CMR TECHNICAL CAMPUS

UGC AUTONOMOUS

Kandlakoya(V), Medchal Road, Hyderabad – 501 401

Accredited by NBA and NAAC with A Grade

Approved by AICTE, New Delhi and Affiliated to JNTU, Hyderabad

DEPARTMENT OF CSE (AI & ML)

DATA MINING LAB MANUAL (R20)

COURSE CODE: 20CS504PC

Prepared by:

Mrs. Swaroopa Rani B

Assistant Professor

20CS504PC: DATA MINING LAB

III Year B.Tech. CSE(AI&ML) I-Sem

L	T	P	C
0	0	3	1.5

Course Objectives:

1. To obtain practical experience using data mining techniques on real-world data sets.
2. Emphasize hands-on experience working with classification and clusteringalgorithms.

Course Outcomes: At the end of the course, student will be able to

1. Apply classification mining algorithms as a component to the existing tools.
2. Apply clustering mining techniques for realistic data.

Softwares used:

1. WEKA Tool
2. Python

List of Experiments:

1. Demonstration of preprocessing techniques on dataset student.arff
2. Demonstration of preprocessing techniques on dataset labor.arff
3. Demonstration of Association rule mining on dataset contactlenses.arff using apriori algorithm
4. Demonstration of Association rule mining on dataset test.arff using apriori algorithm
5. Demonstration of classification process on dataset student.arff using j48 algorithm
6. Demonstration of classification process on dataset employee.arff using j48 algorithm
7. Demonstration of classification process on dataset employee.arff using id3 algorithm
8. Demonstration of classification process on dataset employee.arff using naïve bayes algorithm
9. Demonstration of clustering process on dataset iris.arff using simple k-means algorithm.
10. Demonstration of clustering process on dataset iris.arff using hierarchical clustering algorithm
11. Demonstration of clustering process on dataset iris.arff using density based clustering algorithm.
12. Implement the following data mining algorithms using python
    a) Decision Tree
    b) Naïve bayes
    c) K-Nearest Neighbor
    d) K-Means

References:

1. https://wekatutorial.com/
2. https://www.springboard.com/blog/data-science/data-mining-python-tutorial/

Exp.1 Demonstration of preprocessing techniques on dataset student.arff

Aim: This experiment illustrates some of the basic data preprocessing operations that can be performed using WEKA-Explorer. The sample dataset used for this example is the student data available in arff format.

Step1: Loading the data. We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.

Step2: 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).

Step3: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.,

Step4: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.

Step5:Selecting or filtering attributes

Removing an attribute-When we need to remove an attribute,we can do this by using the attribute filters in weka.In the filter model panel,click on choose button,This will show a popup window with a list of available filters.

Scroll down the list and select the “weka.filters.unsupervised.attribute.remove” filters.

Step 6:a)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.

b) Make sure that invert selection option is set to false.The click OK now in the filter box.you will see “Remove-R-7”.

c) Click the apply button to apply filter to this data.This will remove the attribute and create new working relation.

d) Save the new working relation as an arff file by clicking save button on the top(button)panel.(student.arff)

Discretization

1) 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 age attribute.

◎Let us divide the values of age attribute into three bins(intervals).

◎First load the dataset into weka(student.arff)

◎Select the age 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 age.So we must enter '1' corresponding to the age 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 bins.

◎Save the new working relation in a file called student-data-discretized.arff

Dataset student .arff

@relation student

@attribute age {<30,30-40,>40}

@attribute income {low, medium, high}

@attribute student {yes, no}

@attribute credit-rating {fair, excellent}

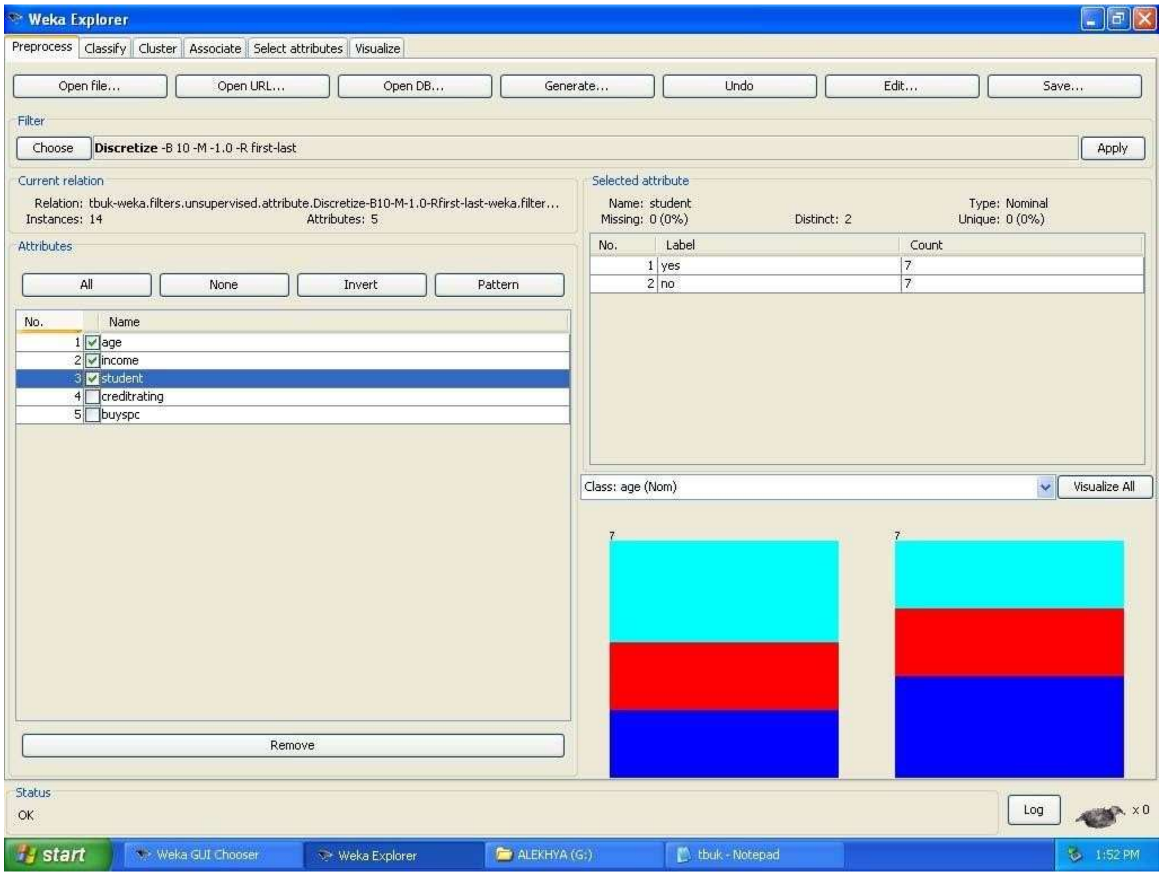
@attribute buyspc {yes, no}

@data

%

```
<30, high, no, fair, no

<30, high, no, excellent, no

30-40, high, no, fair, yes

>40, medium, no, fair, yes

>40, low, yes, fair, yes

>40, low, yes, excellent, no

30-40, low, yes, excellent, yes

<30, medium, no, fair, no

<30, low, yes, fair, no

>40, medium, yes, fair, yes

<30, medium, yes, excellent, yes

30-40, medium, no, excellent, yes

30-40, high, yes, fair, yes

>40, medium, no, excellent, no

%
```

The following screenshot shows the effect of discretization.



Discretize -B 10 -M -1.0 -R first-last

Current relation
Relation: tbuk-weka.filters.unsupervised.attribute.Discretize-B10-M-1.0-Rfirst-last-weka.filter...
Instances: 14 Attributes: 5

Attributes

No.		Name
1	✓	age
2	✓	income
3	✓	student
4		creditrating
5		buyspc

Selected attribute
Name: student Type: Nominal
Missing: 0 (0%) Distinct: 2 Unique: 0 (0%)

No.	Label	Count
1	yes	7
2	no	7

Class: age (Nom)

Status
OK

Exp.2 Demonstration of preprocessing techniques on dataset labor.arff

Aim: This experiment illustrates some of the basic data preprocessing operations that can be performed using WEKA-Explorer. The sample dataset used for this example is the labor data available in arff format.

Step1:Loading the data. We can load the dataset into weka by clicking on open button in preprocessing interface and selecting the appropriate file.

Step2: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).

Step3: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.,

Step4: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.

Step5:Selecting or filtering attributes

Removing an attribute-When we need to remove an attribute,we can do this by using the attribute filters in weka.In the filter model panel,click on choose button,This will show a popup window with a list of available filters.

Scroll down the list and select the "weka.filters.unsupervised.attribute.remove" filters.

Step 6:a)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.

b) Make sure that invert selection option is set to false.The click OK now in the filter box.you will see "Remove-R-7".

c) Click the apply button to apply filter to this data.This will remove the attribute and create new working relation.

d) Save the new working relation as an arff file by clicking save button on the top(button)panel.(labor.arff)

Discretization

1) 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.

◎Let us divide the values of duration attribute into three bins(intervals).

◎First load the dataset into weka(labor.arff)

◎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.

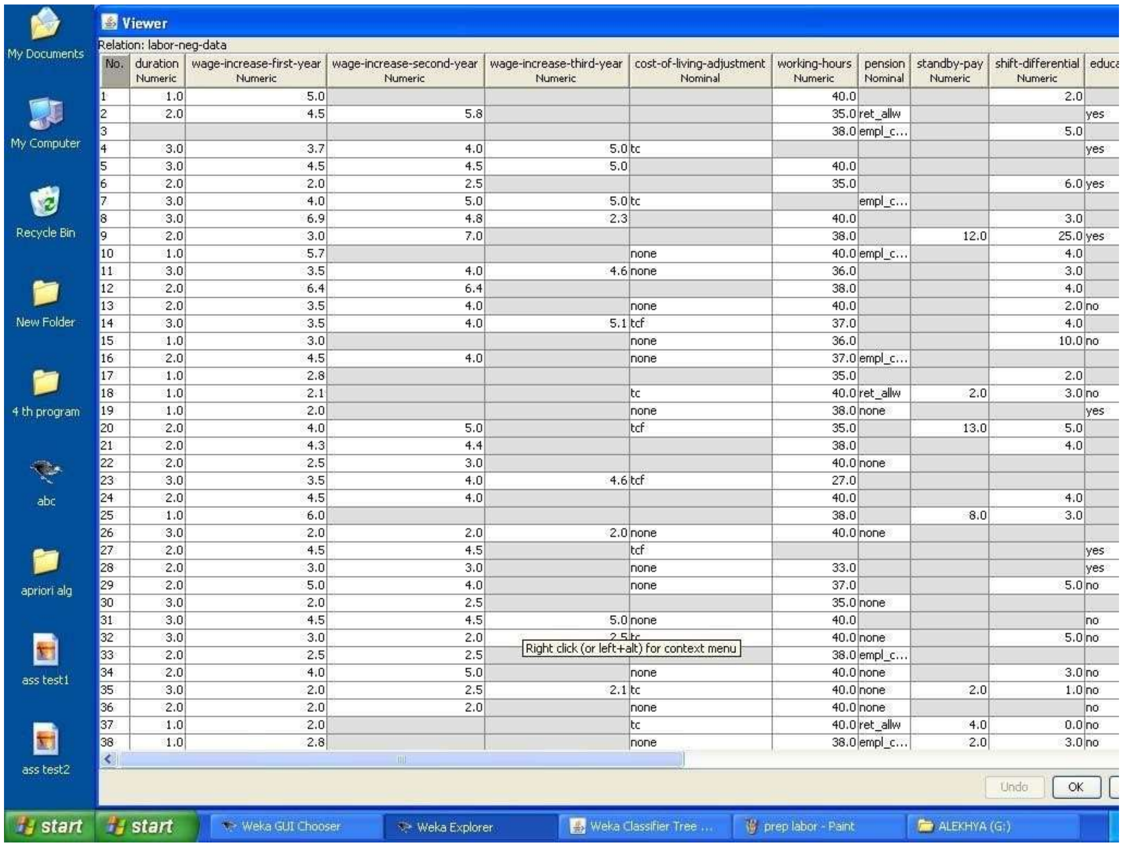
◎Enter '1' as the number of bins.Leave the remaining field values as they are.

◎Click OK button.

◎Click apply in the filter panel.This will result in a new working relation with the selected attribute partition into 1 bin.

◎Save the new working relation in a file called labor-data-discretized.arff

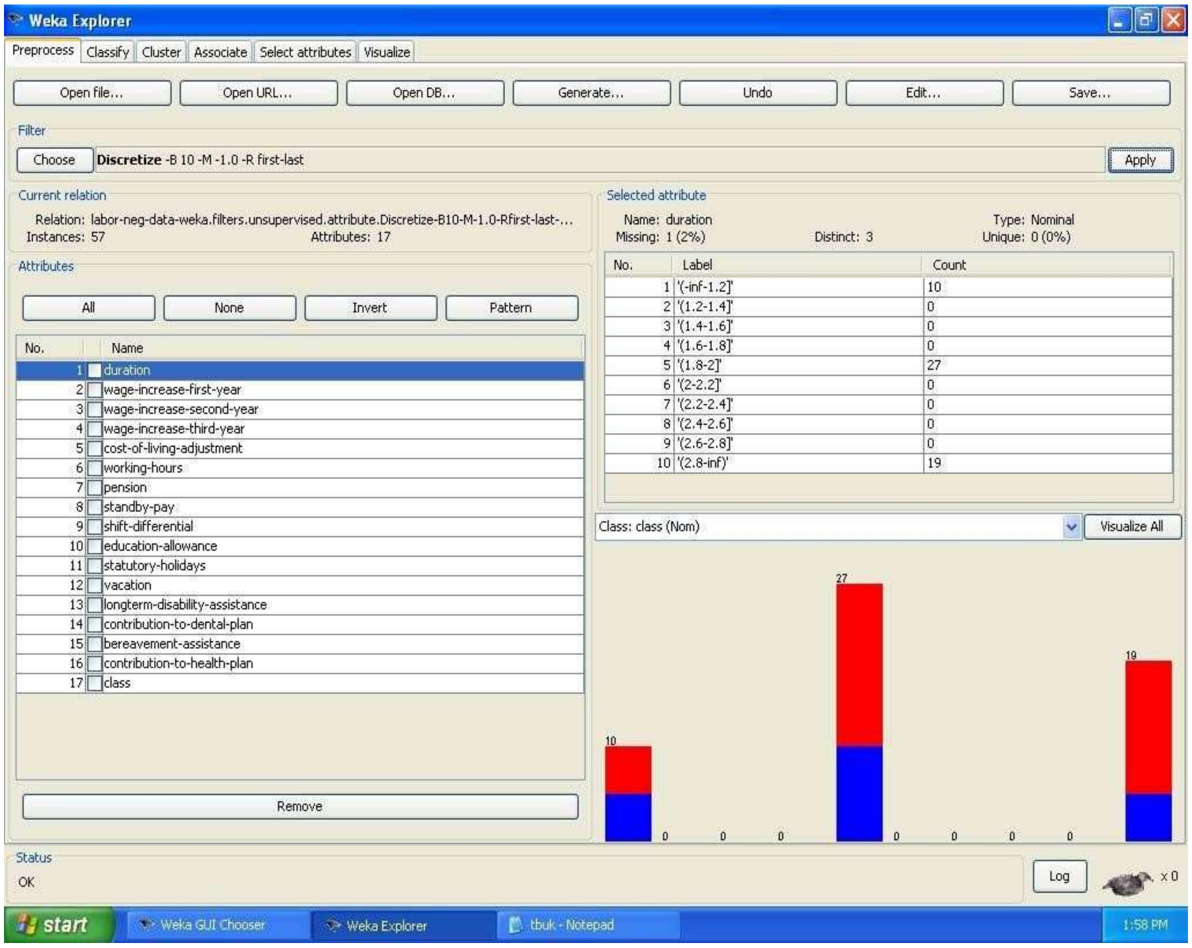
Dataset labor.arff



Relation: labor-neg-data

No.	duration Numeric	wage-increase-first-year Numeric	wage-increase-second-year Numeric	wage-increase-third-year Numeric	cost-of-living-adjustment Nominal	working-hours Numeric	pension Nominal	standby-pay Numeric	shift-differential Numeric	educa
1	1.0	5.0				40.0			2.0	
2	2.0	4.5	5.8			35.0	ret_allw			yes
3						38.0	empl_c...		5.0	
4	3.0	3.7	4.0	5.0	tc					yes
5	3.0	4.5	4.5	5.0		40.0				
6	2.0	2.0	2.5			35.0			6.0	yes
7	3.0	4.0	5.0	5.0	tc		empl_c...			
8	3.0	6.9	4.8	2.3		40.0			3.0	
9	2.0	3.0	7.0			38.0		12.0	25.0	yes
10	1.0	5.7			none	40.0	empl_c...		4.0	
11	3.0	3.5	4.0	4.6	none	36.0			3.0	
12	2.0	6.4	6.4			38.0			4.0	
13	2.0	3.5	4.0		none	40.0			2.0	no
14	3.0	3.5	4.0	5.1	tcf	37.0			4.0	
15	1.0	3.0			none	36.0			10.0	no
16	2.0	4.5	4.0		none	37.0	empl_c...			
17	1.0	2.8				35.0			2.0	
18	1.0	2.1			tc	40.0	ret_allw	2.0	3.0	no
19	1.0	2.0			none	38.0	none			yes
20	2.0	4.0	5.0		tcf	35.0		13.0	5.0	
21	2.0	4.3	4.4			38.0			4.0	
22	2.0	2.5	3.0			40.0	none			
23	3.0	3.5	4.0	4.6	tcf	27.0				
24	2.0	4.5	4.0			40.0			4.0	
25	1.0	6.0				38.0		8.0	3.0	
26	3.0	2.0	2.0	2.0	none	40.0	none			
27	2.0	4.5	4.5		tcf					yes
28	2.0	3.0	3.0		none	33.0				yes
29	2.0	5.0	4.0		none	37.0			5.0	no
30	3.0	2.0	2.5			35.0	none			
31	3.0	4.5	4.5	5.0	none	40.0				no
32	3.0	3.0	2.0	2.5	tc	40.0	none		5.0	no
33	2.0	2.5	2.5			38.0	empl_c...			
34	2.0	4.0	5.0		none	40.0	none		3.0	no
35	3.0	2.0	2.5	2.1	tc	40.0	none	2.0	1.0	no
36	2.0	2.0	2.0		none	40.0	none			no
37	1.0	2.0			tc	40.0	ret_allw	4.0	0.0	no
38	1.0	2.8			none	38.0	empl_c...	2.0	3.0	no

The following screenshot shows the effect of discretization



Discretize -B 10 -M -1.0 -R first-last

Current relation
Relation: labor-neg-data-weka.filters.unsupervised.attribute.Discretize-B10-M-1.0-Rfirst-last-...
Instances: 57 Attributes: 17

Selected attribute
Name: duration Type: Nominal
Missing: 1 (2%) Distinct: 3 Unique: 0 (0%)

No.	Name
1	duration
2	wage-increase-first-year
3	wage-increase-second-year
4	wage-increase-third-year
5	cost-of-living-adjustment
6	working-hours
7	pension
8	standby-pay
9	shift-differential
10	education-allowance
11	statutory-holidays
12	vacation
13	longterm-disability-assistance
14	contribution-to-dental-plan
15	bereavement-assistance
16	contribution-to-health-plan
17	class

No.	Label	Count
1	'(-inf-1.2]'	10
2	'(1.2-1.4]'	0
3	'(1.4-1.6]'	0
4	'(1.6-1.8]'	0
5	'(1.8-2]'	27
6	'(2-2.2]'	0
7	'(2.2-2.4]'	0
8	'(2.4-2.6]'	0
9	'(2.6-2.8]'	0
10	'(2.8-inf)'	19

Class: class (Nom)

```
10  0  0  0  27  0  0  0  0  19
```

Status
OK

x 0

Exp .3 Demonstration of Association rule mining on dataset contactlenses.arff using apriori algorithm

Aim: This experiment illustrates some of the basic elements of asscociation rule mining using WEKA. The sample dataset used for this example is contactlenses.arff

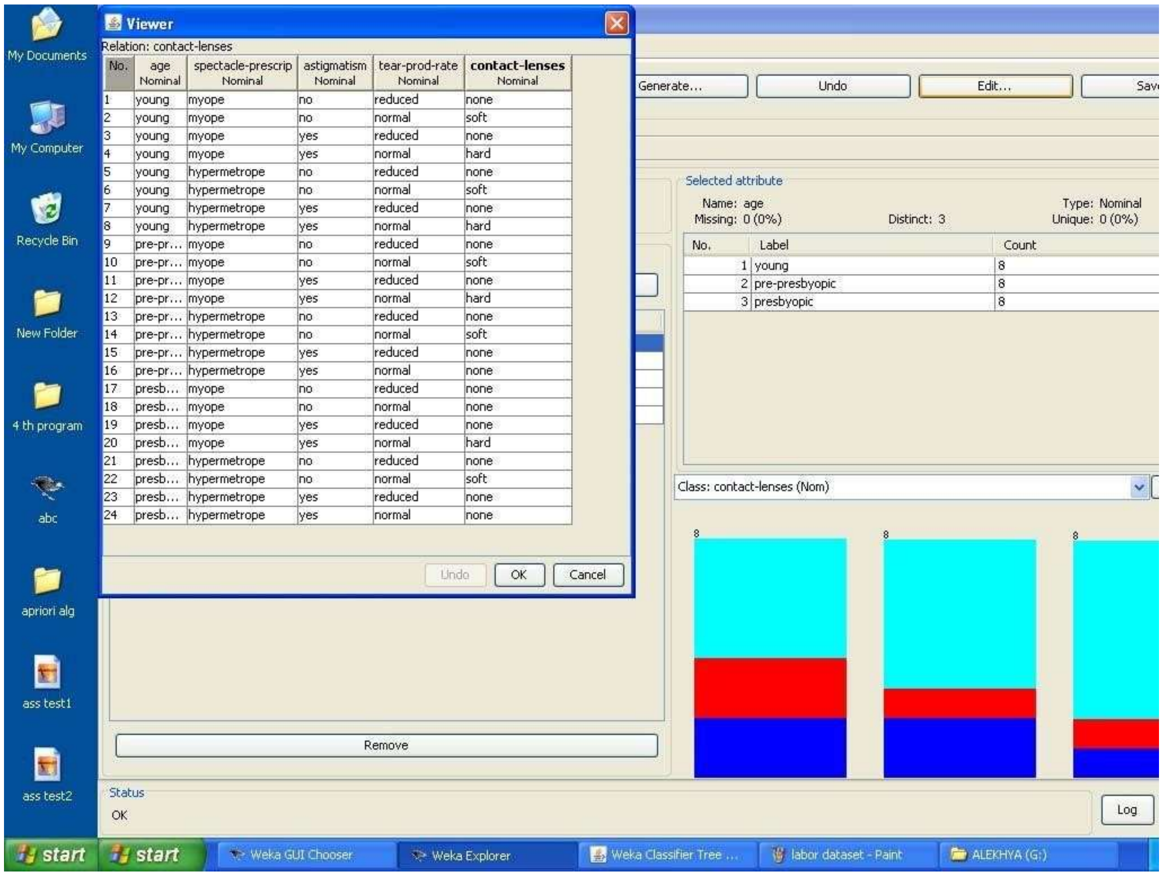
Step1: Open the data file in Weka Explorer. It is presumed that the required data fields have been discretized. In this example it is age attribute.

Step2: Clicking on the associate tab will bring up the interface for association rule algorithm.

Step3: We will use apriori algorithm. This is the default algorithm.

Step4: Inorder to change the parameters for the run (example support, confidence etc) we click on the text box immediately to the right of the choose button.

Dataset contactlenses.arff


Relation: contact-lenses

No.	age Nominal	spectacle-prescrip Nominal	astigmatism Nominal	tear-prod-rate Nominal	contact-lenses Nominal
1	young	myope	no	reduced	none
2	young	myope	no	normal	soft
3	young	myope	yes	reduced	none
4	young	myope	yes	normal	hard
5	young	hypermetrope	no	reduced	none
6	young	hypermetrope	no	normal	soft
7	young	hypermetrope	yes	reduced	none
8	young	hypermetrope	yes	normal	hard
9	pre-pr...	myope	no	reduced	none
10	pre-pr...	myope	no	normal	soft
11	pre-pr...	myope	yes	reduced	none
12	pre-pr...	myope	yes	normal	hard
13	pre-pr...	hypermetrope	no	reduced	none
14	pre-pr...	hypermetrope	no	normal	soft
15	pre-pr...	hypermetrope	yes	reduced	none
16	pre-pr...	hypermetrope	yes	normal	none
17	presb...	myope	no	reduced	none
18	presb...	myope	no	normal	none
19	presb...	myope	yes	reduced	none
20	presb...	myope	yes	normal	hard
21	presb...	hypermetrope	no	reduced	none
22	presb...	hypermetrope	no	normal	soft
23	presb...	hypermetrope	yes	reduced	none
24	presb...	hypermetrope	yes	normal	none

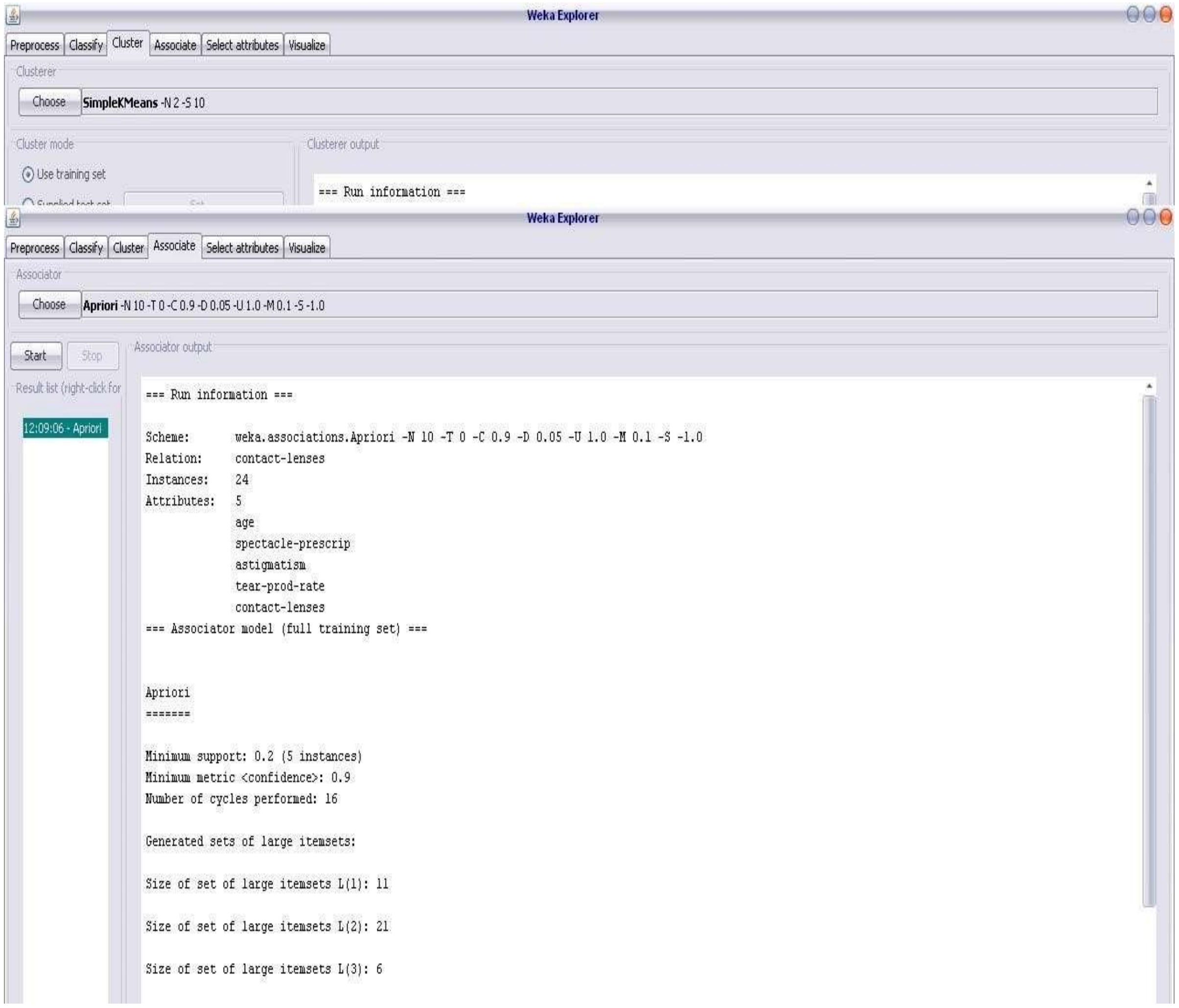
Selected attribute
Name: age
Missing: 0 (0%)
Distinct: 3
Type: Nominal
Unique: 0 (0%)

No.	Label	Count
1	young	8
2	pre-presbyopic	8
3	presbyopic	8

Class: contact-lenses (Nom)

Status
OK

The following screenshot shows the association rules that were generated when apriori algorithm is applied on the given dataset.



Clusterer: SimpleKMeans -N 2 -S 10

Cluster mode: Use training set

Clusterer output

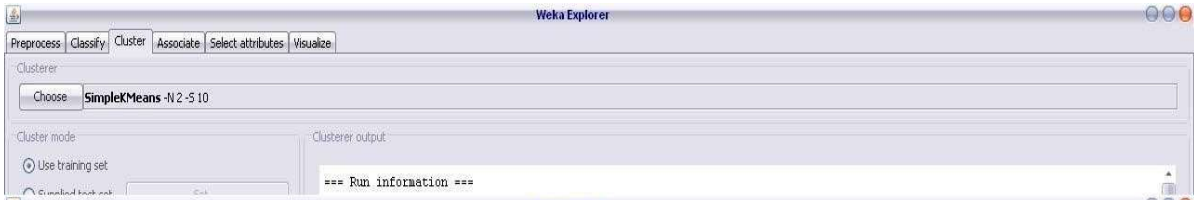
```
=== Run information ===
```

Associator: Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0

Result list (right-click for: 12:09:06 - Apriori

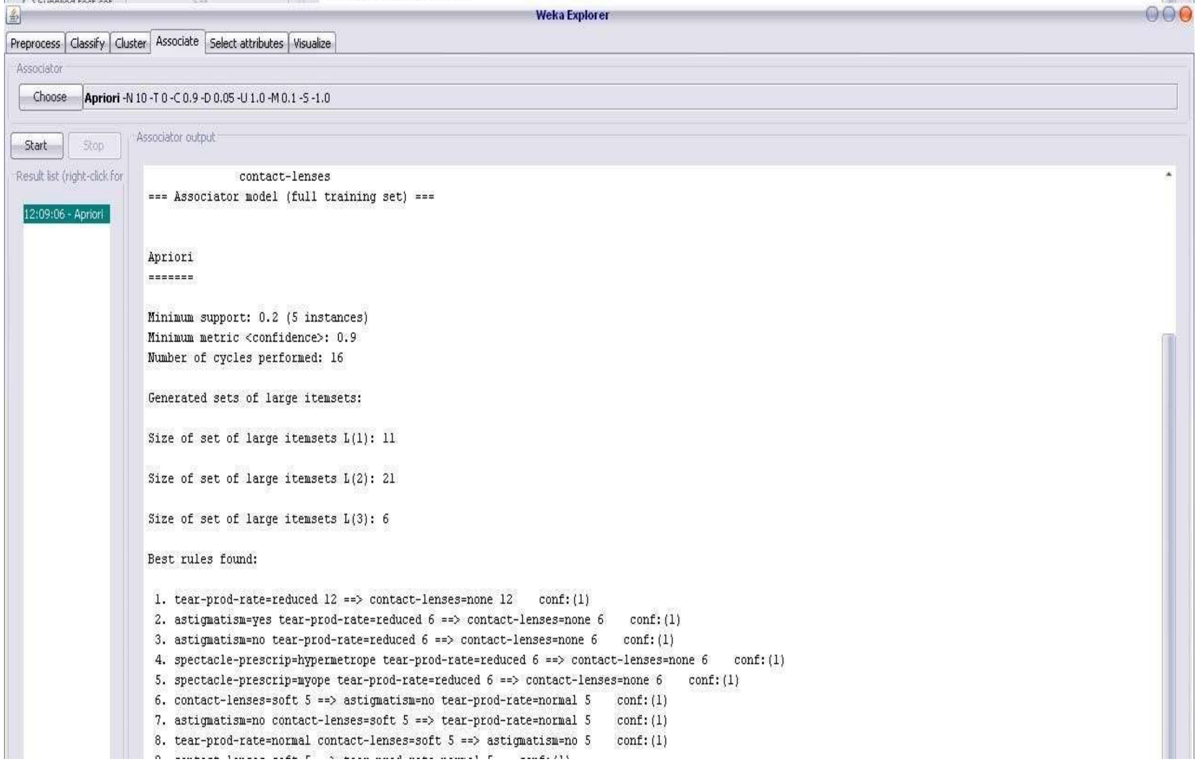
Associator output

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0
Relation:     contact-lenses
Instances:    24
Attributes:   5
              age
              spectacle-prescrip
              astigmatism
              tear-prod-rate
              contact-lenses
=== Associator model (full training set) ===


Apriori
=======

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

Generated sets of large itemsets:

Size of set of large itemsets L(1): 11

Size of set of large itemsets L(2): 21

Size of set of large itemsets L(3): 6
```



SimpleKMeans -N 2 -S 10

```
=== Run information ===
```



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

```
              contact-lenses
=== Associator model (full training set) ===


Apriori
=======

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

Generated sets of large itemsets:

Size of set of large itemsets L(1): 11

Size of set of large itemsets L(2): 21

Size of set of large itemsets L(3): 6

Best rules found:

 1. tear-prod-rate=reduced 12 ==> contact-lenses=none 12    conf:(1)
 2. astigmatism=yes tear-prod-rate=reduced 6 ==> contact-lenses=none 6    conf:(1)
 3. astigmatism=no tear-prod-rate=reduced 6 ==> contact-lenses=none 6    conf:(1)
 4. spectacle-prescrip=hypermetrope tear-prod-rate=reduced 6 ==> contact-lenses=none 6    conf:(1)
 5. spectacle-prescrip=myope tear-prod-rate=reduced 6 ==> contact-lenses=none 6    conf:(1)
 6. contact-lenses=soft 5 ==> astigmatism=no tear-prod-rate=normal 5    conf:(1)
 7. astigmatism=no contact-lenses=soft 5 ==> tear-prod-rate=normal 5    conf:(1)
 8. tear-prod-rate=normal contact-lenses=soft 5 ==> astigmatism=no 5    conf:(1)
```

Exp.4 Demonstration of Association rule mining on dataset test.arff using apriori algorithm.

Aim: This experiment illustrates some of the basic elements of asscociation rule mining using WEKA. The sample dataset used for this example is test.arff

Step1: Open the data file in Weka Explorer. It is presumed that the required data fields have been discretized. In this example it is age attribute.

Step2: Clicking on the associate tab will bring up the interface for association rule algorithm.

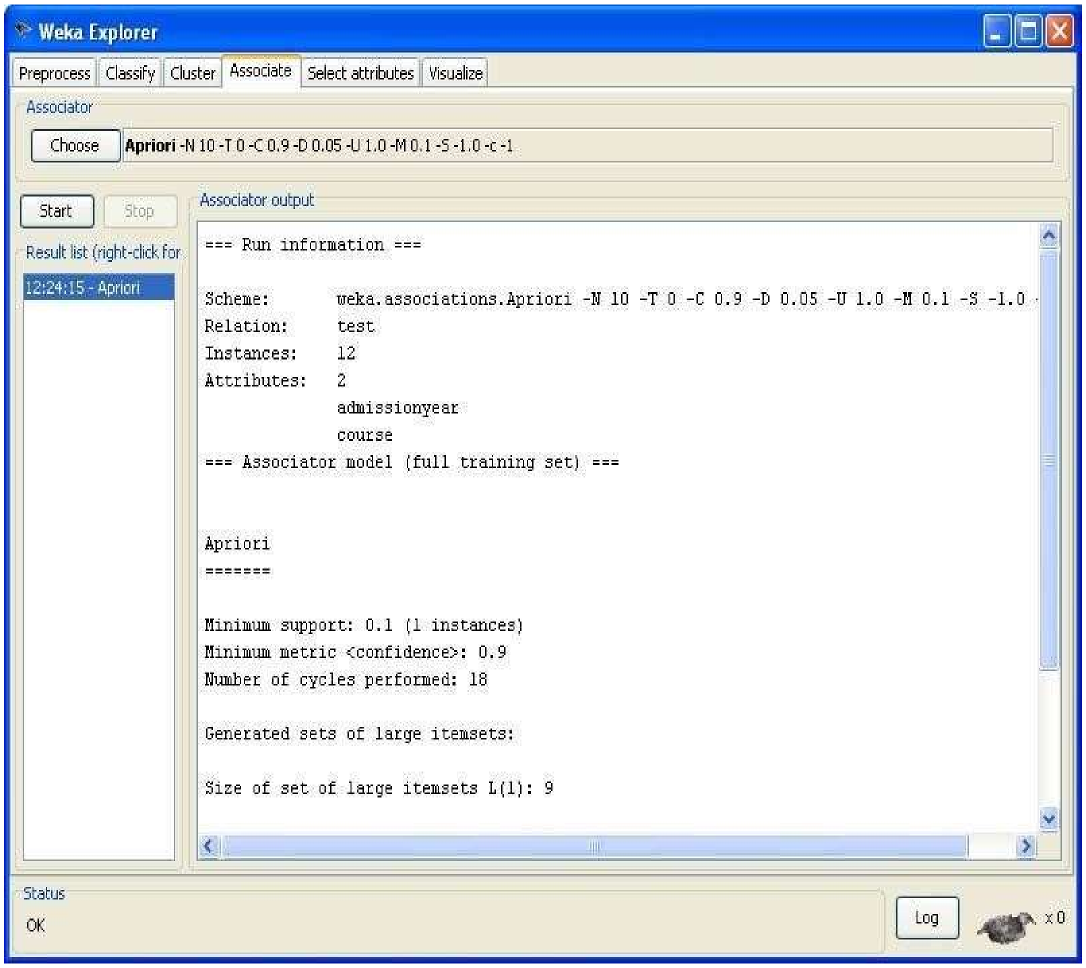
Step3: We will use apriori algorithm. This is the default algorithm.

Step4: Inorder to change the parameters for the run (example support, confidence etc) we click on the text box immediately to the right of the choose button.

Dataset test.arff

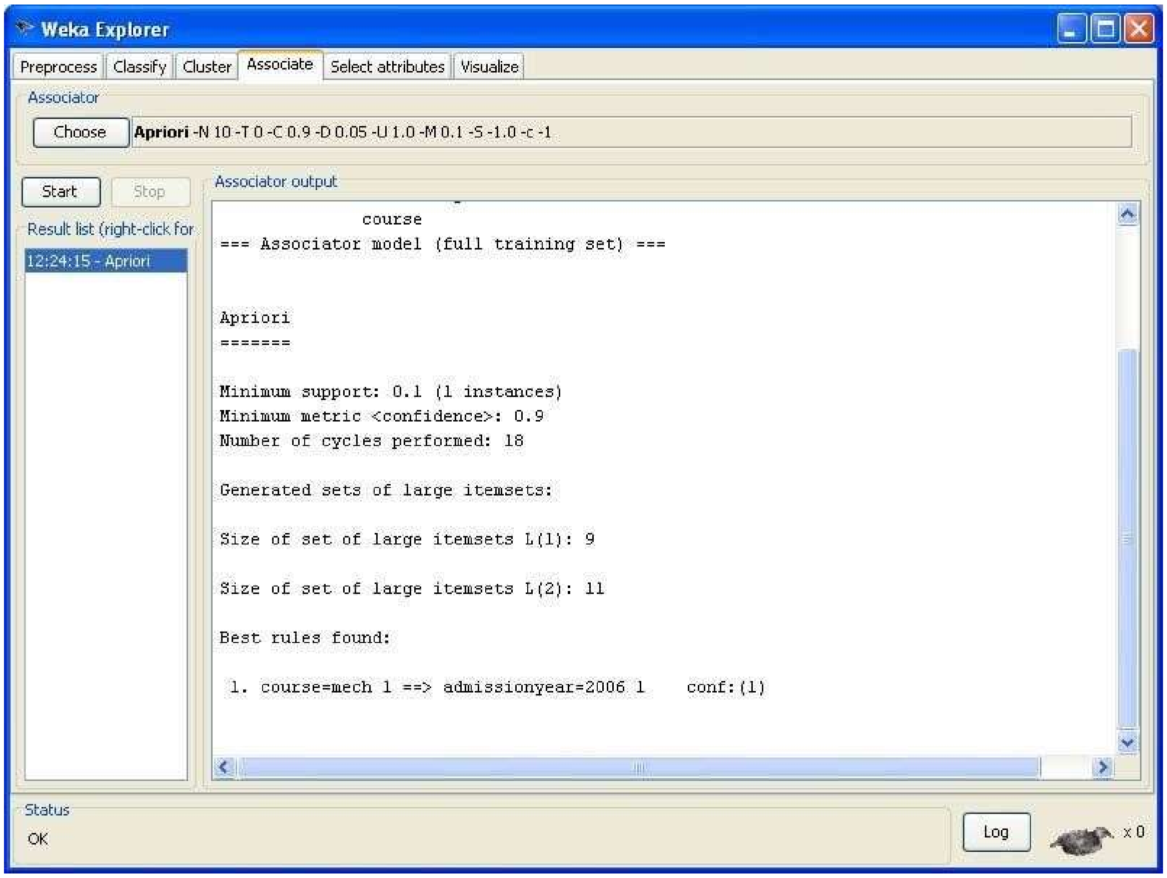
```
@relation test

@attribute admissionyear {2005,2006,2007,2008,2009,2010}

@attribute course {cse,mech,it,ece}

@data

%

2005, cse

2005, it

2005, cse

2006, mech

2006, it

2006, ece

2007, it

2007, cse

2008, it

2008, cse

2009, it

2009, ece

%
```

The following screenshot shows the association rules that were generated when apriori algorithm is applied on the given dataset.



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

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 
Relation:     test
Instances:    12
Attributes:   2
              admissionyear
              course
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.1 (1 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 18

Generated sets of large itemsets:

Size of set of large itemsets L(1): 9
```



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

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


Apriori
=======

Minimum support: 0.1 (1 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 18

Generated sets of large itemsets:

Size of set of large itemsets L(1): 9

Size of set of large itemsets L(2): 11

Best rules found:

 1. course=mech 1 ==> admissionyear=2006 1    conf:(1)
```

Exp. 5 Demonstration of classification process on dataset student.arff using j48 algorithm.

Aim: This experiment illustrates the use of j-48 classifier in weka. The sample data set used in this experiment is “student” data available at arff format. This document assumes that appropriate data pre processing has been performed.

Steps involved in this experiment:

Step-1: We begin the experiment by loading the data (student.arff)into weka.

Step2: Next we select the “classify” tab and click “choose” button t o select the “j48”classifier.

Step3: Now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button. In this example, we accept the default values. The default version does perform some pruning but does not perform error pruning.

Step4: Under the “text” options in the main panel. We select the 10-fold cross validation as our evaluation approach. Since we don’t have separate evaluation data set, this is necessary to get a reasonable idea of accuracy of generated model.

Step-5: We now click ”start” to generate the model .the Ascii version of the tree as well as evaluation statistic will appear in the right panel when the model construction is complete.

Step-6: Note that the classification accuracy of model is about 69%.this indicates that we may find more work. (Either in preprocessing or in selecting current parameters for the classification)

Step-7: Now weka also lets us a view a graphical version of the classification tree. This can be done by right clicking the last result set and selecting “visualize tree” from the pop-up menu.

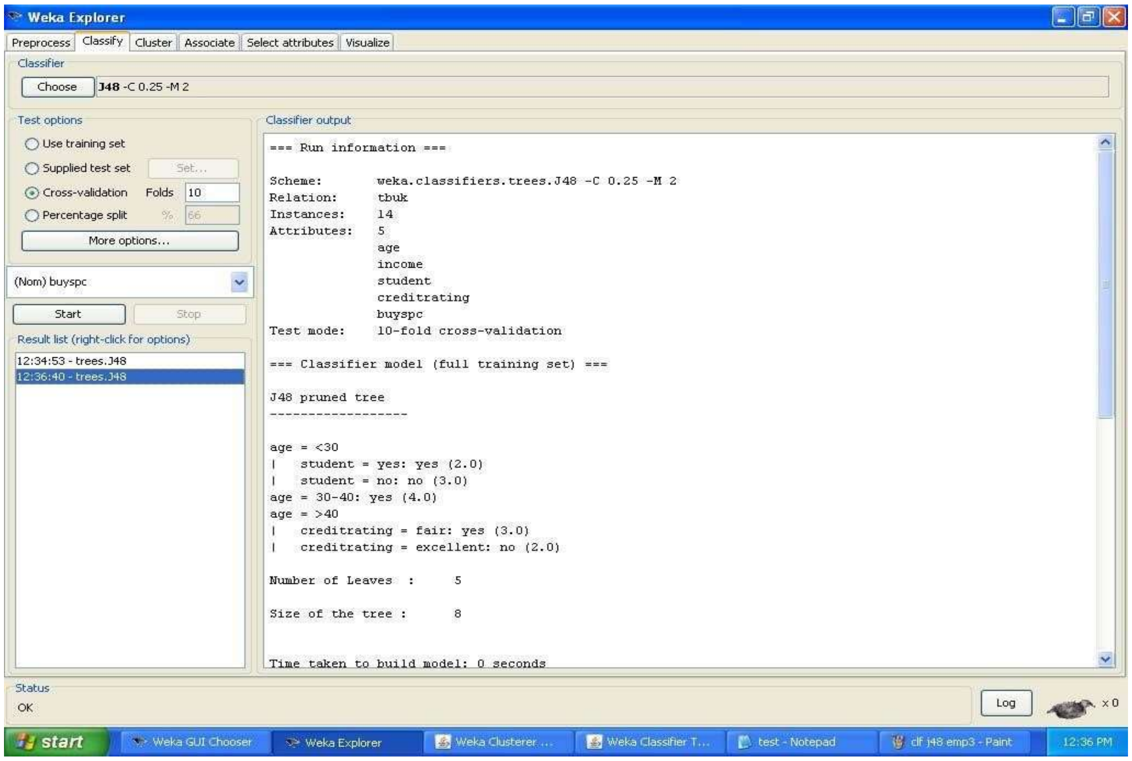
Step-8: We will use our model to classify the new instances.

Step-9: In the main panel under “text” options click the “supplied test set” radio button and then click the “set” button. This wills pop-up a window which will allow you to open the file containing test instances.

Dataset student .arff

```
@relation student

@attribute age {<30,30-40,>40}

@attribute income {low, medium, high}

@attribute student {yes, no}

@attribute credit-rating {fair, excellent}

@attribute buyspc {yes, no}

@data

%

<30, high, no, fair, no

<30, high, no, excellent, no

30-40, high, no, fair, yes

>40, medium, no, fair, yes

>40, low, yes, fair, yes

>40, low, yes, excellent, no

30-40, low, yes, excellent, yes

<30, medium, no, fair, no

<30, low, yes, fair, no

>40, medium, yes, fair, yes

<30, medium, yes, excellent, yes

30-40, medium, no, excellent, yes

30-40, high, yes, fair, yes

>40, medium, no, excellent, no

%
```

The following screenshot shows the classification rules that were generated when j48 algorithm is applied on the given dataset.



Classifier: J48 -C 0.25 -M 2

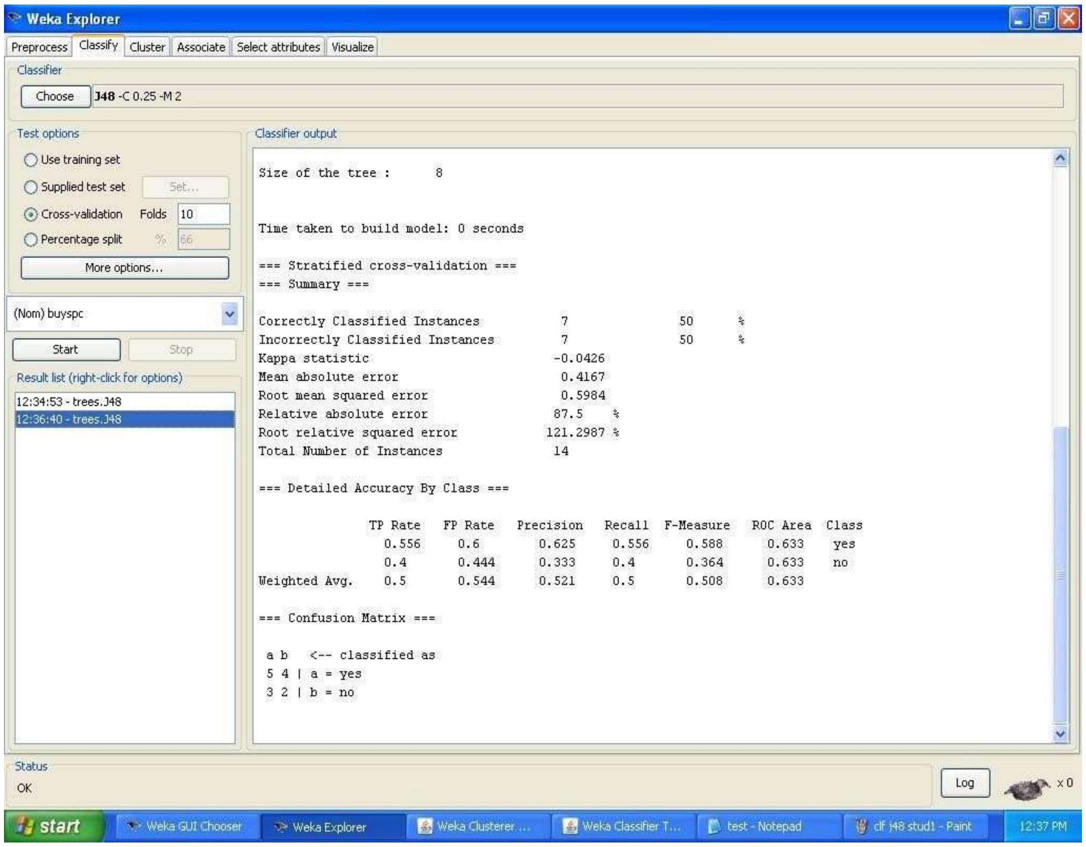
Test options: Cross-validation [✓] Folds 10; Percentage split % 66

(Nom) buyspc

Result list:
12:34:53 - trees.J48
12:36:40 - trees.J48

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     tbuk
Instances:    14
Attributes:   5
              age
              income
              student
              creditrating
              buyspc
Test mode:    10-fold cross-validation

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

J48 pruned tree
------------------

age = <30
|   student = yes: yes (2.0)
|   student = no: no (3.0)
age = 30-40: yes (4.0)
age = >40
|   creditrating = fair: yes (3.0)
|   creditrating = excellent: no (2.0)

Number of Leaves  :     5

Size of the tree :      8


Time taken to build model: 0 seconds
```

Status: OK



J48 -C 0.25 -M 2

Cross-validation Folds 10

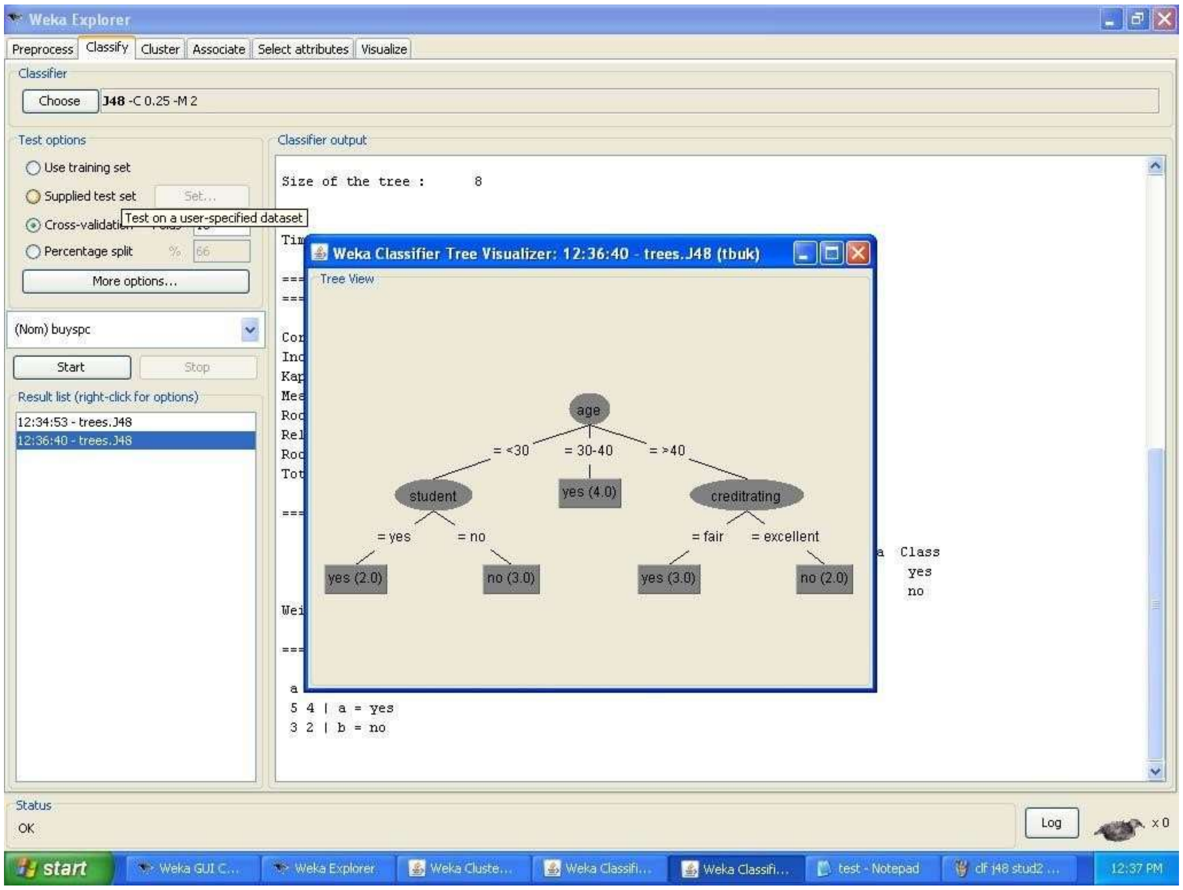
% 66

(Nom) buyspc

12:34:53 - trees.J48
12:36:40 - trees.J48

```
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   ROC Area  Class
                 0.556     0.6        0.625     0.556     0.588      0.633    yes
                 0.4       0.444      0.333     0.4       0.364      0.633    no
Weighted Avg.    0.5       0.544      0.521     0.5       0.508      0.633

=== Confusion Matrix ===

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

OK



Classifier

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

Test options

- Use training set
- Supplied test set   Set...
- [✓ Cross-validation] Folds 10
- Percentage split % 66

Test on a user-specified dataset

More options...

(Nom) buyspc

Start   Stop

Result list (right-click for options)

```
12:34:53 - trees.J48
12:36:40 - trees.J48
```

Classifier output

```
Size of the tree :       8

Tim
===
===

Cor
Inc
Kap
Mea
Roo
Rel
Roo
Tot

===
                                                     a   Class
                                                          yes
                                                          no

Wei

===

 a
 5 4 | a = yes
 3 2 | b = no
```

Status

OK

Log   x 0

Exp.6 Demonstration of classification process on dataset employee.arff using j48 algorithm.

Aim: This experiment illustrates the use of j-48 classifier in weka.the sample data set used in this experiment is “employee”data available at arff format. This document assumes that appropriate data pre processing has been performed.

Steps involved in this experiment:

Step 1: We begin the experiment by loading the data (employee.arff) into weka.

Step2: Next we select the “classify” tab and click “choose” button to select the “j48”classifier.

Step3: Now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button. In this example, we accept the default values the default version does perform some pruning but does not perform error pruning.

Step4: Under the “text “options in the main panel. We select the 10-fold cross validation as our evaluation approach. Since we don’t have separate evaluation data set, this is necessary to get a reasonable idea of accuracy of generated model.

Step-5: We now click ”start” to generate the model .the ASCII version of the tree as well as evaluation statistic will appear in the right panel when the model construction is complete.

Step-6: Note that the classification accuracy of model is about 69%.this indicates that we may find more work. (Either in preprocessing or in selecting current parameters for the classification)

Step-7: Now weka also lets us a view a graphical version of the classification tree. This can be done by right clicking the last result set and selecting “visualize tree” from the pop-up menu.

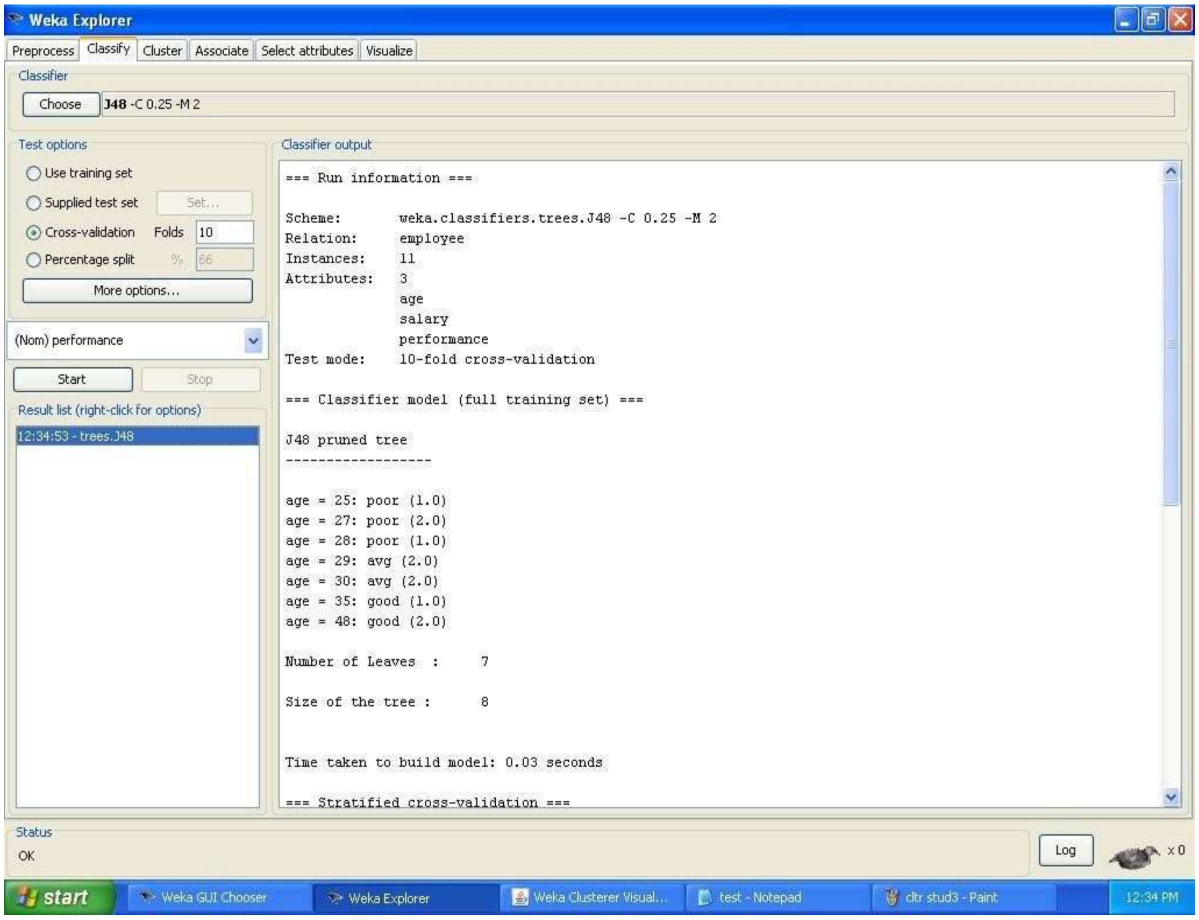
Step-8: We will use our model to classify the new instances.

Step-9: In the main panel under “text “options click the “supplied test set” radio button and then click the “set” button. This wills pop-up a window which will allow you to open the file containing test instances.

Data set employee.arff:

```
@relation employee

@attribute age {25, 27, 28, 29, 30, 35, 48}

@attribute salary{10k,15k,17k,20k,25k,30k,35k,32k}

@attribute performance {good, avg, poor}

@data

%

25, 10k, poor

27, 15k, poor

27, 17k, poor

28, 17k, poor

29, 20k, avg

30, 25k, avg

29, 25k, avg

30, 20k, avg

35, 32k, good

48, 34k, good

48, 32k,good

%
```

The following screenshot shows the classification rules that were generated whenj48 algorithm is applied on the given dataset.



J48 -C 0.25 -M 2

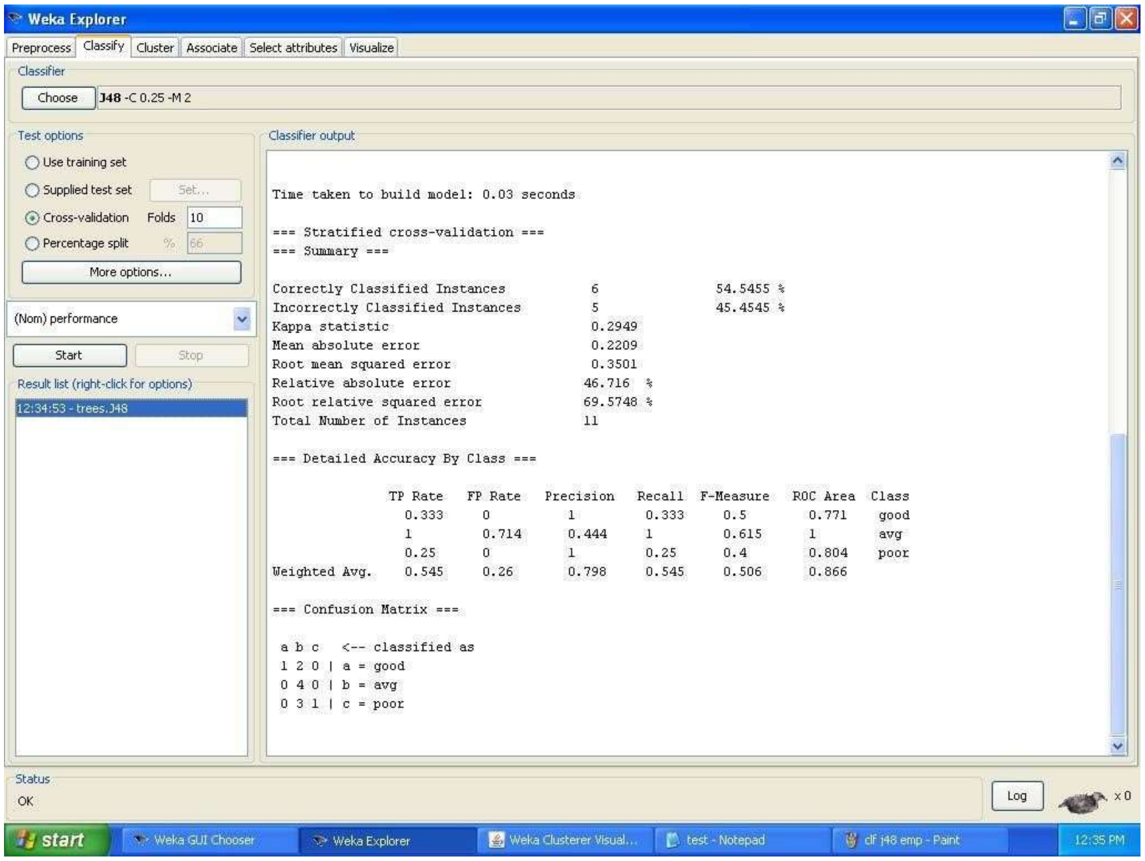
Cross-validation Folds 10

Percentage split % 66

(Nom) performance

12:34:53 - trees.J48

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     employee
Instances:    11
Attributes:   3
              age
              salary
              performance
Test mode:    10-fold cross-validation

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

J48 pruned tree
------------------

age = 25: poor (1.0)
age = 27: poor (2.0)
age = 28: poor (1.0)
age = 29: avg (2.0)
age = 30: avg (2.0)
age = 35: good (1.0)
age = 48: good (2.0)

Number of Leaves  :     7

Size of the tree :      8


Time taken to build model: 0.03 seconds

=== Stratified cross-validation ===
```



J48 -C 0.25 -M 2

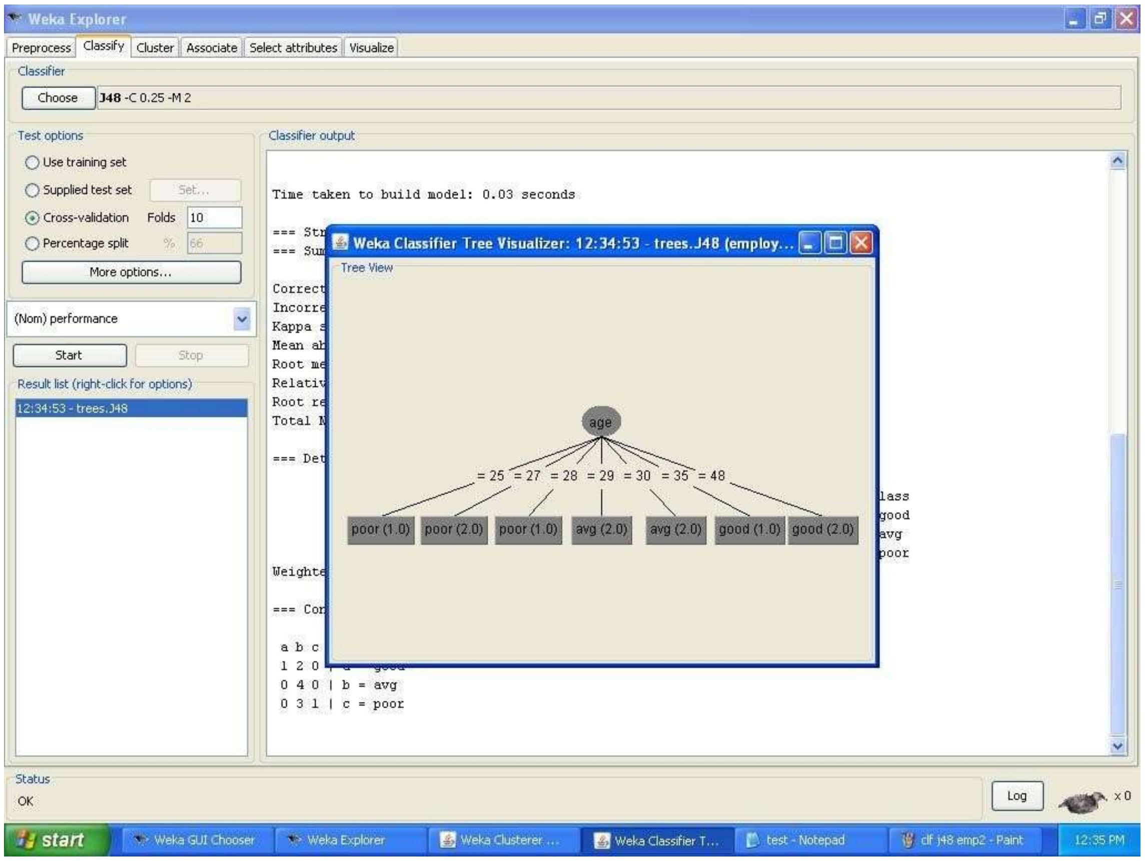
Use training set
Supplied test set
[✓ Cross-validation] Folds 10
Percentage split % 66

(Nom) performance

12:34:53 - trees.J48

```
Time taken to build model: 0.03 seconds

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

Correctly Classified Instances           6               54.5455 %
Incorrectly Classified Instances         5               45.4545 %
Kappa statistic                          0.2949
Mean absolute error                      0.2209
Root mean squared error                  0.3501
Relative absolute error                 46.716  %
Root relative squared error             69.5748 %
Total Number of Instances               11

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.333     0          1         0.333     0.5        0.771    good
                 1         0.714      0.444     1         0.615      1        avg
                 0.25      0          1         0.25      0.4        0.804    poor
Weighted Avg.    0.545     0.26       0.798     0.545     0.506      0.866

=== Confusion Matrix ===

 a b c   <-- classified as
 1 2 0 | a = good
 0 4 0 | b = avg
 0 3 1 | c = poor
```

OK



Classifier

Choose J48 -C 0.25 -M 2

Test options

Use training set

Supplied test set

[✓ Cross-validation] Folds 10

Percentage split % 66

More options...

(Nom) performance

Start Stop

Result list (right-click for options)

12:34:53 - trees.J48

Classifier output

```
Time taken to build model: 0.03 seconds

=== Str
=== Sum

Correct
Incorre
Kappa s
Mean ab
Root me
Relativ
Root re
Total N

=== Det

                                                     lass
                                                     good
                                                     avg
                                                     poor

Weighte

=== Con

 a b c
 1 2 0
 0 4 0 | b = avg
 0 3 1 | c = poor
```

Status

OK

Log x 0

Exp.7 Demonstration of classification process on dataset employee.arff using id3 algorithm.

Aim: This experiment illustrates the use of id3 classifier in weka. The sample data set used in this experiment is “employee”data available at arff format. This document assumes that appropriate data pre processing has been performed.

Steps involved in this experiment:

1. We begin the experiment by loading the data (employee.arff) into weka.

Step2: next we select the “classify” tab and click “choose” button to select the “id3”classifier.

Step3: now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button. In this example, we accept the default values his default version does perform some pruning but does not perform error pruning.

Step4: under the “text “options in the main panel. We select the 10-fold cross validation as our evaluation approach. Since we don’t have separate evaluation data set, this is necessary to get a reasonable idea of accuracy of generated model.

Step-5: we now click”start”to generate the model .the ASCII version of the tree as well as evaluation statistic will appear in the right panel when the model construction is complete.

Step-6: note that the classification accuracy of model is about 69%.this indicates that we may find more work. (Either in preprocessing or in selecting current parameters for the classification)

Step-7: now weka also lets us a view a graphical version of the classification tree. This can be done by right clicking the last result set and selecting “visualize tree” from the pop-up menu.

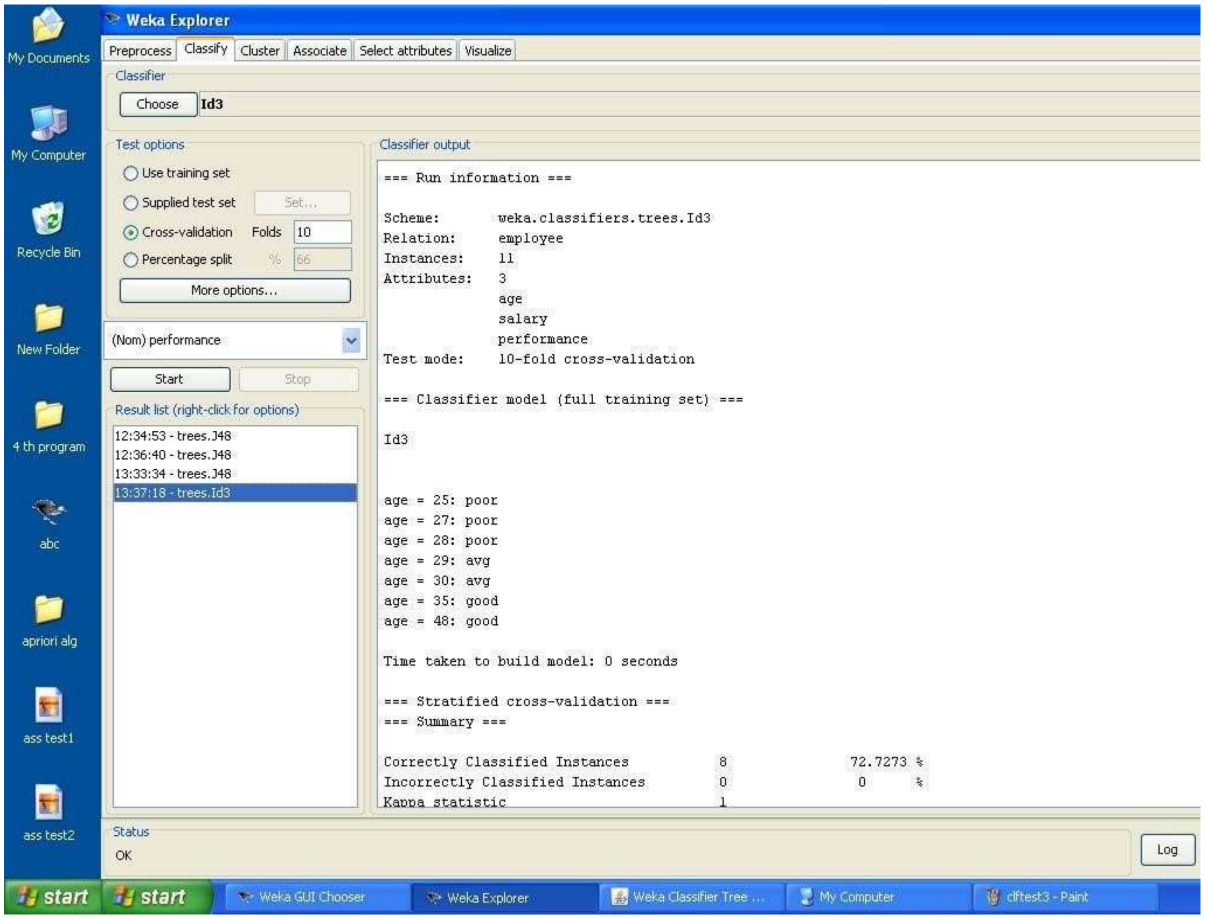
Step-8: we will use our model to classify the new instances.

Step-9: In the main panel under “text “options click the “supplied test set” radio button and then click the “set” button. This will show pop-up window which will allow you to open the file containing test instances.

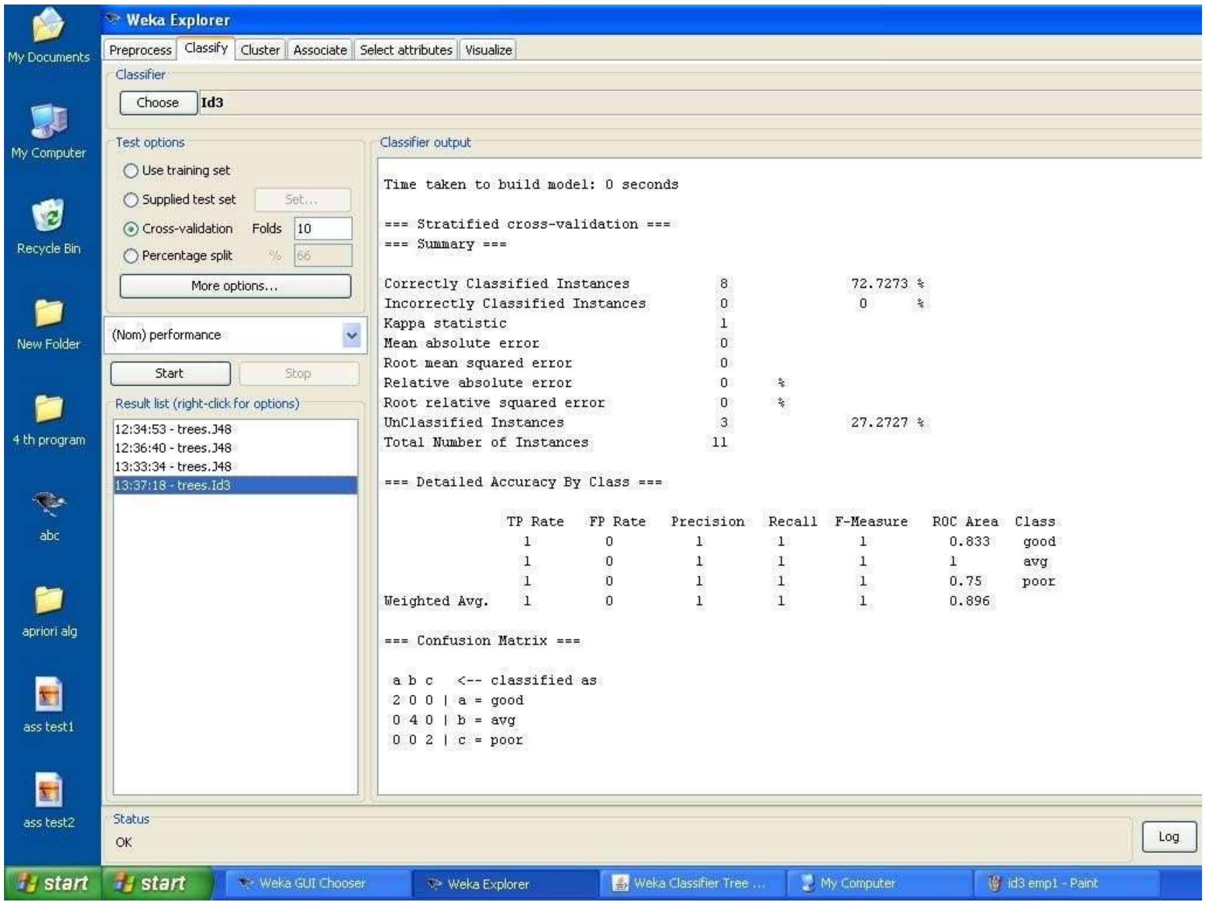
Data set employee.arff:

```
@relation employee

@attribute age {25, 27, 28, 29, 30, 35, 48}

@attribute salary{10k,15k,17k,20k,25k,30k,35k,32k}

@attribute performance {good, avg, poor}

@data

%

25, 10k, poor

27, 15k, poor

27, 17k, poor

28, 17k, poor

29, 20k, avg

30, 25k, avg

29, 25k, avg

30, 20k, avg

35, 32k, good

48, 34k, good

48, 32k, good

%
```

The following screenshot shows the classification rules that were generated when id3 algorithm is applied on the given dataset.



```
=== Run information ===

Scheme:       weka.classifiers.trees.Id3
Relation:     employee
Instances:    11
Attributes:   3
              age
              salary
              performance
Test mode:    10-fold cross-validation

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

Id3


age = 25: poor
age = 27: poor
age = 28: poor
age = 29: avg
age = 30: avg
age = 35: good
age = 48: good

Time taken to build model: 0 seconds

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

Correctly Classified Instances          8               72.7273 %
Incorrectly Classified Instances        0                0      %
Kappa statistic                         1
```



```
Time taken to build model: 0 seconds

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

Correctly Classified Instances           8              72.7273 %
Incorrectly Classified Instances         0               0      %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0      %
Root relative squared error              0      %
UnClassified Instances                   3              27.2727 %
Total Number of Instances               11

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         0         1          1        1          0.833    good
                 1         0         1          1        1          1        avg
                 1         0         1          1        1          0.75     poor
Weighted Avg.    1         0         1          1        1          0.896

=== Confusion Matrix ===

 a b c   <-- classified as
 2 0 0 | a = good
 0 4 0 | b = avg
 0 0 2 | c = poor
```

Exp.8 Demonstration of classification process on dataset employee.arff using naïve bayes algorithm.

Aim: This experiment illustrates the use of naïve bayes classifier in weka. The sample data set used in this experiment is “employee”data available at arff format. This document assumes that appropriate data pre processing has been performed.

Steps involved in this experiment:

1. We begin the experiment by loading the data (employee.arff) into weka.

Step2: next we select the “classify” tab and click “choose” button to select the “id3”classifier.

Step3: now we specify the various parameters. These can be specified by clicking in the text box to the right of the chose button. In this example, we accept the default values his default version does perform some pruning but does not perform error pruning.

Step4: under the “text “options in the main panel. We select the 10-fold cross validation as our evaluation approach. Since we don’t have separate evaluation data set, this is necessary to get a reasonable idea of accuracy of generated model.

Step-5: we now click”start”to generate the model .the ASCII version of the tree as well as evaluation statistic will appear in the right panel when the model construction is complete.

Step-6: note that the classification accuracy of model is about 69%.this indicates that we may find more work. (Either in preprocessing or in selecting current parameters for the classification)

Step-7: now weka also lets us a view a graphical version of the classification tree. This can be done by right clicking the last result set and selecting “visualize tree” from the pop-up menu.

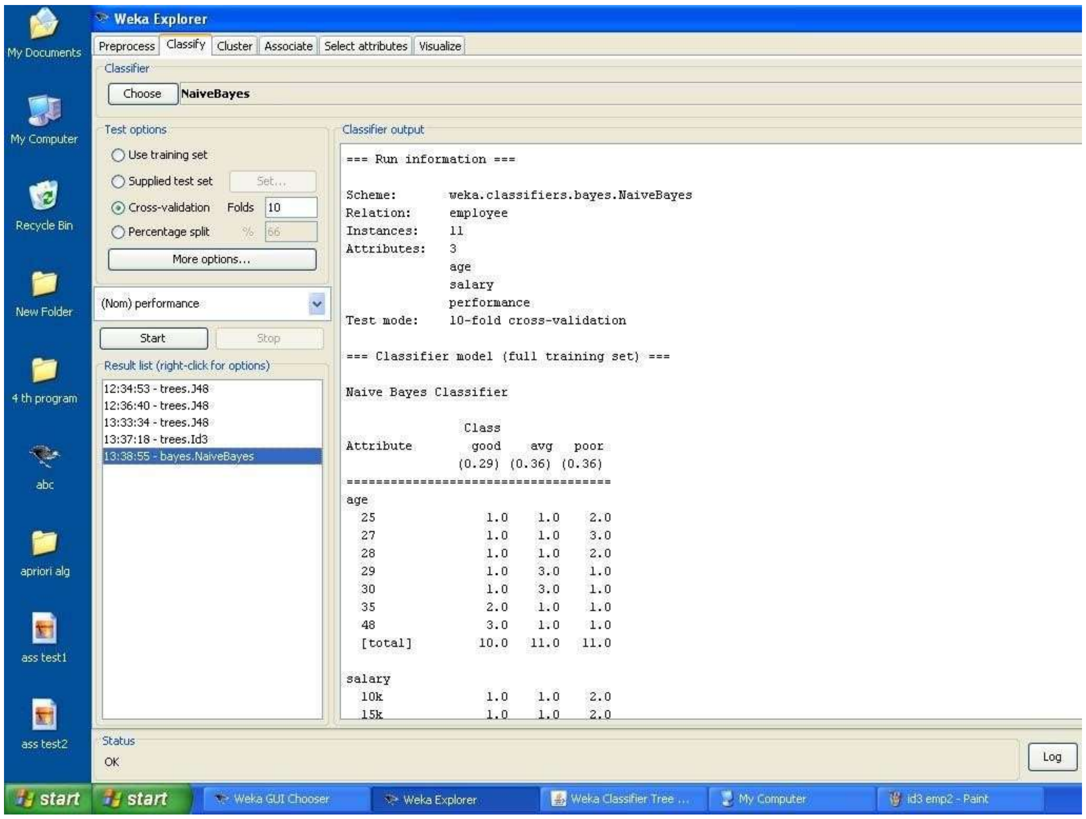
Step-8: we will use our model to classify the new instances.

Step-9: In the main panel under “text “options click the “supplied test set” radio button and then click the “set” button. This will show pop-up window which will allow you to open the file containing test instances.

Data set employee.arff:

```
@relation employee

@attribute age {25, 27, 28, 29, 30, 35, 48}

@attribute salary{10k,15k,17k,20k,25k,30k,35k,32k}

@attribute performance {good, avg, poor}

@data

%

25, 10k, poor

27, 15k, poor

27, 17k, poor

28, 17k, poor

29, 20k, avg

30, 25k, avg

29, 25k, avg

30, 20k, avg

35, 32k, good

48, 34k, good

48, 32k, good

%
```

The following screenshot shows the classification rules that were generated when naive bayes algorithm is applied on the given dataset.



Classifier: NaiveBayes

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

(Nom) performance

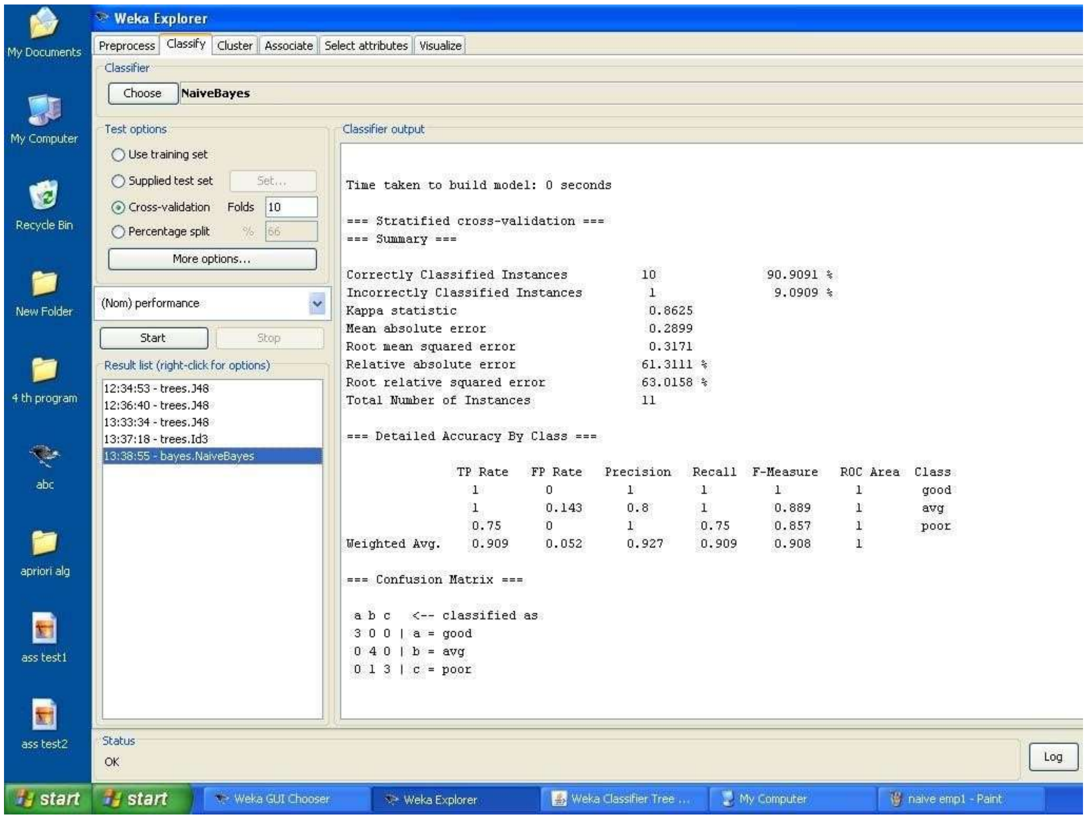
Result list (right-click for options)

12:34:53 - trees.J48
12:36:40 - trees.J48
13:33:34 - trees.J48
13:37:18 - trees.Id3
13:38:55 - bayes.NaiveBayes

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     employee
Instances:    11
Attributes:   3
              age
              salary
              performance
Test mode:    10-fold cross-validation

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

Naive Bayes Classifier

                 Class
Attribute         good    avg   poor
                 (0.29) (0.36) (0.36)
====================================
age
  25               1.0    1.0    2.0
  27               1.0    1.0    3.0
  28               1.0    1.0    2.0
  29               1.0    3.0    1.0
  30               1.0    3.0    1.0
  35               2.0    1.0    1.0
  48               3.0    1.0    1.0
  [total]         10.0   11.0   11.0

salary
  10k              1.0    1.0    2.0
  15k              1.0    1.0    2.0
```

Status
OK



Classifier: NaiveBayes

Cross-validation Folds 10

(Nom) performance

```
12:34:53 - trees.J48
12:36:40 - trees.J48
13:33:34 - trees.J48
13:37:18 - trees.Id3
13:38:55 - bayes.NaiveBayes
```

```
Time taken to build model: 0 seconds

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

Correctly Classified Instances          10               90.9091 %
Incorrectly Classified Instances         1                9.0909 %
Kappa statistic                          0.8625
Mean absolute error                      0.2899
Root mean squared error                  0.3171
Relative absolute error                 61.3111 %
Root relative squared error             63.0158 %
Total Number of Instances               11

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         0          1         1         1          1        good
                 1         0.143      0.8       1         0.889      1        avg
                 0.75      0          1         0.75      0.857      1        poor
Weighted Avg.    0.909     0.052      0.927     0.909     0.908      1

=== Confusion Matrix ===

 a b c   <-- classified as
 3 0 0 | a = good
 0 4 0 | b = avg
 0 1 3 | c = poor
```

Exp. 9 Demonstration of clustering process on dataset iris.arff using simple k-means algorithm.

Aim: This experiment illustrates the use of simple k-mean clustering with Weka explorer. The sample data set used for this example is based on the iris data available in ARFF format. This document assumes that appropriate preprocessing has been performed. This iris dataset includes 150 instances.

Steps involved in this Experiment

Step 1: Run the Weka explorer and load the data file iris.arff in preprocessing interface.

Step 2: Inorder to perform clustering select the 'cluster' tab in the explorer and click on the choose button. This step results in a dropdown list of available clustering algorithms.

Step 3 : In this case we select 'simple k-means'.

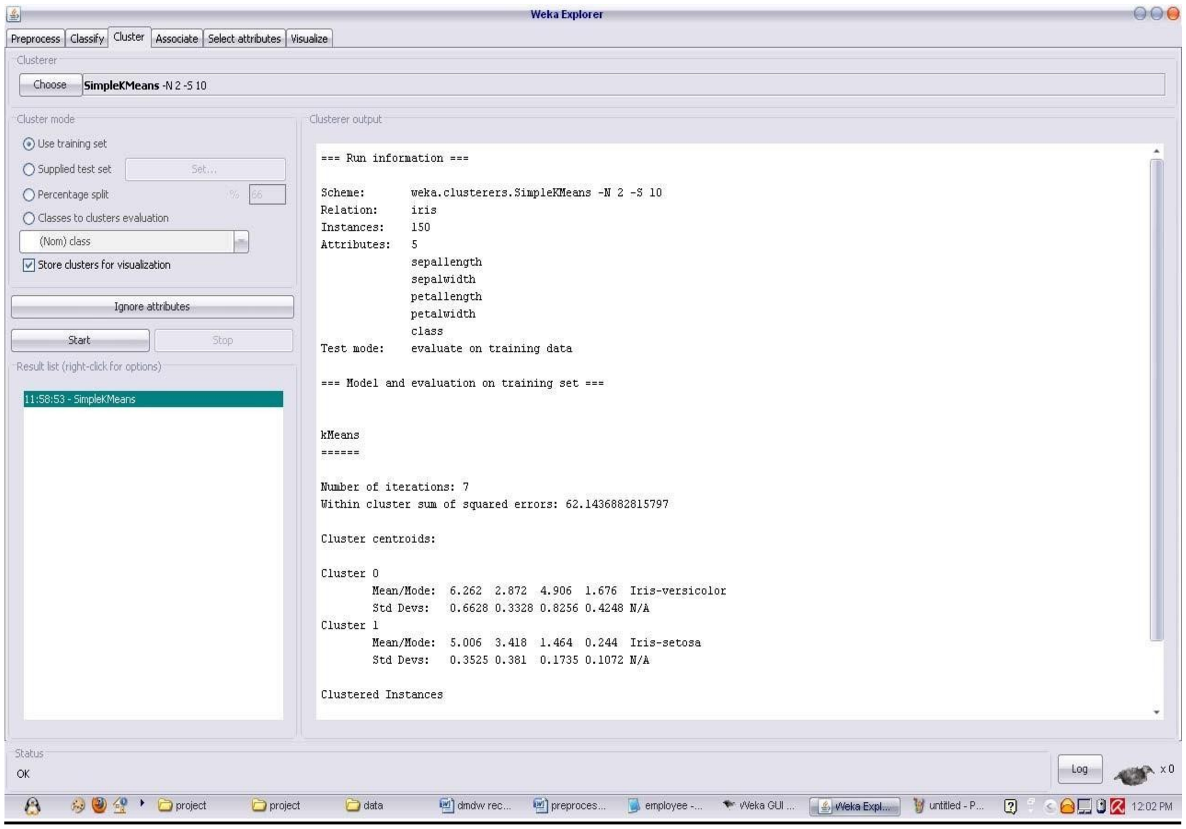
Step 4: Next click in text button to the right of the choose button to get popup window shown in the screenshots. In this window we enter six on the number of clusters and we leave the value of the seed on as it is. The seed value is used in generating a random number which is used for making the internal assignments of instances of clusters.

Step 5 : Once of the option have been specified. We run the clustering algorithm there we must make sure that they are in the 'cluster mode' panel. The use of training set option is selected and then we click 'start' button. This process and resulting window are shown in the following screenshots.

Step 6 : The result window shows the centroid of each cluster as well as statistics on the number and the percent of instances assigned to different clusters. Here clusters centroid are means vectors for each clusters. This clusters can be used to characterized the cluster.For eg, the centroid of cluster1 shows the class iris.versicolor mean value of the sepal length is 5.4706, sepal width 2.4765, petal width 1.1294, petal length 3.7941.

Step 7: Another way of understanding characterstics of each cluster through visualization ,we can do this, try right clicking the result set on the result. List panel and selecting the visualize cluster assignments.

The following screenshot shows the clustering rules that were generated when simple k means algorithm is applied on the given dataset.



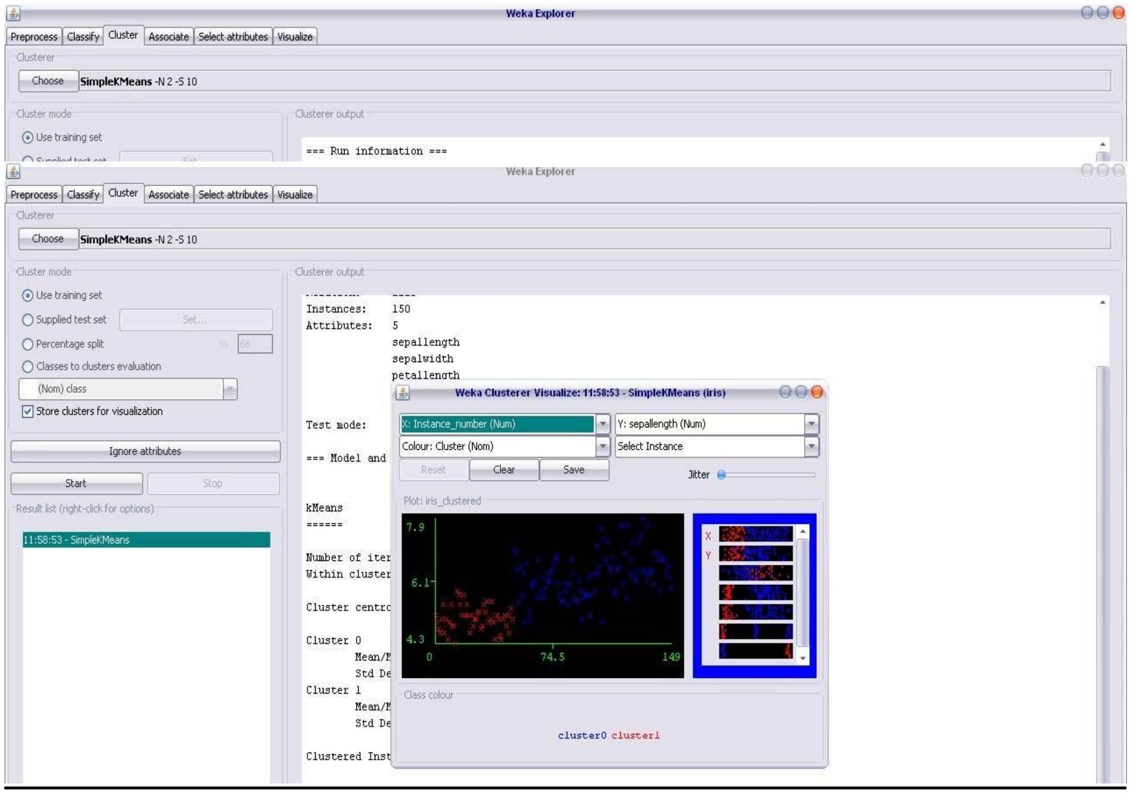
SimpleKMeans -N 2 -S 10

```
=== Run information ===

Scheme:       weka.clusterers.SimpleKMeans -N 2 -S 10
Relation:     iris
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:    evaluate on training data

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


kMeans
======

Number of iterations: 7
Within cluster sum of squared errors: 62.1436882815797

Cluster centroids:

Cluster 0
        Mean/Mode:  6.262  2.872  4.906  1.676  Iris-versicolor
        Std Devs:   0.6628 0.3328 0.8256 0.4248 N/A
Cluster 1
        Mean/Mode:  5.006  3.418  1.464  0.244  Iris-setosa
        Std Devs:   0.3525 0.381  0.1735 0.1072 N/A

Clustered Instances
```

Interpretation of the above visualization

From the above visualization, we can understand the distribution of sepal length and petal length in each cluster. For instance, for each cluster is dominated by petal length. In this case by changing the color dimension to other attributes we can see their distribution with in each of the cluster.

Step 8: We can assure that resulting dataset which included each instance along with its assign cluster. To do so we click the save button in the visualization window and save the result iris k-mean .The top portion of this file is shown in the following figure.



SimpleKMeans -N 2 -S 10

Use training set [✓]

Percentage split % 66

(Nom) class

[✓] Store clusters for visualization

11:58:53 - SimpleKMeans

X: Instance_number (Num)
Y: sepallength (Num)
Colour: Cluster (Nom)
Select Instance

cluster0 cluster1

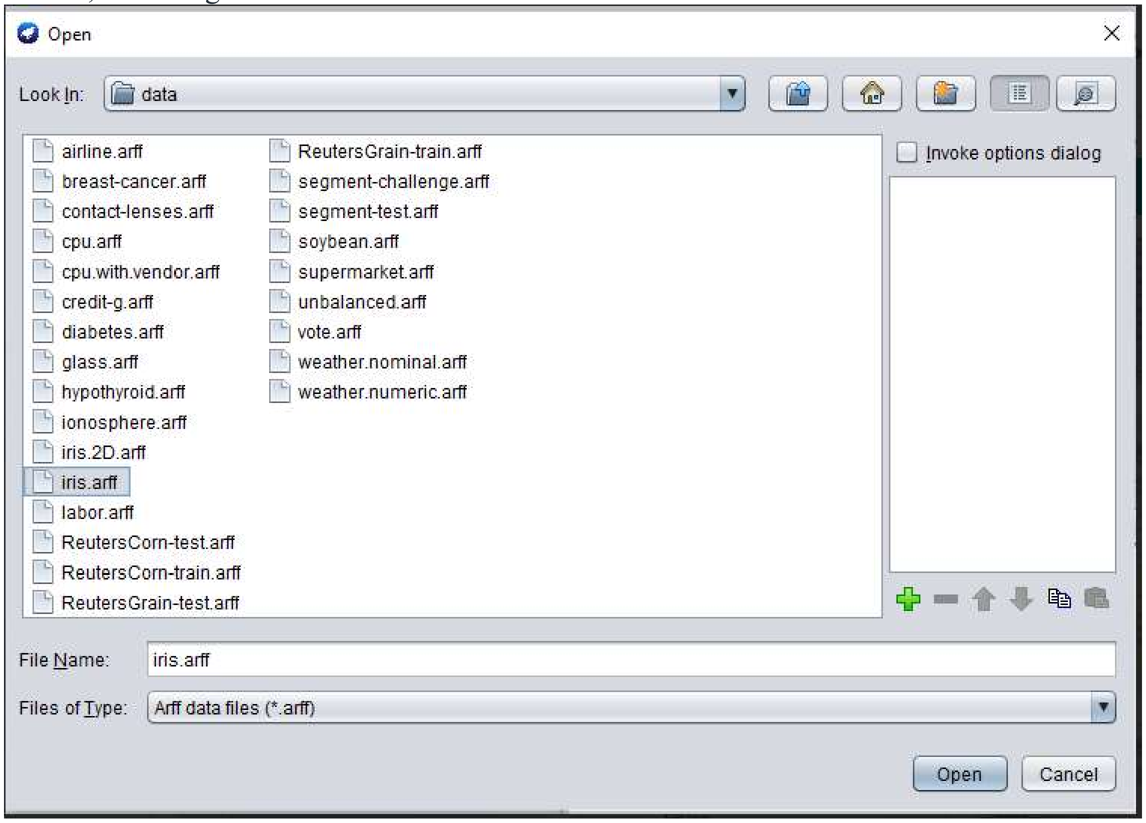
```
=== Run information ===

Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength

Test mode:

=== Model and

kMeans
======

Number of iter
Within cluster

Cluster centro

Cluster 0
        Mean/M
        Std De
Cluster 1
        Mean/M
        Std De

Clustered Inst
```

Exp.10 Demonstration of clustering process on dataset iris.arff using hierarchical clustering algorithm

Hierarchical clustering, also known as Hierarchical cluster analysis or HCA, is an unsupervised clustering approach that includes forming groups with a top-to-bottom order. For example, on our hard drive, all files and folders are organized in a hierarchy.

The program divides objects into clusters based on their similarity. The endpoint is a collection of clusters or groups, each of which is distinct from the others, yet the items inside each cluster are broadly similar.

Steps to be followed:
Step 1: Open the Weka explorer in the preprocessing interface and import the appropriate dataset; I'm using the iris.arff dataset.

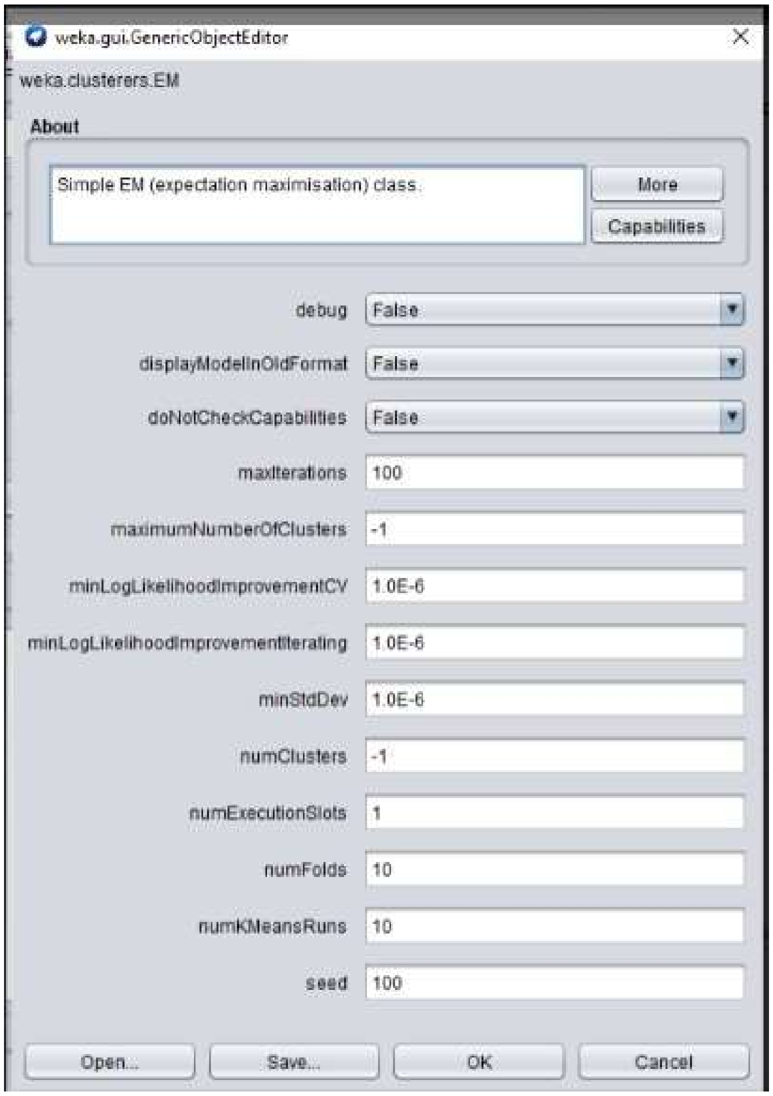


Look In: data

File Name: iris.arff

Files of Type: Arff data files (*.arff)

Step 2: To perform clustering, go to the explorer's 'cluster' tab and select the select button. As a result of this step, a dropdown list of available clustering algorithms displays; pick the Hierarchical algorithm.
Step 3: Then press the text button to the right of the pick icon to bring up the popup window seen in the screenshots. In this window, we input three for the number of clusters and leave the seed value alone. The seed value is used to generate a random number that is used to allocate cluster instances to each other internally.


weka.gui.GenericObjectEditor

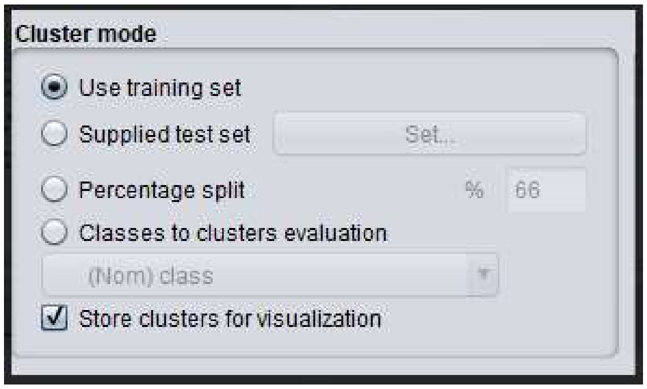
weka.clusterers.EM

About

Simple EM (expectation maximisation) class.

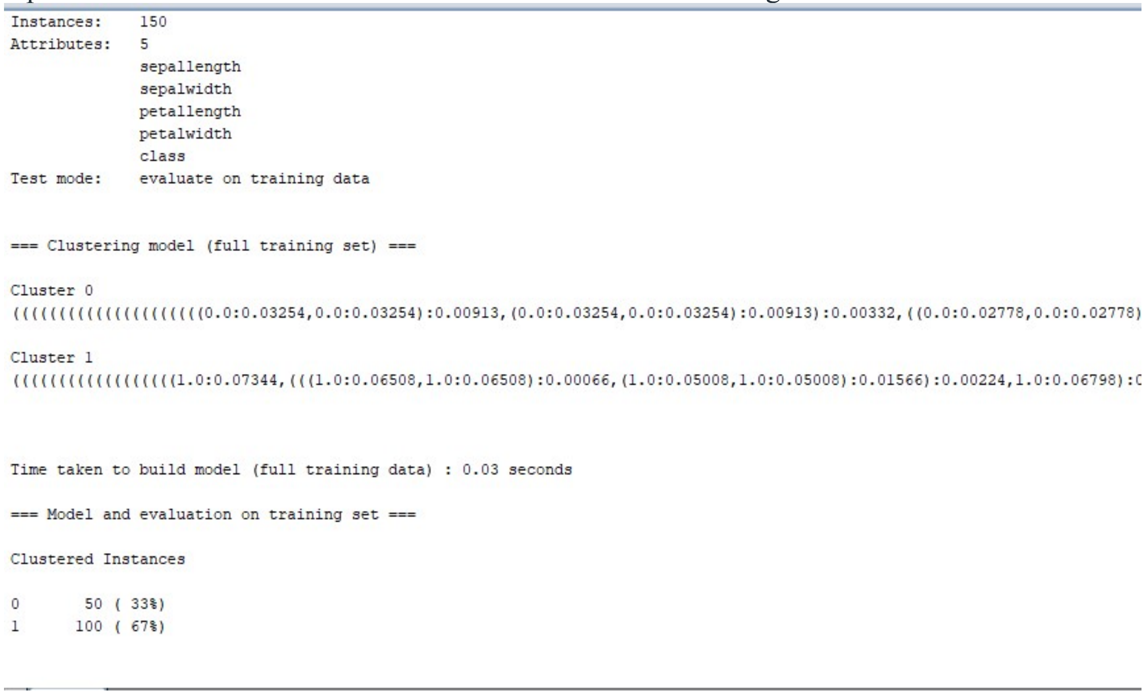
debug	False
displayModelInOldFormat	False
doNotCheckCapabilities	False
maxIterations	100
maximumNumberOfClusters	-1
minLogLikelihoodImprovementCV	1.0E-6
minLogLikelihoodImprovementIterating	1.0E-6
minStdDev	1.0E-6
numClusters	-1
numExecutionSlots	1
numFolds	10
numKMeansRuns	10
seed	100

Step 4: One of the options has been selected. Before we execute the clustering method, we need to make sure they're in the 'cluster mode' panel. The option to employ a training set is chosen, after which the 'start' button is hit. The process and the resulting window are depicted in the screenshots below.


Cluster mode

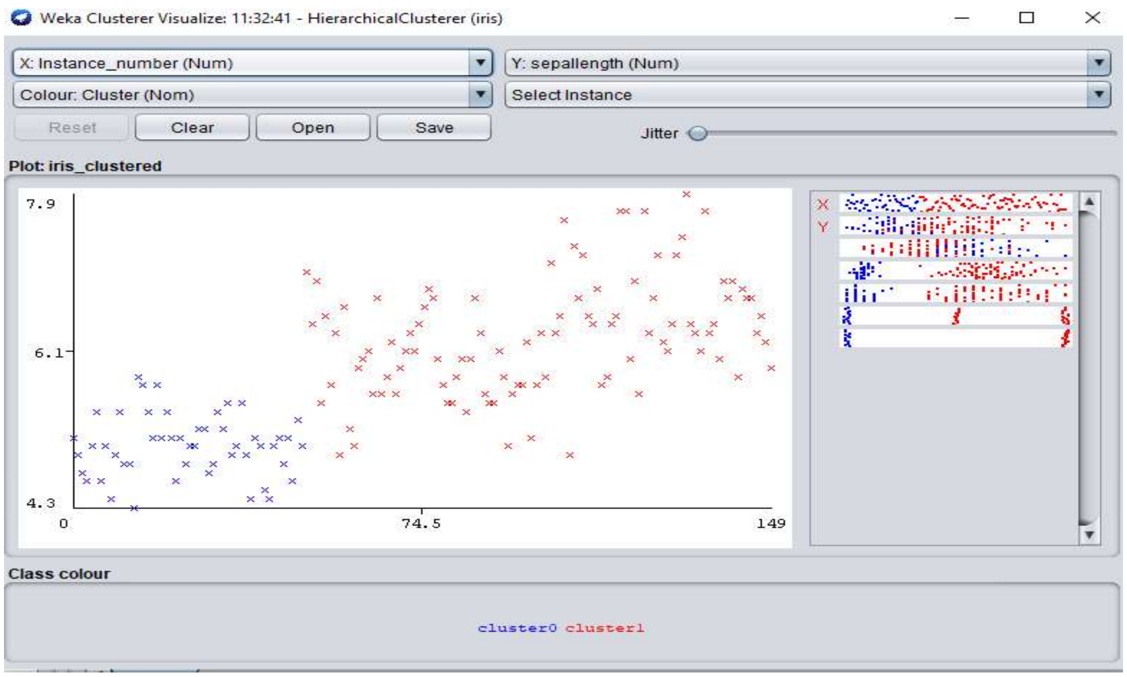
- [✓ Use training set]
- Supplied test set
- Percentage split % 66
- Classes to clusters evaluation
- (Nom) class
- [✓ Store clusters for visualization]

Step 5: The resulting window displays the centroid of each cluster, as well as data on the number and proportion of instances assigned to each cluster. A mean vector is used to represent each cluster centroid. A cluster can be described using this cluster.



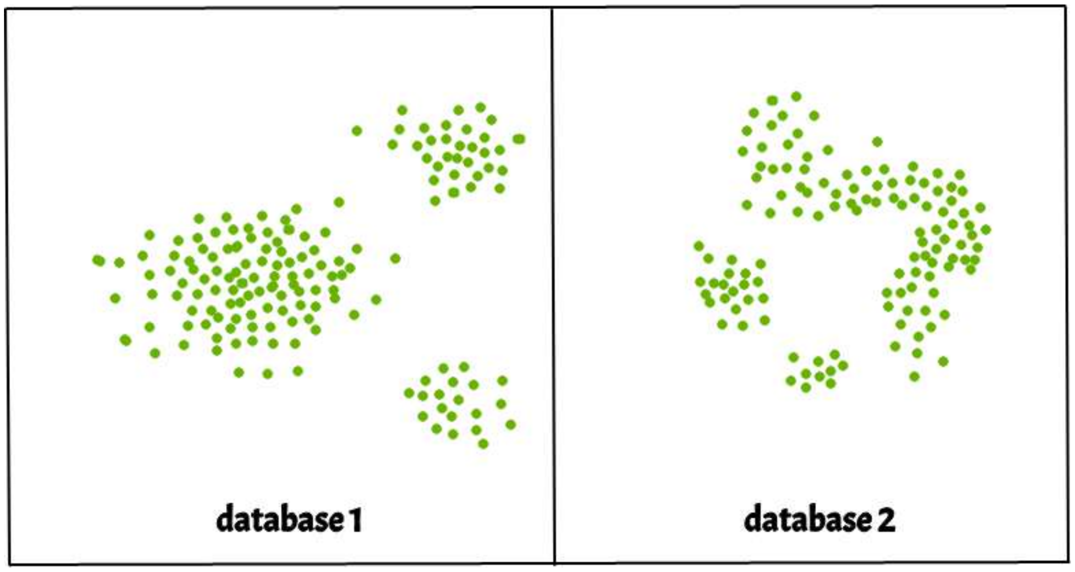
```
Instances:     150
Attributes:    5
               sepallength
               sepalwidth
               petallength
               petalwidth
               class
Test mode:     evaluate on training data


=== Clustering model (full training set) ===

Cluster 0
((((((((((((((((((0.0:0.03254,0.0:0.03254):0.00913,(0.0:0.03254,0.0:0.03254):0.00913):0.00332,((0.0:0.02778,0.0:0.02778)

Cluster 1
((((((((((((((((1.0:0.07344,(((1.0:0.06508,1.0:0.06508):0.00066,(1.0:0.05008,1.0:0.05008):0.01566):0.00224,1.0:0.06798):C



Time taken to build model (full training data) : 0.03 seconds

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

Clustered Instances

0       50 ( 33%)
1      100 ( 67%)
```

Step 6: Visualizing the qualities of each cluster is another approach to grasp them. Right-click the result set on the result to do so. To visualize cluster assignments are selected from the list column.



Exp. 11 Demonstration of clustering process using density based clustering Algorithm.

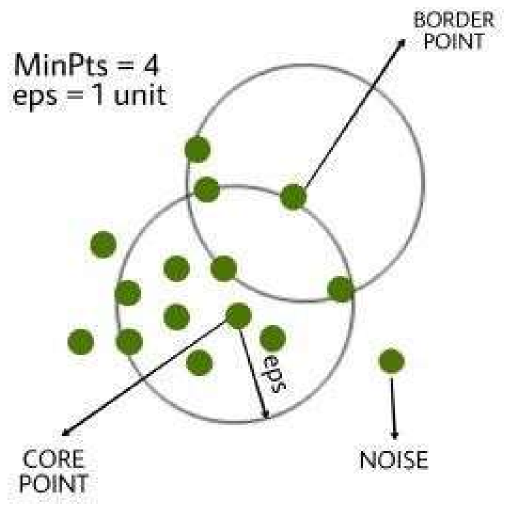
Clusters are dense regions in the data space, separated by regions of the lower density of points. The ***DBSCAN algorithm*** is based on this intuitive notion of "clusters" and "noise". The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points.



DBSCAN algorithm requires two parameters:

1. **eps** : It defines the neighborhood around a data point i.e. if the distance between two points is lower or equal to 'eps' then they are considered as neighbors. If the eps value is chosen too small then large part of the data will be considered as outliers. If it is chosen very large then the clusters will merge and majority of the data points will be in the same clusters. One way to find the eps value is based on the ***k-distance graph***.
2. **MinPts**: Minimum number of neighbors (data points) within eps radius. Larger the dataset, the larger value of MinPts must be chosen. As a general rule, the minimum MinPts can be derived from the number of dimensions D in the dataset as, MinPts >= D+1. The minimum value of MinPts must be chosen at least 3.

In this algorithm, we have 3 types of data points.
Core Point: *A point is a core point if it has more than MinPts points within eps.*
Border Point: *A point which has fewer than MinPts within eps but it is in the neighborhood of a core point.*
Noise or outlier: *A point which is not a core point or border point.*



DBSCAN algorithm can be abstracted in the following steps :

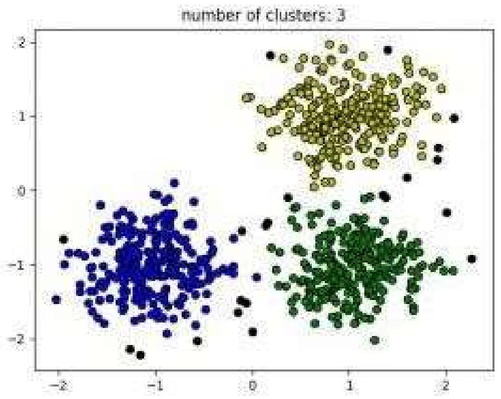
1. Find all the neighbor points within eps and identify the core points or visited with more than MinPts neighbors.
2. For each core point if it is not already assigned to a cluster, create a new cluster.
3. Find recursively all its density connected points and assign them to the same cluster as the core point.
   A point *a* and *b* are said to be density connected if there exist a point *c* which has a sufficient number of points in its neighbors and both the points *a* and *b* are within the *eps distance*. This is a chaining process. So, if *b* is neighbor of *c*, *c* is neighbor of *d*, *d* is neighbor of *e*, which in turn is neighbor of *a* implies that *b* is neighbor of *a*.
4. Iterate through the remaining unvisited points in the dataset. Those points that do not belong to any cluster are noise.

Implementation of DBSCAN algorithm in Python :

```
import matplotlib.pyplot as plt

import numpy as np

from sklearn.cluster import DBSCAN

from sklearn import metrics

from sklearn.datasets.samples_generator import make_blobs

from sklearn.preprocessing import StandardScaler

from sklearn import datasets


# Load data in X

X, y_true = make_blobs(n_samples=300, centers=4,
                                        cluster_std=0.50, random_state=0)
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

print(labels)

# Plot result

# Black removed and is used for noise instead.
unique_labels = set(labels)
colors = ['y', 'b', 'g', 'r']
print(colors)
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = 'k'

    class_member_mask = (labels == k)

    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=col,
                markeredgecolor='k',
                markersize=6)

    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=col,
                markeredgecolor='k',
                markersize=6)
```

```
plt.title('number of clusters: %d' % n_clusters_)

plt.show()
```

Output:



12. **Implement the following data mining algorithms using python**

a) **Decision Tree**

b) **Naïve bayes**

c) **K-Nearest Neighbor**

d) **K-Means**

a) Decision Tree

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Function importing Dataset
def importdata():
	balance_data = pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-'+
'databases/balance-scale/balance-scale.data',
	sep= ',', header = None)

	# Printing the dataswet shape
	print ("Dataset Length: ", len(balance_data))
	print ("Dataset Shape: ", balance_data.shape)

	# Printing the dataset obseravtions
	print ("Dataset: ",balance_data.head())
	return balance_data

# Function to split the dataset
def splitdataset(balance_data):

	# Separating the target variable
	X = balance_data.values[:, 1:5]
	Y = balance_data.values[:, 0]

	# Splitting the dataset into train and test
	X_train, X_test, y_train, y_test = train_test_split(
	X, Y, test_size = 0.3, random_state = 100)

	return X, Y, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):

	# Creating the classifier object
```

```
    clf_gini = DecisionTreeClassifier(criterion = "gini",
            random_state = 100,max_depth=3, min_samples_leaf=5)

    # Performing training
    clf_gini.fit(X_train, y_train)
    return clf_gini

# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):

    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
            criterion = "entropy", random_state = 100,
            max_depth = 3, min_samples_leaf = 5)

    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy


# Function to make predictions
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

    print("Confusion Matrix: ",
        confusion_matrix(y_test, y_pred))

    print ("Accuracy : ",
    accuracy_score(y_test,y_pred)*100)

    print("Report : ",
    classification_report(y_test, y_pred))

# Driver code
def main():

    # Building Phase
    data = importdata()
```

```
        X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
        clf_gini = train_using_gini(X_train, X_test, y_train)
        clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

        # Operational  Phase
        print("Results Using Gini Index:")

        # Prediction using gini
        y_pred_gini = prediction(X_test, clf_gini)
        cal_accuracy(y_test, y_pred_gini)

        print("Results Using Entropy:")
        # Prediction using entropy
        y_pred_entropy = prediction(X_test, clf_entropy)
        cal_accuracy(y_test, y_pred_entropy)


# Calling main function
if __name__=="_main__":
        main()
```

OUTPUT:

```
                                Data Infomation:

Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
                           Results Using Gini Index:

Predicted values:
['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L'
'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L'
'L' 'L'
 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R'
'L' 'R'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L'
'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L'
'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R'
'R' 'R'
```

```
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L'
'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R'
'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R'
'R' 'R'
 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
'R' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix:  [[ 0  6  7]
                    [ 0 67 18]
                    [ 0 19 71]]
Accuracy : 73.4042553191
Report :
       precision    recall  f1-score   support
  B       0.00      0.00      0.00        13
  L       0.73      0.79      0.76        85
  R       0.74      0.79      0.76        90
avg/total 0.68      0.73      0.71       188
```

Results Using Entropy:

```
Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L'
'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L'
'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R'
'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L'
'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R'
'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R'
'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R'
'L' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R'
'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
'L' 'R'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix:  [[ 0  6  7]
                    [ 0 63 22]
                    [ 0 20 70]]
```

```
Accuracy : 70.7446808511
Report :
          precision    recall  f1-score   support
    B       0.00      0.00      0.00        13
    L       0.71      0.74      0.72        85
    R       0.71      0.78      0.74        90
avg / total 0.66      0.71      0.68       188
```

b) Naïve bayes

```
import pandas as pd
import numpy as np
from sklearn import datasets
iris = datasets.load_iris() # importing the dataset
iris.data # showing the iris data


X=iris.data #assign the data to the X
y=iris.target #assign the target/flower type to the y

print (X.shape)
print (y.shape)

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=9) #Split the
dataset

from sklearn.naive_bayes import GaussianNB
nv = GaussianNB() # create a classifier
nv.fit(X_train,y_train) # fitting the data

from sklearn.metrics import accuracy_score
y_pred = nv.predict(X_test) # store the prediction data
accuracy_score(y_test,y_pred) # calculate the accuracy
```

Output:

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
      .......
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

```
(150, 4)
(150,)
```

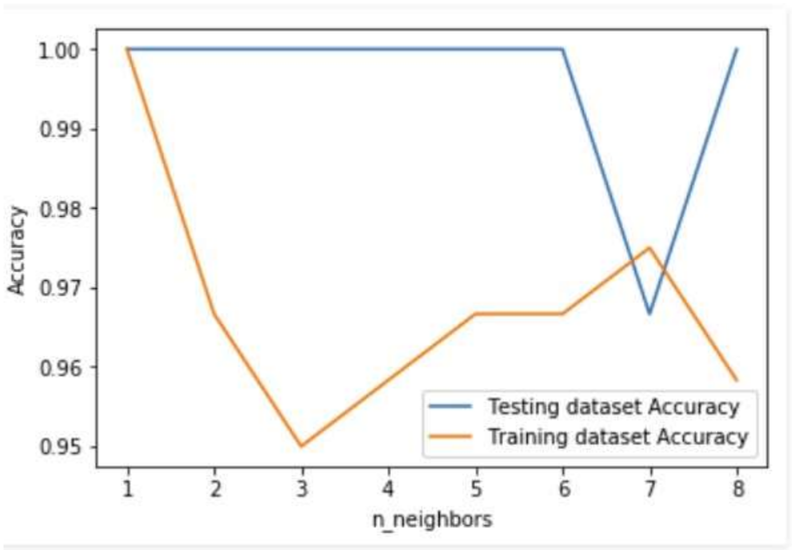
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
1.0
```

c) K-Nearest Neighbor

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt

irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
                             X, y, test_size = 0.2, random_state=42)

neighbors = np.arange(1,  9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop over K values
for i, k in enumerate(neighbors):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        # Compute training and test data accuracy
        train_accuracy[i] = knn.score(X_train, y_train)
        test_accuracy[i] = knn.score(X_test, y_test)

# Generate plot
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```

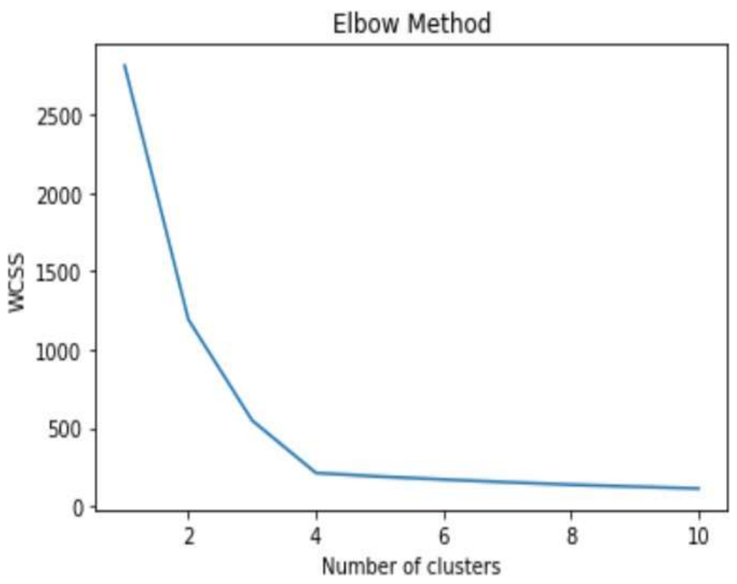
Output:

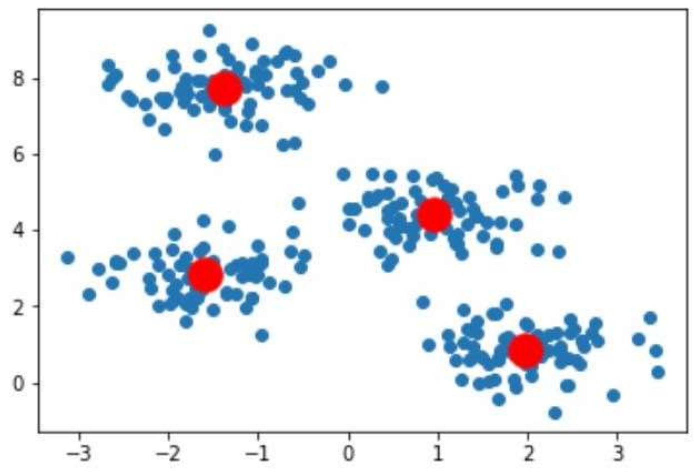


d) K-Means

```
import numpy as np

import pandas as pd

from matplotlib import pyplot as plt

from sklearn.datasets.samples_generator import make_blobs

from sklearn.cluster import KMeans


X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

plt.scatter(X[:,0], X[:,1])
```

```
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10,
random_state=0)
pred_y = kmeans.fit_predict(X)
plt.scatter(X[:,0], X[:,1])
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')
plt.show()
```




8
6
4
2
0
−3
−2
−1
0
1
2
3