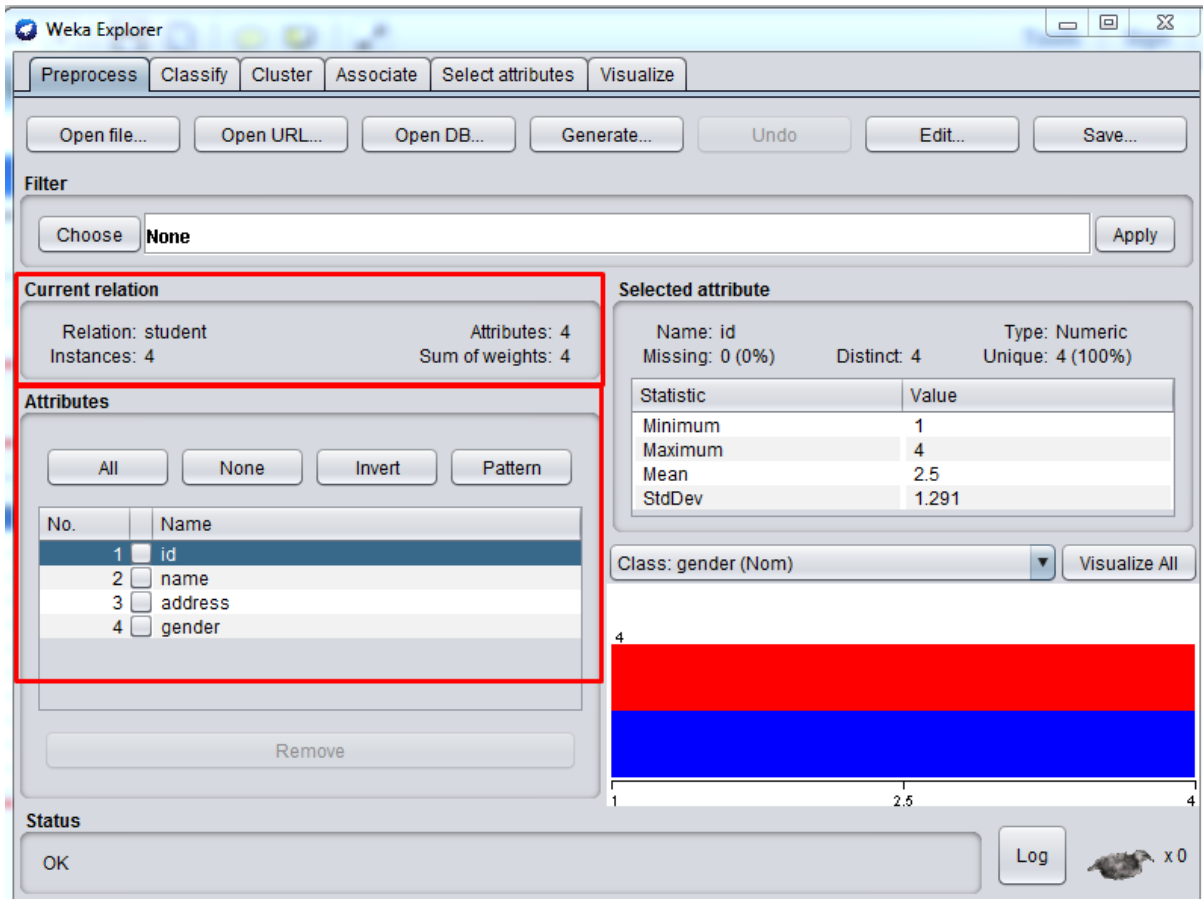
A histogram below the statistics shows the distribution of values across three bins. The x-axis is labeled 1, 2, 3. The y-axis represents frequency. The first bin (1) has a frequency of 10, the second bin (2) has a frequency of 27, and the third bin (3) has a frequency of 19.

AIM:

Demonstration of Data Visualization using Weka

DESCRIPTION:

Once the data is loaded, WEKA will recognize the attributes and during the scan of the data will compute some basic statistics on each attribute. The left panel in below figure shows the list of recognized attributes, while the top panels indicate the names of the base relation (or table) and the current working relation (which are the same initially).



Clicking on any attribute in the left panel will show the basic statistics on that attribute. For categorical attributes, the frequency for each attribute value is shown, while for continuous attributes we can obtain min, max, mean, standard deviation, etc. As an example, see Figures below which show the results of selecting the "id" and "gender" attributes, respectively.

After selection of id attribute:

The screenshot shows the Weka Explorer interface with the 'id' attribute selected. The 'Attributes' list on the left has 'id' checked. The 'Selected attribute' panel on the right displays the following statistics:

Statistic	Value
Minimum	1
Maximum	4
Mean	2.5
StdDev	1.291

The 'Current relation' panel shows: Relation: student, Instances: 4, Attributes: 4, Sum of weights: 4. The 'Status' bar shows 'OK'.

After selection of gender attribute:

The screenshot shows the Weka Explorer interface with the 'gender' attribute selected. The 'Attributes' list on the left has 'gender' checked. The 'Selected attribute' panel on the right displays the following statistics:

No.	Label	Count	Weight
1	M	2	2.0
2	F	2	2.0

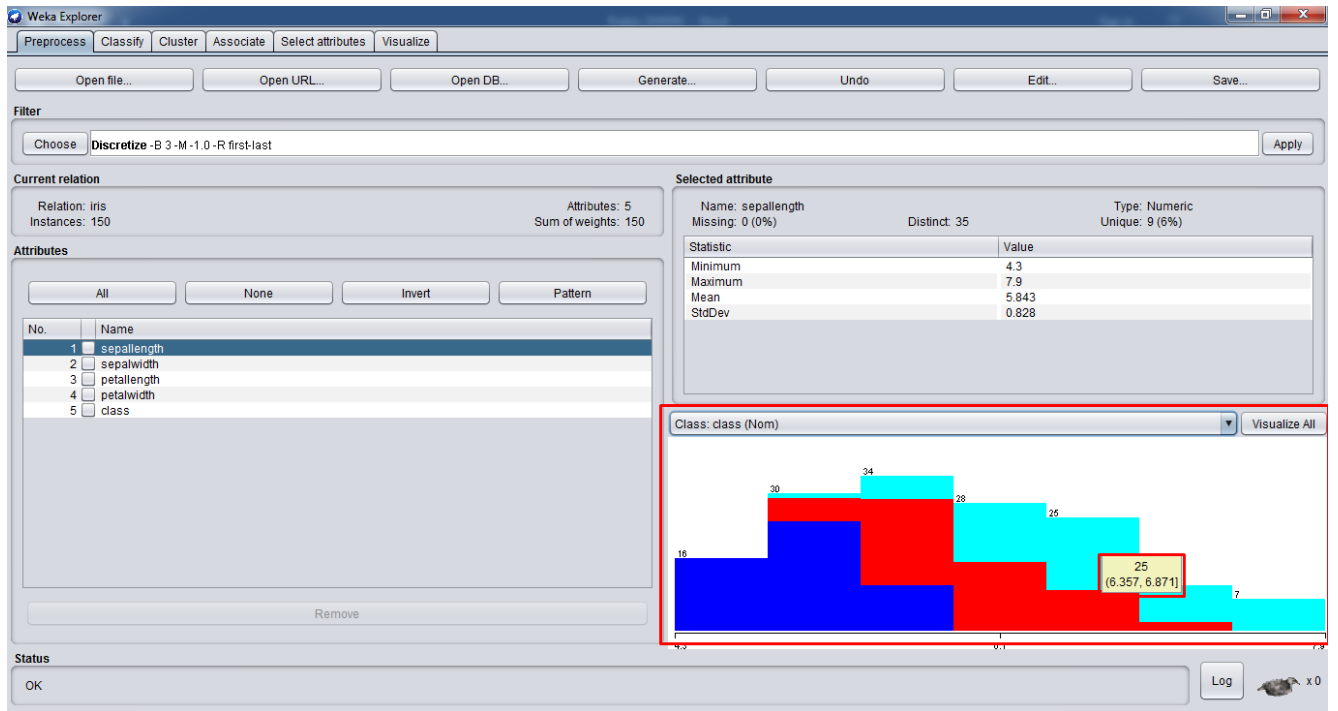
The 'Current relation' panel shows: Relation: student, Instances: 4, Attributes: 4, Sum of weights: 4. The 'Status' bar shows 'OK'.

Visualization with Iris dataset:

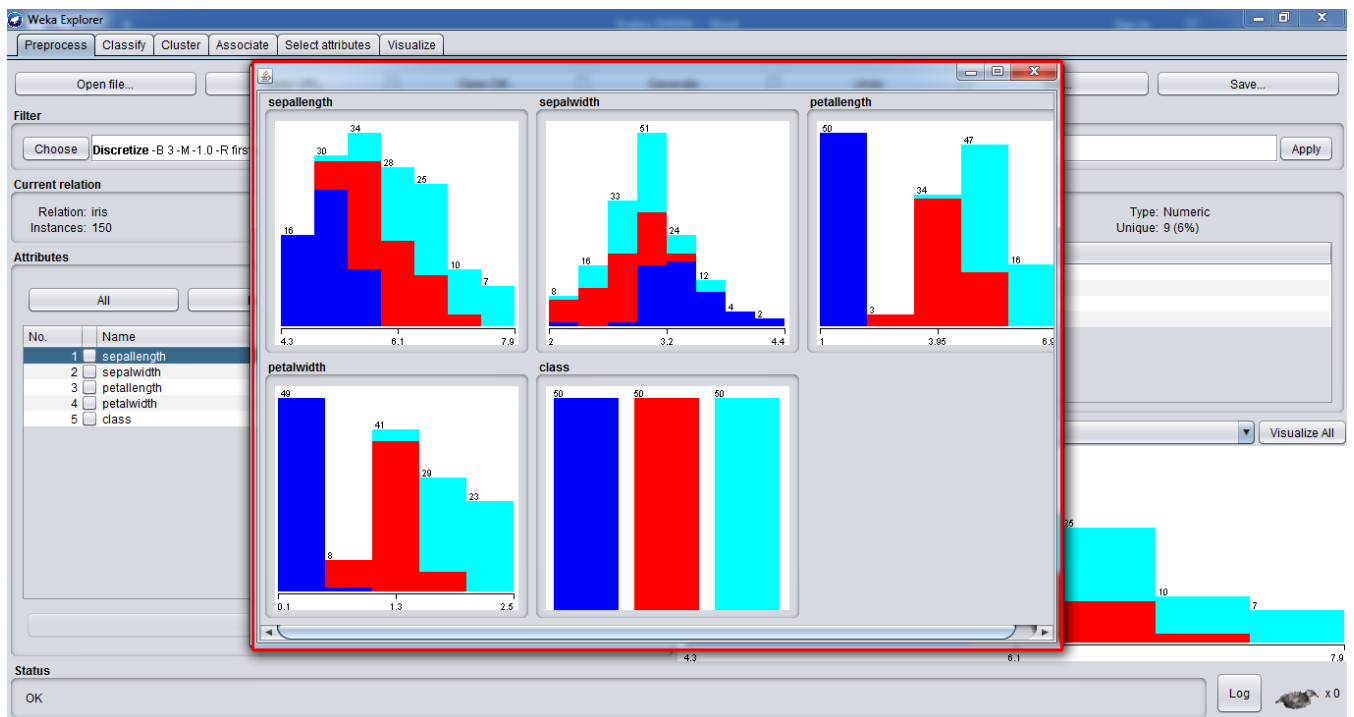
There are a number of ways in which you can use Weka to visualize your data.

After loading dataset the main GUI will show a histogram for the attribute distributions for a single selected attribute at a time, by default this is the class attribute.

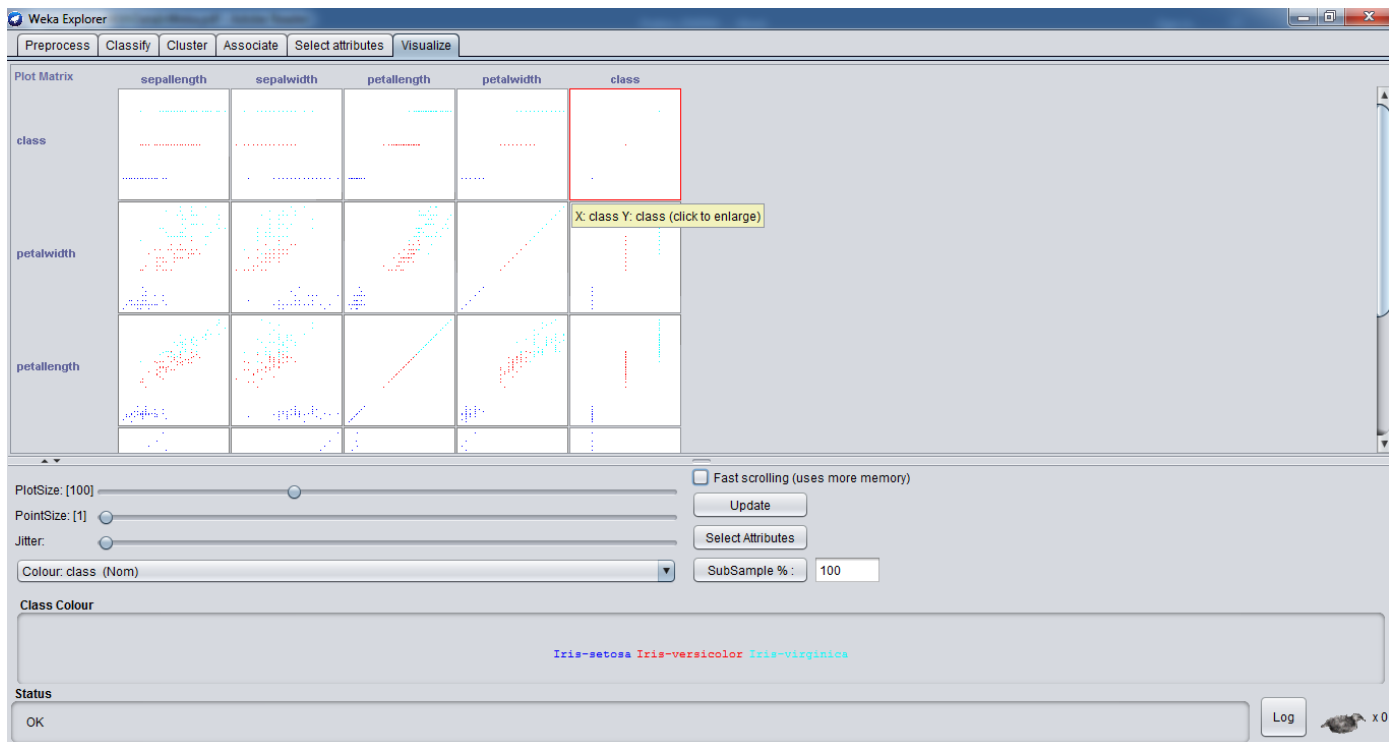
Note that the individual colors indicate the individual classes (the Iris dataset has 3). If you move the mouse over the histogram, it will show you the ranges and how many samples fall in each range.



The button VISUALIZE ALL will let you bring up a screen showing all distributions at once as in the picture below:



There is also a tab called called VISUALIZE. Clicking on that will open the scatterplots for all attribute pairs:



From these scatterplots, we can infer a number of interesting things. For example, in the picture above we can see that in some examples the clusters (for now, think of clusters as collections of points that are physically close to each other on the screen) and the different colors correspond to each other such as for example in the plots for class/(any attribute) pairs and the petalwidth/petallength attribute pair, whereas for other pairs (sepalwidth/sepalength for example) it's much harder to separate the clusters by color.

By default, the colors indicate the different classes, in this case we used red and two shades of blue. Left clicking on any of the highlighted class names towards the bottom of the screenshot allows you to set your own color for the classes. Also, by default, the color is used in conjunction with the class attribute, but it can be useful to color the other attributes as well. For example, changing the color to the fourth attribute by clicking on the arrow next to the bar that currently reads Color: class (Num) and selecting petalwidth enables us to observe even more about the data, for example the fact that for the class/sepalength attribute pair, which range of attribute values (indicated by different color) tends to go along with which class.

AIM: Demonstration of Association Rules extraction on Market basket data using apriori/ FP Algorithm

DESCRIPTION:

Association rule generation is usually split up into two separate steps:

1. First, minimum support is applied to find all frequent itemsets in a database.
2. Second, these frequent itemsets and the minimum confidence constraint are used to form rules.

While the second step is straight forward, the first step needs more attention.

Finding all frequent itemsets in a database is difficult since it involves searching all possible itemsets (item combinations). The set of possible itemsets is the power set over I and has size $2^n - 1$ (excluding the empty set which is not a valid itemset). Although the size of the powerset grows exponentially in the number of items n in I , efficient search is possible using the downward-closure property of support (also called anti-monotonicity) which guarantees that for a frequent itemset, all its subsets are also frequent and thus for an infrequent itemset, all its supersets must also be infrequent. Exploiting this property, efficient algorithms (e.g., Apriori and Eclat) can find all frequent itemsets.

Apriori Algorithm Pseudo code:

```
procedure Apriori (T, minSupport) { //T is the database and minSupport is the minimum support
L1 = { frequent items };
for (k = 2; Lk-1 != ∅; k++) {
Ck = candidates generated from Lk-1
//that is cartesian product Lk-1 x Lk-1 and eliminating any k-1 size itemset that is not
//frequent for each transaction t in database do{
#increment the count of all candidates in Ck that are contained in t
Lk = candidates in Ck with minSupport
} //end for each
} //end for return U ; }
```

As is common in association rule mining, given a set of itemsets (for instance, sets of retail transactions, each listing individual items purchased), the algorithm attempts to find subsets which are common to at least a minimum number C of the itemsets. Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as candidate generation), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found.

Apriori uses breadth-first search and a tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length $k - 1$. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent k -length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.

Apriori, while historically significant, suffers from a number of inefficiencies or trade-offs, which have spawned other algorithms. Candidate generation generates large numbers of subsets (the algorithm attempts to load up the candidate set with as many as possible before each scan). Bottom-up subset exploration (essentially a breadth-first traversal of the subset lattice) finds any maximal subset S only after all $2^{|S|} - 1$ of its proper subsets.

Steps to run data set using apriori algorithm:

1. Load arff dataset
2. From the displayed functions above select associate.
3. Go to choose option below the functions.
4. By clicking on the choose you may select the required algorithm (apriori), to demonstrate association rule process.

OUTPUT:

```

Weka Explorer
Preprocess Classify Cluster Associate Select attributes Visualize
Associate
Choose Apriori -N 10 -I 0 -C 0.5 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

Start Stop
Result list (right-click...)
18:24:54 - Apriori
18:25:34 - Apriori

Associate output
Scheme: weka.associations.Apriori -N 10 -I 0 -C 0.5 -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: 0.25 (6 instances)
Minimum metric <confidence>: 0.5
Number of cycles performed: 15

Generated sets of large itemsets:

Size of set of large itemsets L(1): 10
Size of set of large itemsets L(2): 18
Size of set of large itemsets L(3): 4

Best rules found:

1. tear-prod-rate=reduced 12 ==> contact-lenses=none 12  conf:(1)
2. spectacle-prescrip=myope tear-prod-rate=reduced 6 ==> contact-lenses=none 6  conf:(1)
3. spectacle-prescrip=hypermetrope tear-prod-rate=reduced 6 ==> contact-lenses=none 6  conf:(1)
4. astigmatism=no tear-prod-rate=reduced 6 ==> contact-lenses=none 6  conf:(1)
5. astigmatism=yes tear-prod-rate=reduced 6 ==> contact-lenses=none 6  conf:(1)
6. spectacle-prescrip=myope contact-lenses=none 7 ==> tear-prod-rate=reduced 6  conf:(0.86)
7. astigmatism=no contact-lenses=none 7 ==> tear-prod-rate=reduced 6  conf:(0.86)
8. contact-lenses=none 15 ==> tear-prod-rate=reduced 12  conf:(0.8)
9. age=presbyopic 0 ==> contact-lenses=none 6  conf:(0.75)
10. spectacle-prescrip=hypermetrope contact-lenses=none 9 ==> tear-prod-rate=reduced 6  conf:(0.75)

Status
OK
Log

```

Association rule process using FP-growth algorithm:

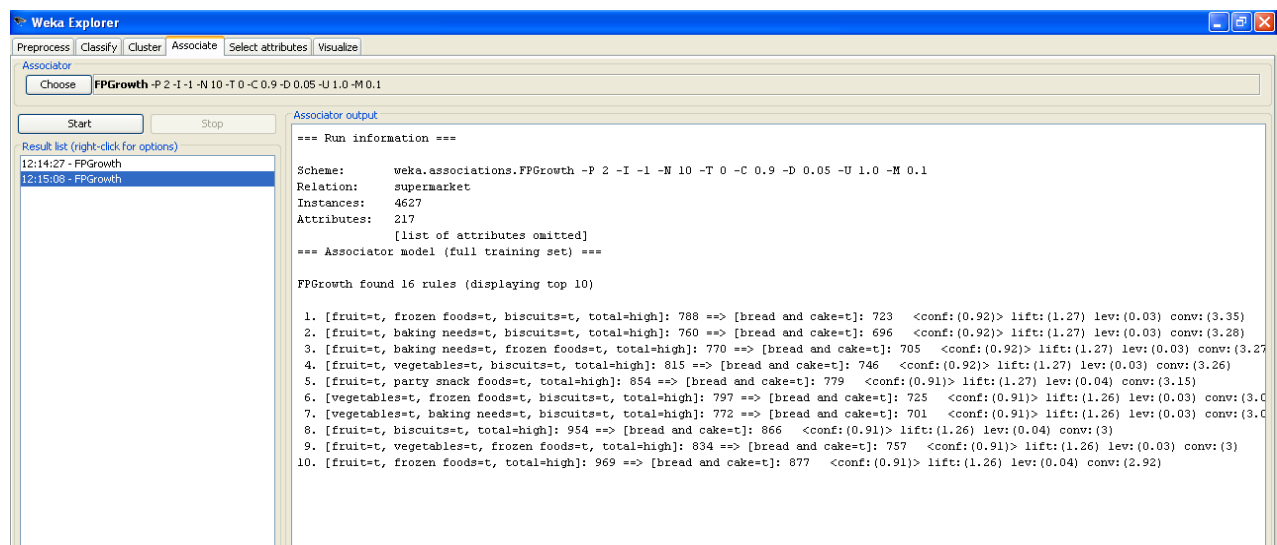
FP stands for frequent pattern. In the first pass, the algorithm counts occurrence of items (attribute-value pairs) in the dataset, and stores them to 'header table'. In the second pass, it builds the FP-tree structure by inserting instances. Items in each instance have to be sorted by descending order of their frequency in the dataset, so that the tree can be processed quickly. Items in each instance that do not meet minimum coverage threshold are discarded. If many instances share most frequent items, FP-tree provides high compression close to tree root.

Recursive processing of this compressed version of main dataset grows large item sets directly, instead of generating candidate items and testing them against the entire database. Growth starts from the bottom of the header table (having longest branches), by finding all instances matching given condition. New tree is created, with counts projected from the original tree corresponding to the set of instances that are conditional on the attribute, with each node getting sum of its children counts. Recursive growth ends when no individual items conditional on the attribute meet minimum support threshold, and processing continues on the remaining header items of the original FP-tree. Once the recursive process has completed, all large item sets with minimum coverage have been found, and association rule creation begins.

Steps to run data set using FPGrowth algorithm:

1. Load arff dataset
2. From the displayed functions above select associate.
3. Go to choose option below the functions.
4. By clicking on the choose you may select the required algorithm (FPGrowth), to demonstrate association rule process.

OUTPUT:



The screenshot shows the Weka Explorer interface with the Associate tab selected. The FPGrowth algorithm is chosen, and the output window displays the following information:

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

FPGrowth found 16 rules (displaying top 10)

1. [fruit=t, frozen foods=t, biscuits=t, total=high]: 788 ==> [bread and cake=t]: 723 <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.35)
2. [fruit=t, baking needs=t, biscuits=t, total=high]: 760 ==> [bread and cake=t]: 696 <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.28)
3. [fruit=t, baking needs=t, frozen foods=t, total=high]: 770 ==> [bread and cake=t]: 705 <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.27)
4. [fruit=t, vegetables=t, biscuits=t, total=high]: 815 ==> [bread and cake=t]: 746 <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.26)
5. [fruit=t, party snack foods=t, total=high]: 854 ==> [bread and cake=t]: 779 <conf:(0.91)> lift:(1.27) lev:(0.04) conv:(3.15)
6. [vegetables=t, frozen foods=t, biscuits=t, total=high]: 797 ==> [bread and cake=t]: 725 <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3.0)
7. [vegetables=t, baking needs=t, biscuits=t, total=high]: 772 ==> [bread and cake=t]: 701 <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3.0)
8. [fruit=t, biscuits=t, total=high]: 954 ==> [bread and cake=t]: 866 <conf:(0.91)> lift:(1.26) lev:(0.04) conv:(3)
9. [fruit=t, vegetables=t, frozen foods=t, total=high]: 834 ==> [bread and cake=t]: 757 <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3)
10. [fruit=t, frozen foods=t, total=high]: 969 ==> [bread and cake=t]: 877 <conf:(0.91)> lift:(1.26) lev:(0.04) conv:(2.92)
```

AIM: Demonstration of Classification Rule extraction a bench mark dataset using j48/ID3 Algorithm

DESCRIPTION:

Classification rule process using ID3 Algorithm:

In decision tree learning, ID3 (Iterative Dichotomiser) is an algorithm invented by Ross Quinlan used to generate a decision tree from the dataset. ID3 is typically used in the machine learning and natural language processing domains. The decision tree technique involves constructing a tree to model the classification process. Once a tree is built, it is applied to each tuple in the database and results in classification for that tuple. The following issues are faced by most decision tree algorithms :

- Choosing splitting attributes
- Ordering of splitting attributes
- Number of splits to take
- Balance of tree structure and pruning
- Stopping criteria

The ID3 algorithm is a classification algorithm based on Information Entropy, its basic idea is that all examples are mapped to different categories according to different values of the condition attribute set; its core is to determine the best classification attribute form condition attribute sets. The algorithm chooses information gain as attribute selection criteria; usually the attribute that has the highest information gain is selected as the splitting attribute of current node, in order to make information entropy that the divided subsets need smallest . According to the different values of the attribute, branches can be established, and the process above is recursively called on each branch to create other nodes and branches until all the samples in a branch belong to the same category. To select the splitting attributes, the concepts of Entropy and Information Gain are used.

Entropy

Given probabilities p_1, p_2, \dots, p_s , where $\sum p_i = 1$, Entropy is defined as

$$H(p_1, p_2, \dots, p_s) = \sum - (p_i \log p_i)$$

Entropy finds the amount of order in a given database state. A value of $H = 0$ identifies a perfectly classified set. In other words, the higher the entropy, the higher the potential to improve the classification process.

Information Gain

ID3 chooses the splitting attribute with the highest gain in information, where gain is defined as difference between how much information is needed after the split. This is calculated by determining the differences between the entropies of the original dataset and the weighted sum of the entropies from each of the subdivided datasets. The formula used for this purpose is:

$$G(D, S) = H(D) - \sum P(D_i)H(D_i)$$

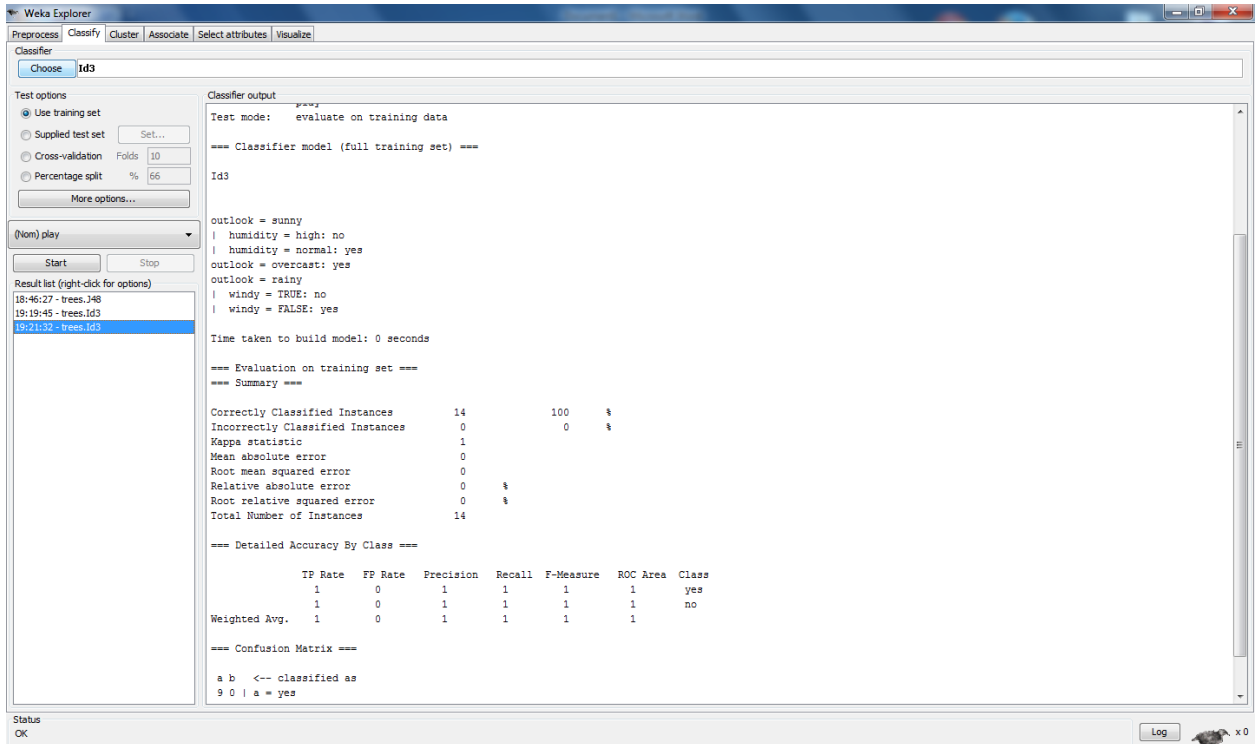
Steps to run data set using ID3 Algorithm:

1. Load arff dataset.
2. From the displayed functions above select classify.
3. Go to choose option below the functions.
4. By clicking on the choose you may observe certain ways for classification.
5. Choose tress from different classifications shown.

6. Now from tress ,select ID3 and run the data set.

Data Mining Lab from prabinsilwal.com.np

OUTPUT:



Classification rule process using J48 :

Classification is the process of building a model of classes from a set of records that contain class labels. Decision Tree Algorithm is to find out the way the attributes-vector behaves for a number of instances. Also on the bases of the training instances the classes for the newly generated instances are being found. This algorithm generates the rules for the prediction of the target variable. With the help of tree classification algorithm the critical distribution of the data is easily understandable.

J48 is an extension of ID3. The additional features of J48 are accounting for missing values, decision trees pruning, continuous attribute value ranges, derivation of rules, etc. In the WEKA data mining tool, J48 is an open source Java implementation of the C4.5 algorithm. The WEKA tool provides a number of options associated with tree pruning. In case of potential over fitting pruning can be used as a tool for précising. In other algorithms the classification is performed recursively till every single leaf is pure, that is the classification of the data should be as perfect as possible. This algorithm it generates the rules from which particular identity of that data is generated. The objective is progressively generalization of a decision tree until it gains equilibrium of flexibility and accuracy.

Basic Steps in the Algorithm:

1. In case the instances belong to the same class the tree represents a leaf so the leaf is returned by labeling with the same class.
2. The potential information is calculated for every attribute, given by a test on the attribute. Then the gain in information is calculated that would result from a test on the attribute.
3. Then the best attribute is found on the basis of the present selection criterion and that attribute selected for branching.

Counting Gain

This process uses the “Entropy” which is a measure of the data disorder. The Entropy of is calculated by And Gain is

$$Entropy(\bar{y}) = - \sum_{j=1}^n \frac{|y_j|}{|\bar{y}|} \log \left(\frac{|y_j|}{|\bar{y}|} \right)$$

$$Entropy(j|\bar{y}) = \frac{|y_j|}{|\bar{y}|} \log \left(\frac{|y_j|}{|\bar{y}|} \right)$$

And Gain is

$$Gain(\bar{y}, j) = Entropy(\bar{y}) - Entropy(j|\bar{y})$$

The objective is to maximize the Gain, dividing by overall entropy due to split argument \bar{y} by value j.

Steps to run data set using J48 Decision Tree:

1. Load arff dataset
2. From the displayed functions above select classify.
3. Go to choose option below the functions.
4. By clicking on the choose you may observe certain ways for classification.
5. Choose tress from different classifications shown.
6. Now from tress, select J48 and run the data set.

OUTPUT:

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

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

Test options

- Use training set
- Supplied test set
- Cross-validation Folds
- Percentage split %
-

(Nom) class

Result list (right-click for options)

18:46:27 - trees.J48

Classifier output

```

==== Classifier model (full training set) ====
J48 pruned tree
-----
petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth > 0.6
|   petalwidth <= 1.7
|   |   petallength <= 4.9: Iris-versicolor (48.0/1.0)
|   |   petallength > 4.9
|   |   |   petalwidth <= 1.5: Iris-virginica (3.0)
|   |   |   petalwidth > 1.5: Iris-versicolor (3.0/1.0)
|   |   petalwidth > 1.7: Iris-virginica (46.0/1.0)
-----
Number of Leaves :    5
Size of the tree :    9

Time taken to build model: 0 seconds

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

==== Detailed Accuracy By Class ====
      TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area  Class
-----
      1         0         1         1         1         1         Iris-setosa
      0.98      0.02      0.961      0.98      0.97      0.99      Iris-versicolor
      0.96      0.01      0.98      0.96      0.97      0.99      Iris-virginica
  
```

Status OK

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

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

Test options

- Use training set
- Supplied test set
- Cross-validation Folds
- Percentage split %
-

(Nom) class

Result list (right-click for options)

18:46:27 - trees.J48

Classifier output

```

-----
|   |   petalwidth <= 1.5: Iris-virginica (3.0)
|   |   petalwidth > 1.5: Iris-versicolor (3.0/1.0)
|   petalwidth > 1.7: Iris-virginica (46.0/1.0)
-----
Number of Leaves :    5
Size of the tree :    9

Time taken to build model: 0 seconds

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

==== Detailed Accuracy By Class ====
      TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area  Class
-----
      1         0         1         1         1         1         Iris-setosa
      0.98      0.02      0.961      0.98      0.97      0.99      Iris-versicolor
      0.96      0.01      0.98      0.96      0.97      0.99      Iris-virginica
Weighted Avg.   0.98      0.01      0.98      0.98      0.98      0.993

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

Weka Classifier Tree Visualizer: 18:46:27 - trees.J48 (iris)

Status OK

EXPERIMENT: 5

AIM: Demonstration of Classification Rule Process on any datasets using Navie Bayes Algorithm

DESCRIPTION:

The Naive Bayesian classifier is based on Bayes' theorem with independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods.

Algorithm:

Bayes theorem provides a way of calculating the posterior probability, $P(c|x)$, from $P(c)$, $P(x)$, and $P(x|c)$. Naive Bayes classifier assume that the effect of the value of a predictor (x) on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$$

- $P(c|x)$ is the posterior probability of *class (target)* given *predictor (attribute)*.
- $P(c)$ is the prior probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of *predictor* given *class*.
- $P(x)$ is the prior probability of *predictor*.

Steps to run data set using Navie Bayes Algorithm:

1. Load arff dataset
2. From the displayed functions above select classify.
3. Go to choose option below the functions.
4. By clicking on the choose you may observe certain ways for classification.
5. Choose bayes from different classifications shown.
6. Now from bayes ,select Navie Bayes and run the data set.

OUTPUT:

The screenshot shows the Weka Explorer interface with the Naive Bayes classifier selected. The 'Classifier output' pane displays the following information:

```
=== Run information ===
Scheme: weka.classifiers.bayes.NaiveBayes
Relation: weather
Instances: 14
Attributes: 5
  outlook
  temperature
  humidity
  windy
  play
Test mode: evaluate on training data

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

Attribute      Class      yes      no
              (0.63)   (0.38)
=====
outlook
sunny          3.0       4.0
overcast      5.0       1.0
rainy         4.0       3.0
[total]      12.0      8.0

temperature
mean          72.9697  74.8364
std. dev.    5.2304   7.384
weight sum    9        5
precision    1.9091   1.9091

humidity
mean          78.8395  86.1111
std. dev.    9.8023   9.2424
weight sum    9        5
precision    3.4444   3.4444

windy
```

The screenshot shows the Weka Explorer interface with the Naive Bayes classifier selected. The 'Classifier output' pane displays the following information:

```
mean          78.8395  86.1111
std. dev.    9.8023   9.2424
weight sum    9        5
precision    3.4444   3.4444

windy
TRUE         4.0       4.0
FALSE        7.0       3.0
[total]     11.0      7.0

Time taken to build model: 0 seconds

=== Evaluation on training set ===
=== Summary ===
Correctly Classified Instances      13          92.8571 %
Incorrectly Classified Instances     1          7.1429 %
Kappa statistic                     0.8372
Mean absolute error                  0.2798
Root mean squared error              0.3315
Relative absolute error               60.2576 %
Root relative squared error          69.1352 %
Total Number of Instances           14

=== Detailed Accuracy By Class ===
          TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area  Class
1         0.8      0.2      0.9       1      0.947      0.911   yes
0         0.8      0      1         0.8    0.889      0.911   no
Weighted Avg.  0.929    0.129    0.936    0.929  0.926      0.911

=== Confusion Matrix ===
a b <-- classified as
9 0 | a = yes
1 4 | b = no
```

EXPERIMENT: 6

Data Mining Lab from prabinsilwal.com.np

AIM: Demonstration of Classification Rule Process on any datasets using

K-nearest Neighbor classification Algorithm

DESCRIPTION:

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.

Algorithm:

A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function.

If $K = 1$, then the case is simply assigned to the class of its nearest neighbor.

Distance functions

Euclidean $\sqrt{\sum_{i=1}^k (x_i - y_i)^2}$

Manhattan $\sum_{i=1}^k |x_i - y_i|$

Minkowski $\left(\sum_{i=1}^k (|x_i - y_i|)^q \right)^{1/q}$

It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variable in the dataset.

Steps to run data set using K-means algorithm:

1. Load arff dataset
2. From the displayed functions above select classify.
3. Go to choose option below the functions.
4. By clicking on the choose you may observe certain ways for classification.
5. Choose tress from different classifications shown.
6. Now from tress ,select NBTtree and run the data set.

Output:

The screenshot shows the Weka Explorer interface with the 'Classifier' tab active. The 'NBTree' classifier is selected. The 'Test options' section shows 'Use training set' selected. The 'Result list' on the left shows several 'trees.NBTree' entries, with '14:56:03 - trees.NBTree' selected. The main window displays the following output:

```

Classifier output

Number of Leaves :    4
Size of the tree :    7

Time taken to build model: 0.3 seconds

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

Correctly Classified Instances      176           82.243 %
Incorrectly Classified Instances    38            17.757 %
Kappa statistic                    0.7572
Mean absolute error                 0.0893
Root mean squared error             0.212
Relative absolute error             42.1837 %
Root relative squared error         65.3406 %
Total Number of Instances          214

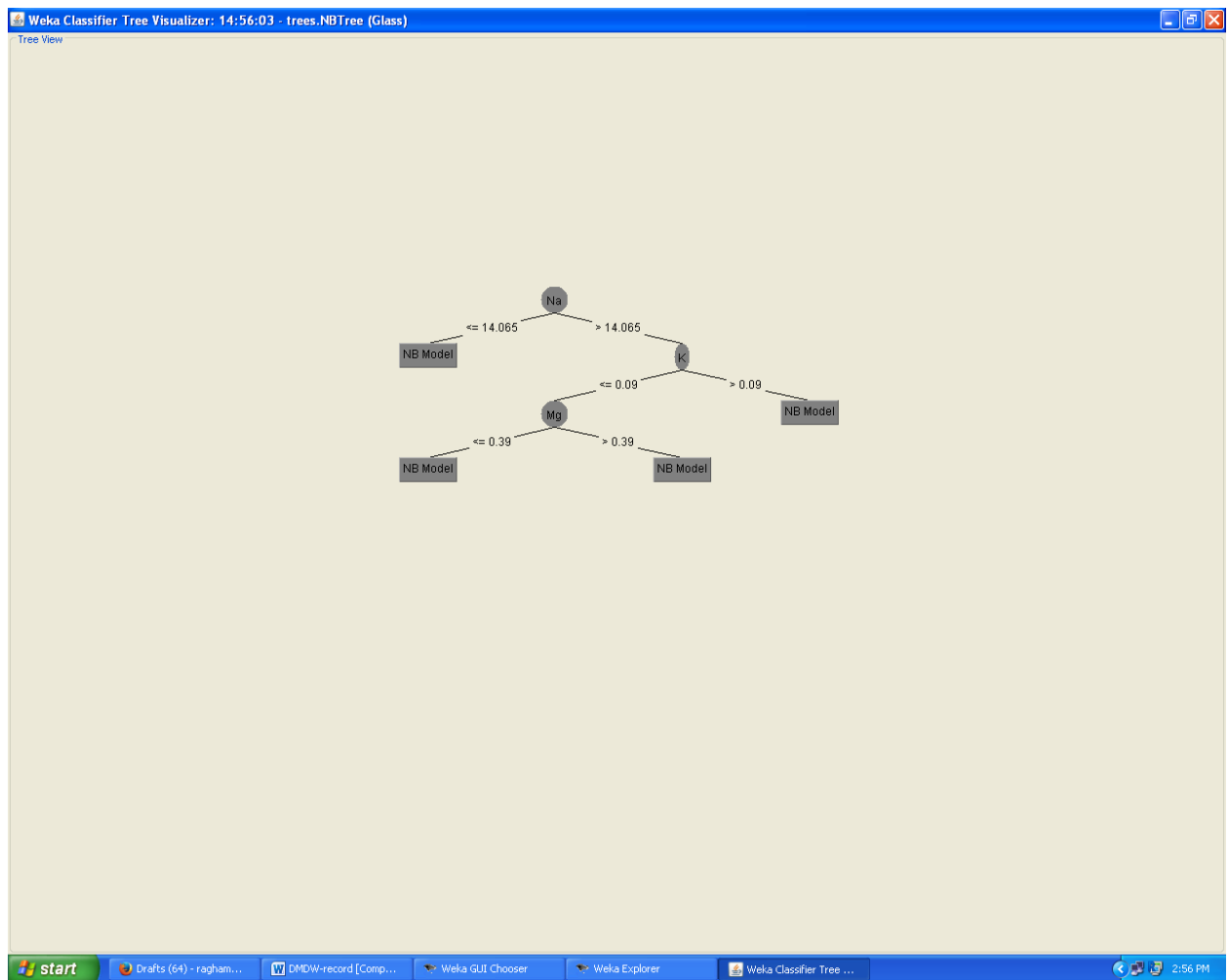
=== Detailed Accuracy By Class ===

              TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area  Class
              -----  -----  -
              0.914    0.139    0.762     0.914    0.831     0.919    build wind float
              0.75     0.08     0.838     0.75     0.792     0.89     build wind non-float
              0.529    0.01     0.818     0.529    0.643     0.934    vehic wind float
              0         0         0         0         0         ?        vehic wind non-float
              1         0.02     0.765     1         0.867     0.991    containers
              0.778    0         1         0.778    0.875     0.998    tableware
              0.897    0.005    0.963     0.897    0.929     0.982    headlamps
Weighted Avg.   0.822    0.076    0.831     0.822    0.819     0.926

=== Confusion Matrix ===

 a b c d e f g <-- classified as
64 6 0 0 0 0 0 | a = build wind float
13 57 2 0 3 0 1 | b = build wind non-float
6 2 9 0 0 0 0 | c = vehic wind float
0 0 0 0 0 0 0 | d = vehic wind non-float
0 0 0 0 13 0 0 | e = containers
0 2 0 0 0 7 0 | f = tableware
1 1 0 0 1 0 26 | g = headlamps

```



EXPERIMENT: 7

Data Mining Lab from prabinsilwal.com.np

AIM: Demonstration of partitional Clustering on any datasets using K-means Algorithm

DESCRIPTION:

k-means is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed and an early group age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more. Finally, this algorithm aims at minimizing an objective function known as squared error function given by:

$$J(V) = \sum_{i=1}^c \sum_{j=1}^{c_i} (\|x_i - v_j\|)^2$$

where,

' $\|x_i - v_j\|$ ' is the Euclidean distance between x_i and v_j .

' c_i ' is the number of data points in i^{th} cluster.

' c ' is the number of cluster centers.

Algorithmic steps for k-means clustering :

Let $X = \{x_1, x_2, x_3, \dots, x_n\}$ be the set of data points and $V = \{v_1, v_2, \dots, v_c\}$ be the set of centers.

1. Randomly select ' c ' cluster centers.
2. Calculate the distance between each data point and cluster centers.
3. Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers..
4. Recalculate the new cluster center using:

$$v_i = (1 / c_i) \sum_{j=1}^{c_i} x_j$$

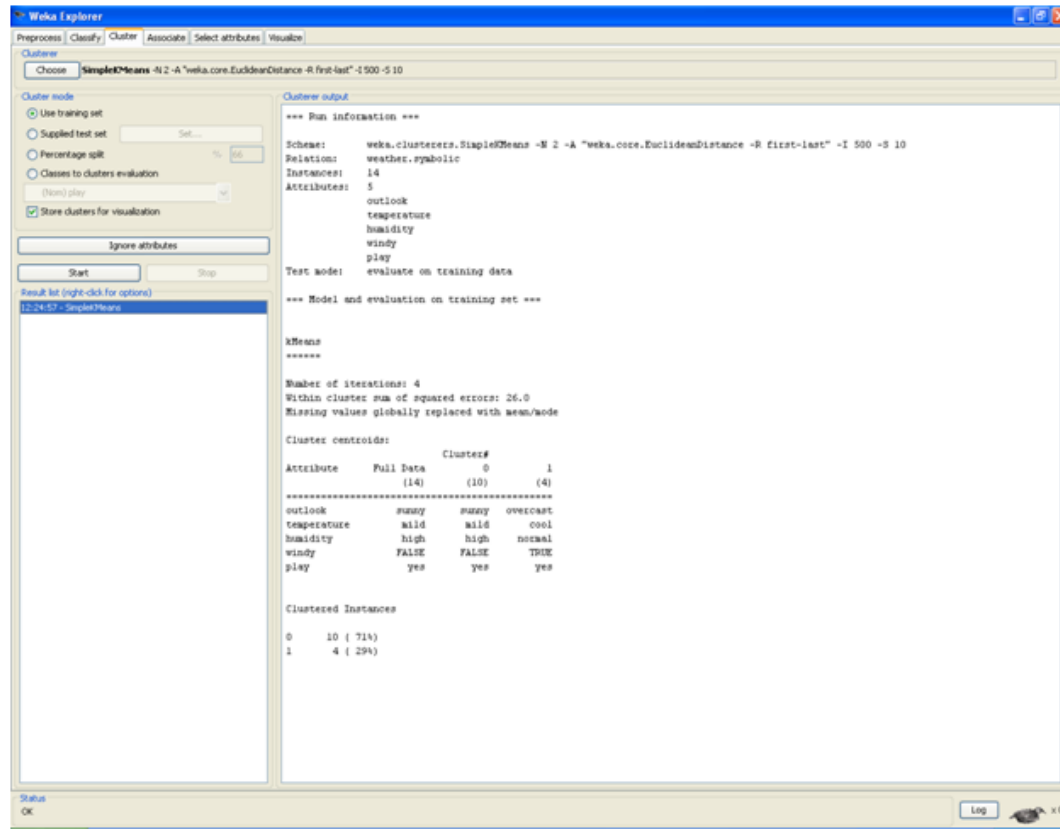
where, ' c_i ' represents the number of data points in i^{th} cluster.

5. Recalculate the distance between each data point and new obtained cluster centers.
6. If no data point was reassigned then stop, otherwise repeat from step 3).

Steps to run data set using simple K-means algorithm:

1. Load Dataset
2. From the displayed functions above select cluster.
3. Go to choose option below the functions.
4. By clicking on the choose you may observe certain forms of clusters.
5. Choose SimpleKMeans.
6. Now run the data set.

OUTPUT:



The screenshot shows the Weka Explorer interface with the SimpleKMeans algorithm selected. The 'Clusterer output' pane displays the following information:

```
*** Run information ***
Scheme: weka.clusterers.SimpleKMeans -M 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10
Relation: weather.symbolic
Instances: 14
Attributes: 5
  outlook
  temperature
  humidity
  windy
  play
Test model: evaluate on training data
*** Model and evaluation on training set ***

KMeans
*****
Number of iterations: 4
Within cluster sum of squared errors: 26.0
Missing values globally replaced with mean/mode

Cluster centroids:
Attribute      Full Data      Cluster#
              (14)          (10)          (4)
-----
outlook      sunny      sunny      overcast
temperature  mild      mild      cool
humidity     high      high      normal
windy        FALSE     FALSE     TRUE
play         yes       yes       yes

Clustered Instances
0      10 ( 71%)
1       4 ( 29%)
```

EXPERIMENT: 8

Data Mining Lab from prabinsilwal.com.np

AIM: Demonstration of Clustering on any datasets using simple K-medoids algorithm

DESCRIPTION:

The **K-medoids** clustering algorithm is similar to the *K-means algorithm*. Both of these two algorithms partition the dataset into groups and minimize the **squared error** based on the distance between points. In contrast to the K-means algorithm which chooses the means as the centroids, the K-medoids chooses datapoints as centers (medoids or exemplars).

The most common realisation of K-medoid clustering algorithm is the Partitioning Around Medoids (PAM) algorithm.

The PAM algorithm works as following steps:

1. Initialization step: randomly select k data points in the dataset as the medoids.
2. Assignment step: associate each data point to the closest medoid. The closeness is measured by any valid distance metric, such as Euclidean distance, Manhattan distance, or Minkowski distance.
3. Updating step:
For each cluster
For each data point o in the cluster
Compute the cost of o as the medoid. The cost is defined as the DistanceError which is
$$DistanceError(o) = \sum_{p \in cluster} dist(o, p)$$
Choose the o with the smallest cost as the new medoid.
4. Repeat steps 2 and 3 until there is no change in medoids.
5. Repeat steps 1 to 4 for several times and output the solution with the least squared error.

K-medoid in weka:

Weka provides new tab called Subspace Clustering for additional clustering methods.

While subspace clustering is a rather young area that has been researched for only one decade, several distinct paradigms can be observed in the literature. Our system includes representatives of these paradigms to provide an overview over the techniques available. We provide implementations of the most recent approaches from different paradigms:

1. Cell-based subspace clustering discretizes the data space for efficient detection of dense grid cells in a bottom-up fashion. It was introduced in the CLIQUE approach which exploits monotonicity on the density of grid cells for pruning. SCHISM extends CLIQUE using a variable threshold adapted to the dimensionality of the subspace as well as efficient heuristics for pruning. In contrast, DOC and MINECLUS use variable cells represented by hypercubes.
2. Density-based subspace clustering defines clusters as dense areas separated by sparsely populated areas. In SUBCLU, a density monotonicity property is used to prune subspaces in a bottom-up fashion. FIRES extends this paradigm by using variable neighborhoods and an approximative heuristics for efficient computation. In INSCY we use dimensionality unbiased density, normalized with respect to the dimensionality of the subspace and in addition in-process pruning of redundant subspace clusters achieves meaningful result sizes.

- Clustering oriented methods optimize the overall clustering result. **PROCLUS** extends the k-medoid algorithm by iteratively refining a full-space **k-medoid** clustering in a top-down manner. P3C combines one-dimensional cluster cores to higher-dimensional clusters bottom-up. STATPC uses a statistical test to remove redundant clusters out of the result.

NEW: Subspace Clustering Tab

Selection of subspace cluster algorithm

Bracketing configuration

Selection of measurements

Bracketing result area

Open single visualization

Open visualization with overview

Evaluation area

SubspaceClusterer output

```

SC_2: [0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 ] #1852 (2 4 6 21 25 27 38 4:
SC_3: [0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 ] #825 (1 5 14 28 40 46 47 6:

Clustering took: 00h 00m 03s (3641ms)

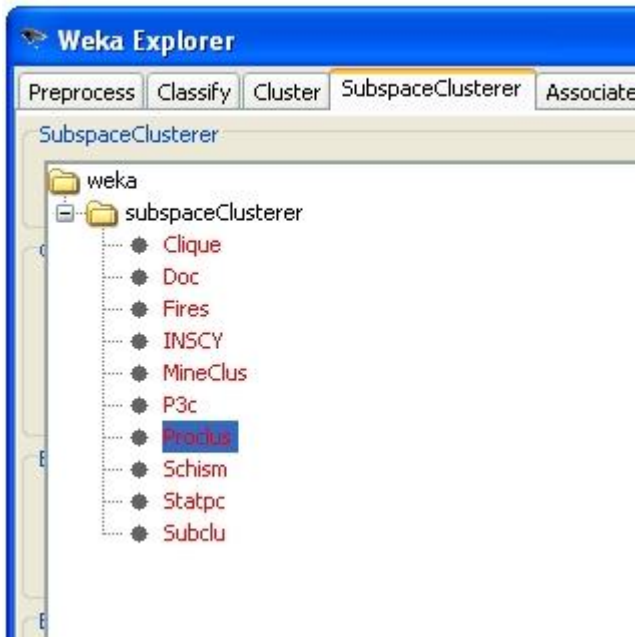
Unclustered instances : 1517

Evaluation measurements:
all          SC_0          SC_1          SC_2
Accuracy     0,32
1.0-ClusErr  0,16
Coverage     0,8          0,25         0,19         0,2
Entropy      0,68         0,7          0,73         0,6
FMeasure     0,18
1.0-RNIA    0,27

Cluster Distribution:
ClusterDist  0 1 2 1 0 0 0 0 0 0 0 0 0 0 0 0
NumOfCluster

```

You can perform k-medoids by selecting Proclus as shown below:



OUTPUT:

Data Mining Lab from prabinsilwal.com.np

The screenshot shows the Weka Explorer interface with the Subspace Clustering task. A 'Calculate' dialog box is open, showing various evaluation measures checked. The main window displays the 'SubspaceClusterer output' and a table of evaluation measurements.

SubspaceClusterer output:

```
SC_6: [0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 ] #955 (10 12 15 16 17 19 35)
SC_7: [0 0 0 1 0 1 0 0 0 0 0 0 1 1 1 1 1 ] #212 (27 46 49 89 128 143)
```

Clustering took: 00h 00m 04s (4219ms)

Unclustered instances : 2114

Evaluation measurements:	all	SC_0	SC_1	SC_2
Accuracy	0,42			
1.0-ClusErr	0,15			
Coverage	0,72	0,01	0,06	0,01
Entropy	0,49	0,32	0,64	0,51
F1Measure	0,37			
1.0-RNIA	0,28			

Cluster Distribution:
ClusterDist 0 0 0 0 2 4 2 0 0 0 0 0 0 0 0 0

Visualization of output:

The screenshot shows the 'Cluster Visualization Overview' window. It displays six small 3D cluster visualizations at the top. Below, two larger 3D-Browsing windows are shown, each displaying a cluster of yellow spheres. The left window shows a large, dense cluster, while the right window shows a smaller, more dispersed cluster. Both windows have a toolbar with buttons for '4', '5', '6', and '7'.

EXPERIMENT: 9

Data Mining Lab from prabinsilwal.com.np

AIM: Demonstration of Clustering rules process on any datasets of images using DB Scan algorithm

DESCRIPTION:

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm. It is a density-based clustering algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms.

Algorithm:

DBSCAN requires two parameters: ϵ (eps) and the minimum number of points required to form a dense region.

It starts with an arbitrary starting point that has not been visited. This point's ϵ -neighborhood is retrieved, and if it contains sufficiently many points, a cluster is started. Otherwise, the point is labeled as noise. Note that this point might later be found in a sufficiently sized ϵ -environment of a different point and hence be made part of a cluster.

In the following, we present a basic version of DBSCAN omitting details of data types and generation of additional information about clusters:

```
DBSCAN (SetOfPoints, Eps, MinPts)
// SetOfPoints is UNCLASSIFIED
ClusterId := nextId(NOISE);
FOR i FROM 1 TO SetOfPoints.size DO
Point := SetOfPoints.get(i);
IF Point.CId = UNCLASSIFIED THEN
IF ExpandCluster(SetOfPoints, Point,
ClusterId, Eps, MinPts) THEN
ClusterId := nextId(ClusterId)
END IF
END IF
END FOR
END; // DBSCAN
```

Steps to run data set using DBScan algorithm:

1. Load Dataset
2. From the displayed functions above select cluster.
3. Go to choose option below the functions.
4. By clicking on the choose you may observe certain forms of clusters.
5. Choose MakeDensityBasedClusterer.
6. Now run the data set.

OUTPUT:

Data Mining Lab from prabinsilwal.com.np

The screenshot shows the Weka Explorer interface with the DBScan algorithm applied to the Iris dataset. The 'Clusterer' tab is active, and the 'Cluster mode' is set to 'Use training set'. The 'Clusterer output' pane displays the following information:

```
=== Run information ===
Scheme: weka.clusterers.DBScan -E 0.9 -M 6 -I weka.clusterers.forOPTICSAndDBScan.Databases.SequentialDatabase -D weka.clusterers.forOPTICSAndDBScan.DataObjects.EuclidianDataObject
Relation: iris
Instances: 150
Attributes: 5
  sepalwidth
  sepallength
  petalwidth
  petallength
  class
Test mode: evaluate on training data

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

DBScan clustering results
-----

Clustered DataObjects: 150
Number of attributes: 5
Epsilon: 0.9; minPoints: 6
Index: weka.clusterers.forOPTICSAndDBScan.Databases.SequentialDatabase
Distance-type: weka.clusterers.forOPTICSAndDBScan.DataObjects.EuclidianDataObject
Number of generated clusters: 3
Elapsed time: .05

( 0.) 5.1,3.5,1.4,0.2,Iris-setosa --> 0
( 1.) 4.9,3.1,4.0,2.2,Iris-setosa --> 0
( 2.) 4.7,3.2,1.3,0.2,Iris-setosa --> 0
( 3.) 4.6,3.1,1.5,0.2,Iris-setosa --> 0
( 4.) 5.3,6.1,4.0,2.2,Iris-setosa --> 0
( 5.) 5.4,3.9,1.7,0.4,Iris-setosa --> 0
( 6.) 4.6,3.4,1.4,0.3,Iris-setosa --> 0
( 7.) 5.3,4.1,5.0,2.2,Iris-setosa --> 0
( 8.) 4.4,2.9,1.4,0.2,Iris-setosa --> 0
( 9.) 4.9,3.1,1.5,0.1,Iris-setosa --> 0
(10.) 5.4,3.7,1.5,0.2,Iris-setosa --> 0
(11.) 4.8,3.4,1.6,0.2,Iris-setosa --> 0
(12.) 4.8,3.1,1.4,0.1,Iris-setosa --> 0
(13.) 4.3,3.1,1.0,1.0,Iris-setosa --> 0
```

The screenshot shows the Weka Explorer interface with the DBScan algorithm applied to the Iris-virginica dataset. The 'Clusterer' tab is active, and the 'Cluster mode' is set to 'Use training set'. The 'Clusterer output' pane displays the following information:

```
Cluster output
(118.) 7.7,2.6,6.9,2.3,Iris-virginica --> 1
(119.) 6.2,2.5,1.5,Iris-virginica --> 1
(120.) 6.9,3.2,5.7,2.3,Iris-virginica --> 1
(121.) 5.6,2.8,4.9,2.2,Iris-virginica --> 1
(122.) 7.7,2.8,6.7,2.2,Iris-virginica --> 1
(123.) 6.3,2.7,4.9,1.8,Iris-virginica --> 1
(124.) 6.7,3.3,5.7,2.1,Iris-virginica --> 1
(125.) 7.2,3.2,6.1,1.8,Iris-virginica --> 1
(126.) 6.2,2.8,4.8,1.8,Iris-virginica --> 1
(127.) 6.1,3.4,9.1,1.8,Iris-virginica --> 1
(128.) 6.4,2.8,5.6,2.1,Iris-virginica --> 1
(129.) 7.2,3.5,8.1,1.6,Iris-virginica --> 1
(130.) 7.4,2.8,6.1,1.9,Iris-virginica --> 1
(131.) 7.9,3.8,6.4,2.2,Iris-virginica --> 1
(132.) 6.4,2.8,5.6,2.2,Iris-virginica --> 1
(133.) 6.3,2.8,5.1,1.5,Iris-virginica --> 1
(134.) 6.1,2.6,5.6,1.4,Iris-virginica --> 1
(135.) 7.7,3.6,6.1,2.3,Iris-virginica --> 1
(136.) 6.3,3.4,5.6,2.4,Iris-virginica --> 1
(137.) 6.4,3.1,5.5,1.8,Iris-virginica --> 1
(138.) 6.3,4.8,1.8,Iris-virginica --> 1
(139.) 6.9,3.1,5.4,2.1,Iris-virginica --> 1
(140.) 6.7,3.1,5.6,2.4,Iris-virginica --> 1
(141.) 6.9,3.1,5.1,2.3,Iris-virginica --> 1
(142.) 5.8,2.7,5.1,1.9,Iris-virginica --> 1
(143.) 6.8,3.2,5.9,2.3,Iris-virginica --> 1
(144.) 6.7,3.3,5.7,2.5,Iris-virginica --> 1
(145.) 6.7,3.5,2.2,3,Iris-virginica --> 1
(146.) 6.3,2.5,5.1,9,Iris-virginica --> 1
(147.) 6.5,3.5,2.2,Iris-virginica --> 1
(148.) 6.2,3.4,5.4,2.3,Iris-virginica --> 1
(149.) 5.9,3.5,1.1,8,Iris-virginica --> 1

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

EXPERIMENT: 10

Data Mining Lab from prabinsilwal.com.np

AIM: Demonstration of Clustering rules process on any datasets using Birch Algorithm

DESCRIPTION:

BIRCH (balanced iterative reducing and clustering using hierarchies) is an unsupervised data mining algorithm used to perform hierarchical clustering over particularly large data-sets. An advantage of BIRCH is its ability to incrementally and dynamically cluster incoming, multi-dimensional metric data points in an attempt to produce the best quality clustering for a given set of resources (memory and time constraints). In most cases, BIRCH only requires a single scan of the database.

Its inventors claim BIRCH to be the "first clustering algorithm proposed in the database area to handle 'noise' (data points that are not part of the underlying pattern) effectively", beating DBSCAN by two months. The algorithm received the SIGMOD 10 year test of time award in 2006.

BIRCH (Balanced Iterative Reducing and Clustering Using Hierarchies) is an integrated agglomerative hierarchical clustering method. It is mainly designed for clustering large amount of metric data. It is mainly suitable when there is limited amount of main memory and have to achieve a linear I/O time requiring only in one database scan. It introduces two concepts, clustering feature and clustering feature tree (CF tree), which are used to summarize cluster representations [Tian Zhang et al., 1996].

A CF tree is a height-balanced tree that stores the clustering features for a hierarchical clustering. It is similar to B+-Tree or R-Tree. CF tree is balanced tree with a branching factor (maximum number of children per none leaf node) B and threshold T. Each internal node contains a CF triple for each of its children. Each leaf node also represents a cluster and contains a CF entry for each sub cluster in it. A sub cluster in a leaf node must have a diameter no greater than a given threshold value (maximum diameter of sub-clusters in the leaf node) [Tian Zhang et al., 1996].

An object is inserted to the closest leaf entry (sub cluster). A leaf node represents a cluster made up of all sub clusters represented by its entries. All the entries in a leaf node must satisfy the threshold requirements with respect to the threshold value T, that is, the diameter of the sub cluster must be less than T. If the diameter of the sub cluster stored in the leaf node after insertion is larger than the threshold

Steps to run data set using BIRCH algorithm:

1. Load Dataset
2. From the displayed functions above select cluster.
3. Go to choose option below the functions.
4. By clicking on the choose you may observe certain forms of clusters.
5. Choose HierarchicalClustering.
6. Now run the data set.

