1. To Create Data in .arff format

Input File:  bank_data.csv

Procedure:

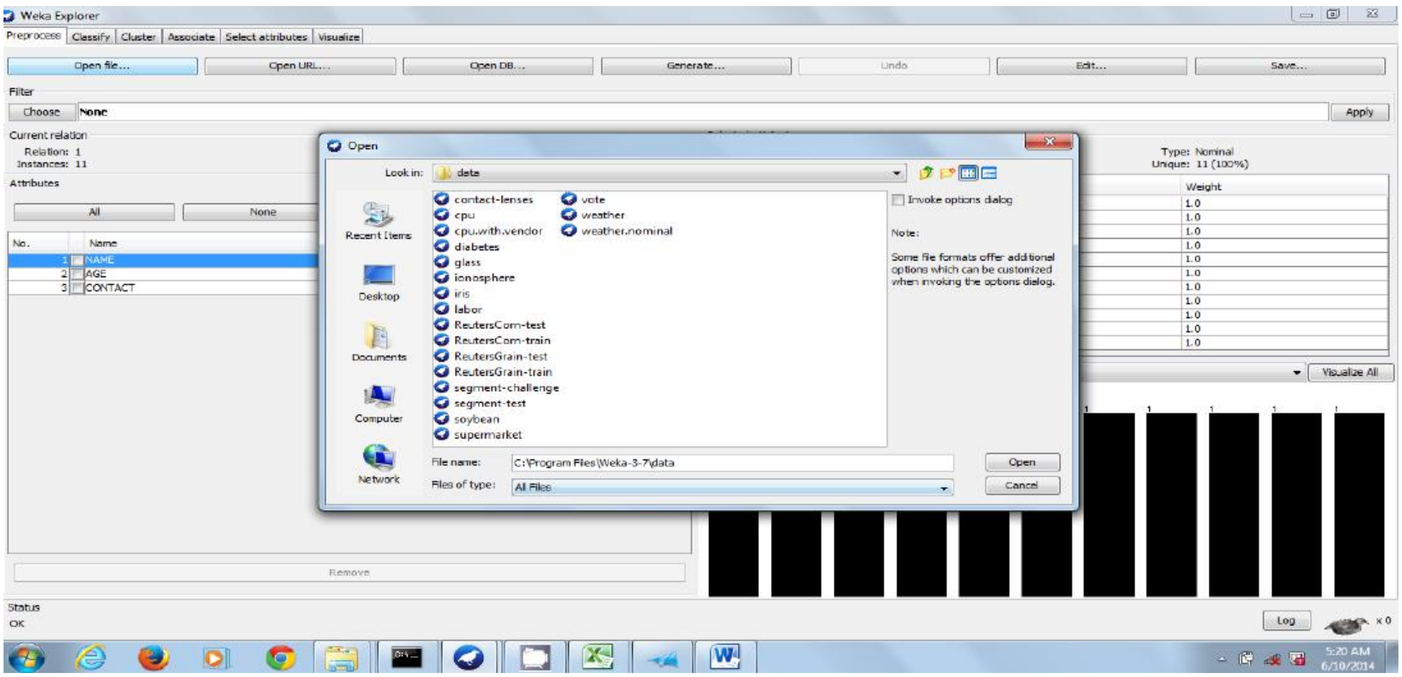
Open MS Excel. Create a new worksheet with respective headings and data.

Save the file with .csv extension

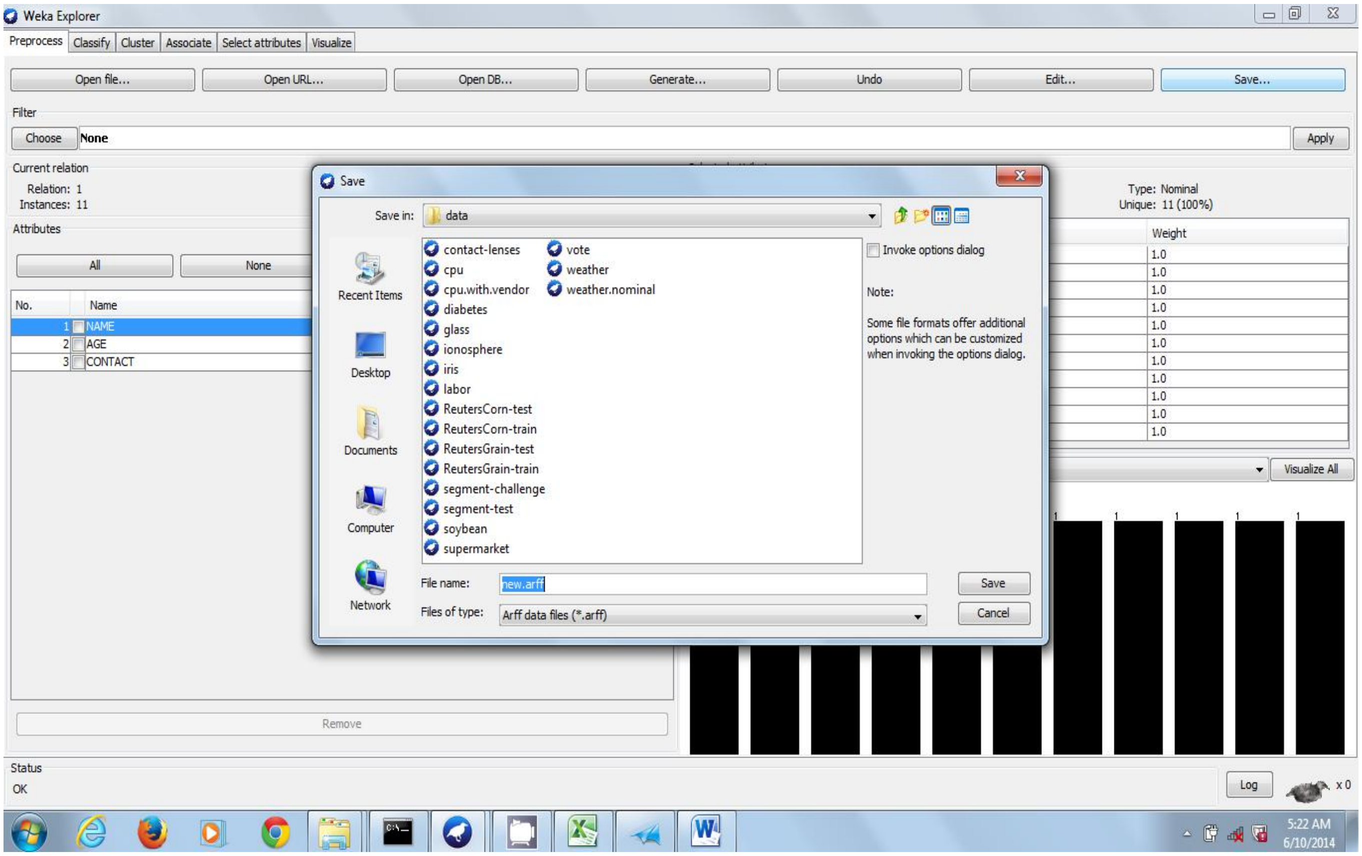
Open Preprocessor tab in explorer

Click open the file button and browse the file to open.

Load the desired .csv file using open File tab.




Click **SAVE** shown… dialog box opens save with extension as **.arff**


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
None
Apply
Current relation
Relation: 1
Instances: 11
Attributes
All
None
No.
Name
1 NAME
2 AGE
3 CONTACT
Remove
Type: Nominal
Unique: 11 (100%)
Weight
Visualize All
Save
Save in: data
contact-lenses
cpu
cpu.with.vendor
diabetes
glass
ionosphere
iris
labor
ReutersCorn-test
ReutersCorn-train
ReutersGrain-test
ReutersGrain-train
segment-challenge
segment-test
soybean
supermarket
vote
weather
weather.nominal
Invoke options dialog
Note:
Some file formats offer additional options which can be customized when invoking the options dialog.
Recent Items
Desktop
Documents
Computer
Network
File name: new.arff
Files of type: Arff data files (*.arff)
Save
Cancel
Status
OK
Log
x 0
5:22 AM
6/10/2014

2. Create data in .csv format and store it in .arff format

**Input:** bank_data

Procedure:

Open notepad and type the arff header information.

Add data with respect to the given field separated by commas

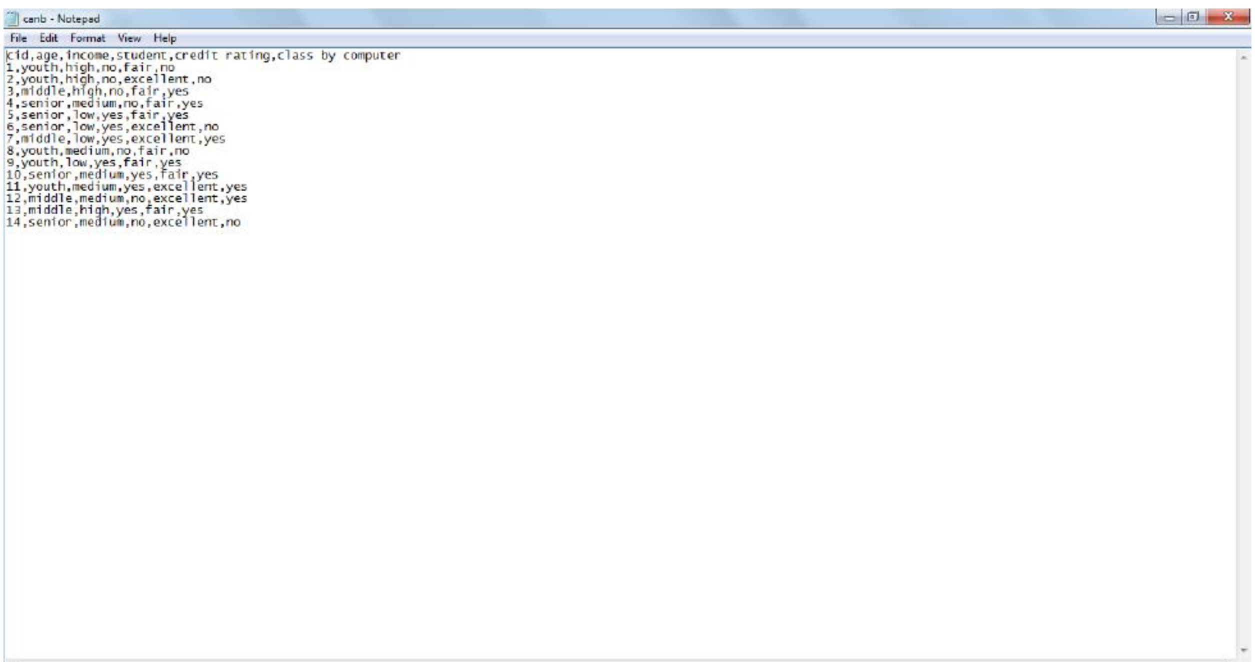
Save file as bank_data with .arff extension

Open preprocessor tab of weka in the explorer

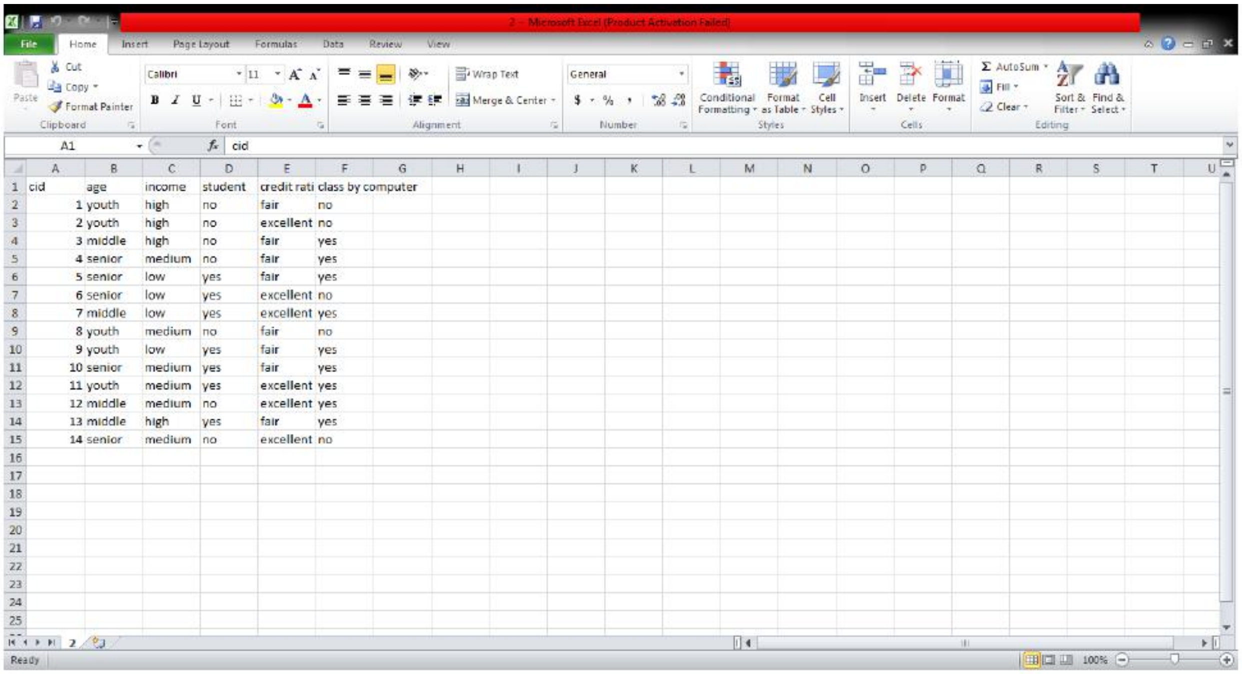
Click open the button and browse the file bank_data.arff

If the data has been entered without any errors then the file details will be available on the preprocessor screen.

**Insert data fields in note pad like shown and save with extension <u>.csv</u> choosing <u>all files</u> of type:**



**An csv file looks like**



| cid | age | income | student | credit rati | class by computer |
|---|---|---|---|---|---|
| 1 | youth | high | no | fair | no |
| 2 | youth | high | no | excellent | no |
| 3 | middle | high | no | fair | yes |
| 4 | senior | medium | no | fair | yes |
| 5 | senior | low | yes | fair | yes |
| 6 | senior | low | yes | excellent | no |
| 7 | middle | low | yes | excellent | yes |
| 8 | youth | medium | no | fair | no |
| 9 | youth | low | yes | fair | yes |
| 10 | senior | medium | yes | fair | yes |
| 11 | youth | medium | yes | excellent | yes |
| 12 | middle | medium | no | excellent | yes |
| 13 | middle | high | yes | fair | yes |
| 14 | senior | medium | no | excellent | no |

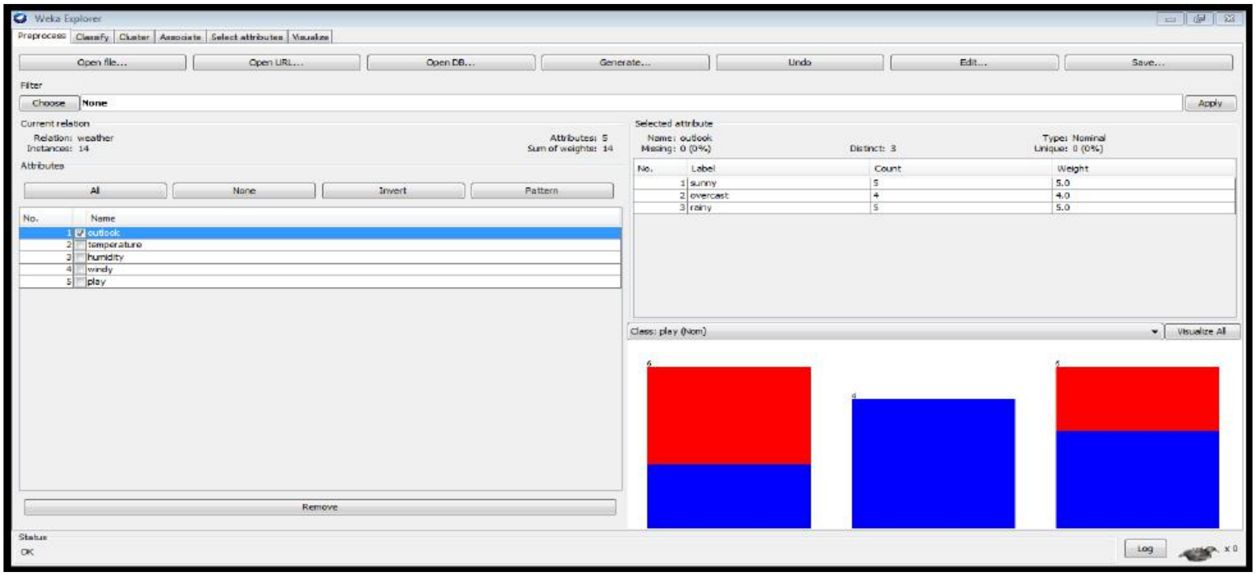
**Output:** Data are successfully uploaded.

3. Dimensionality Reduction or Attribute Removal

Input: weather.arff

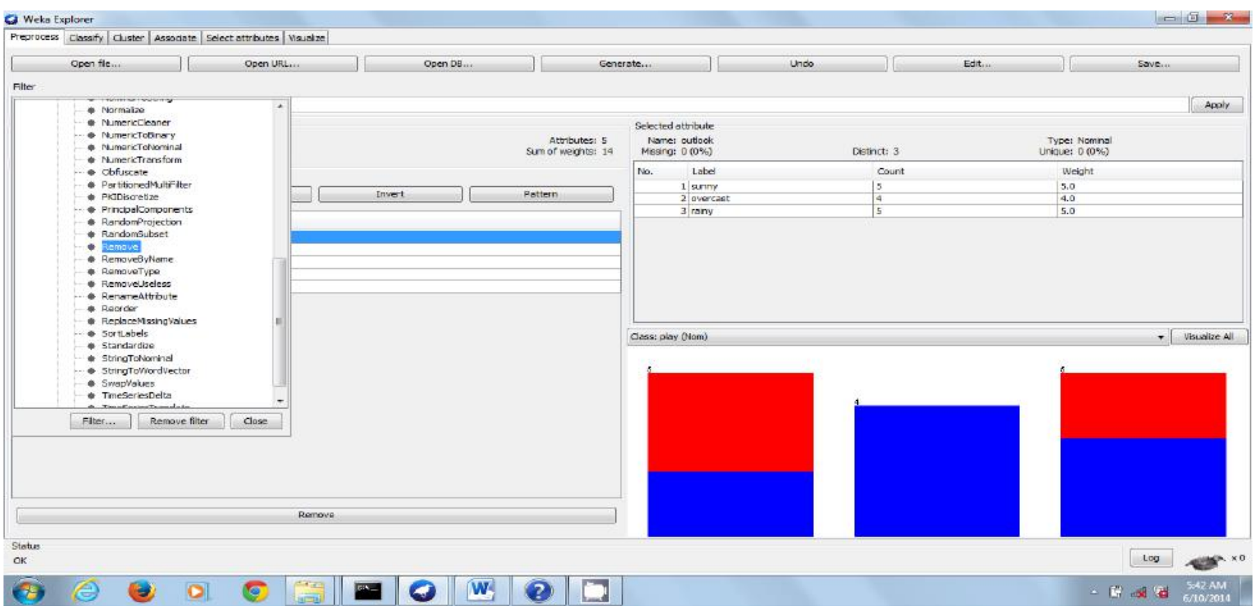
Procedure:

We can directly remove the attribute by selecting the attribute and click REMOVE button as shown below.

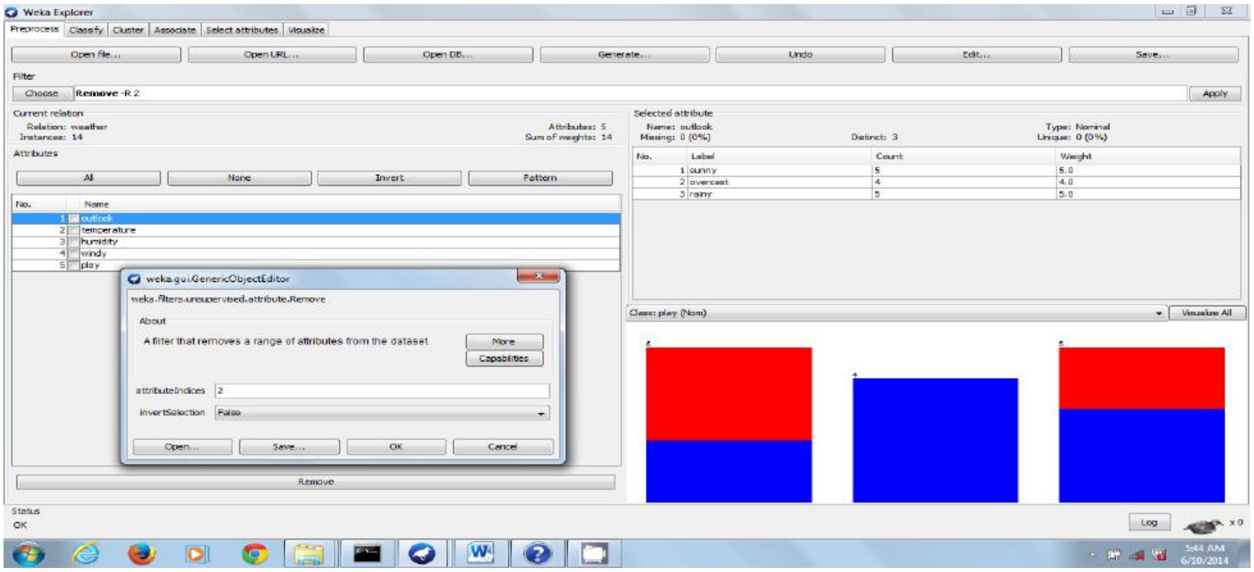



(Or)

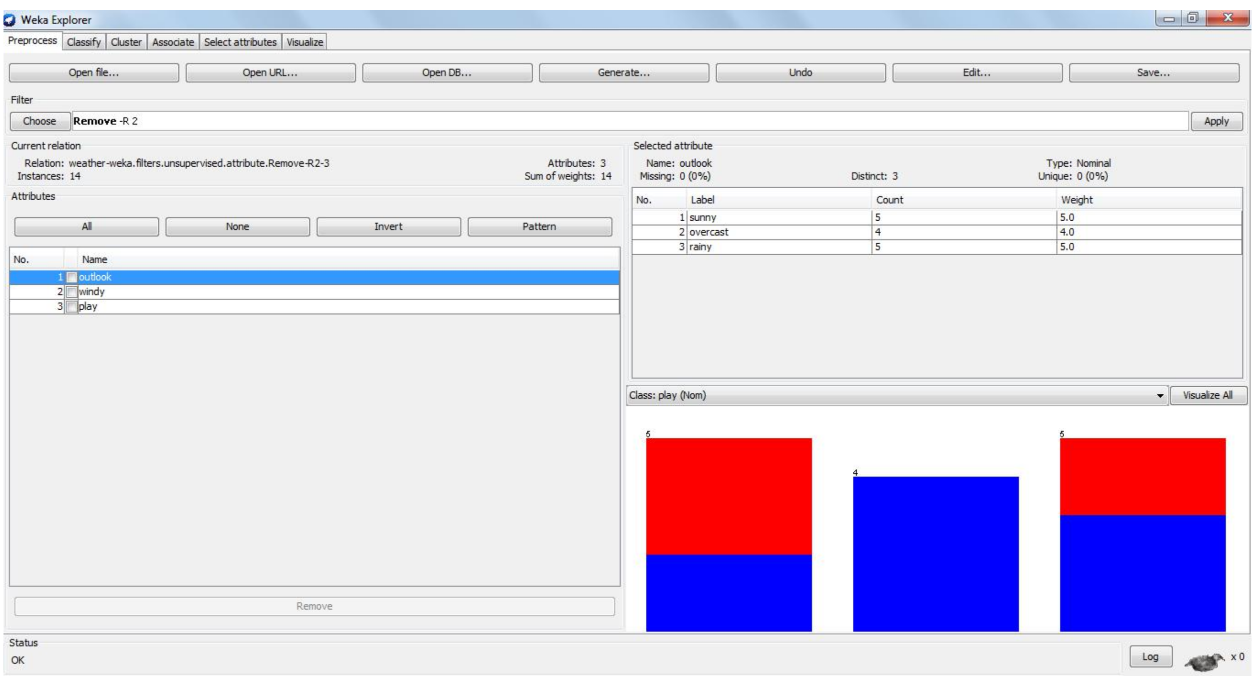
We can choose REMOVE from FILTER tab as shown below.



Even we can specify the attribute indices by right clicking over the filter box, this opens dialog box as:




Finally the attribute list after mining with remove filter.




Click on the save button and store the modified dataset with a new name as weather.arff

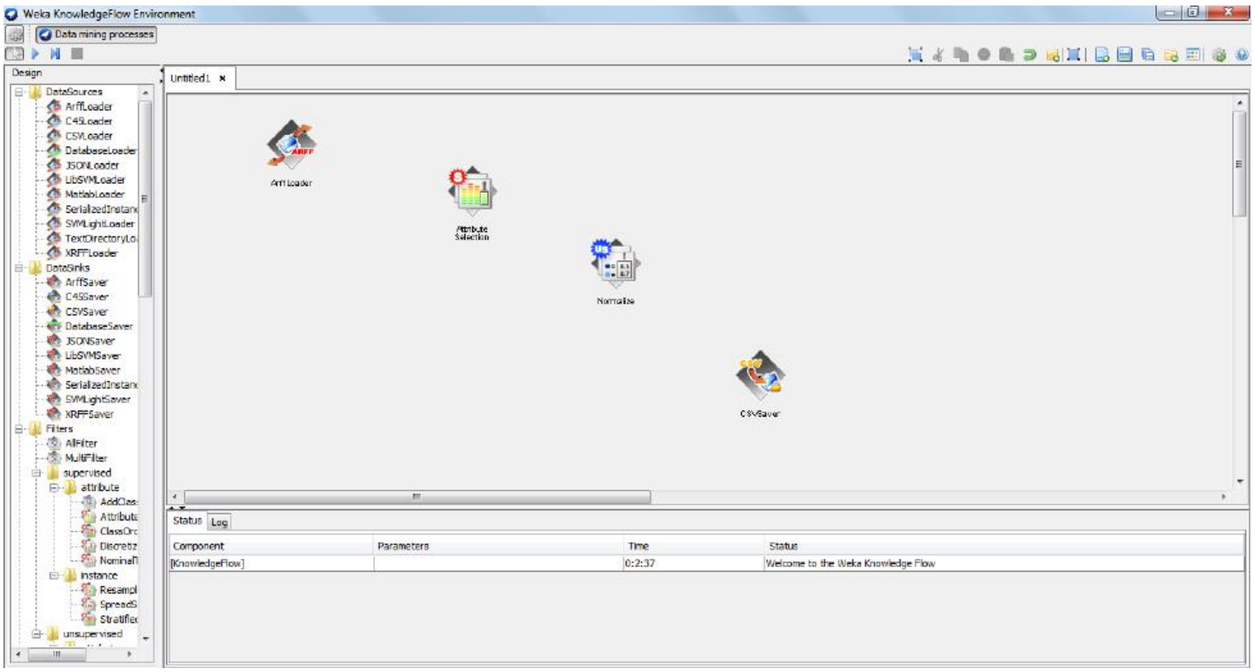
Output: weather1.arff has a new list of attribute after removal of an attribute

4. Data Normalization

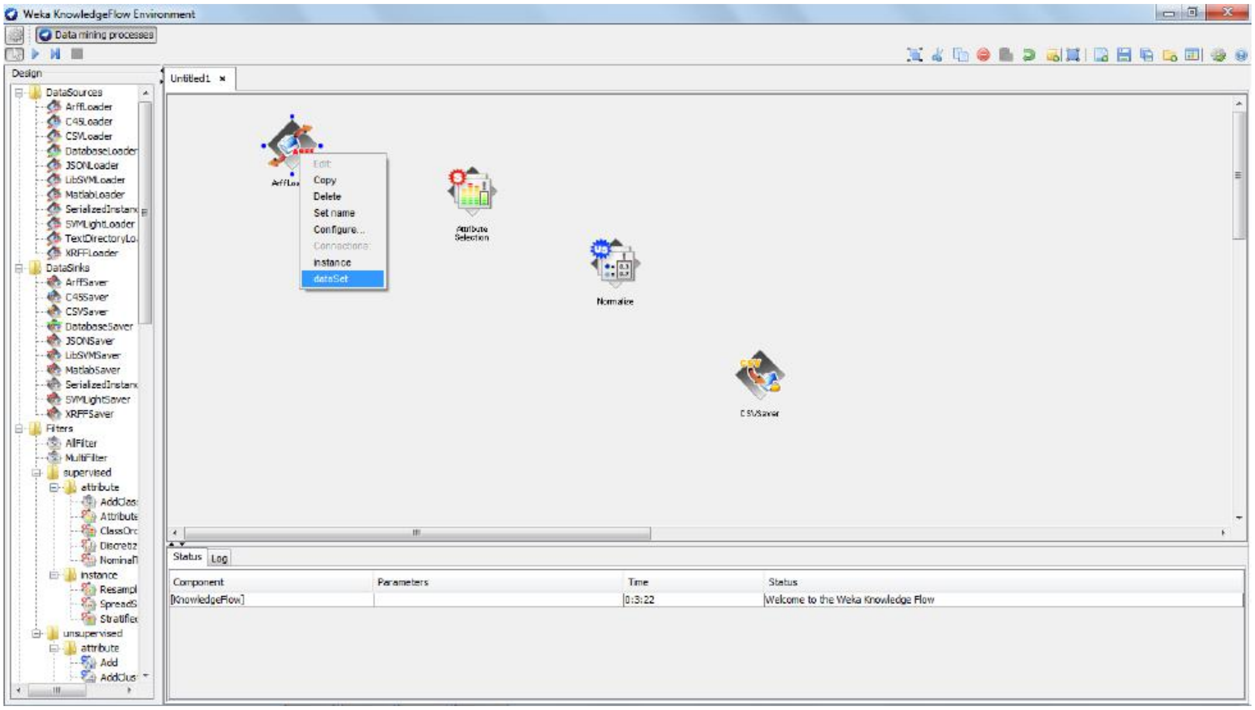
Components Used:

| S.No. | Name of the Icon | Tab | Purpose |
|---|---|---|---|
| 1 | ArffLoader | Datasources | To choose a dataset of arff |
| 2 | AttributeSelection | Filters | To select attributes using EvaluatorSearch Method |
| 3 | Normalization | Filters | To make the numerical dataset values exist between boundaries of 0 and 1 |
| 4 | CSVsaver | DataSinks | DTo make output appear in a separate .csv file format |

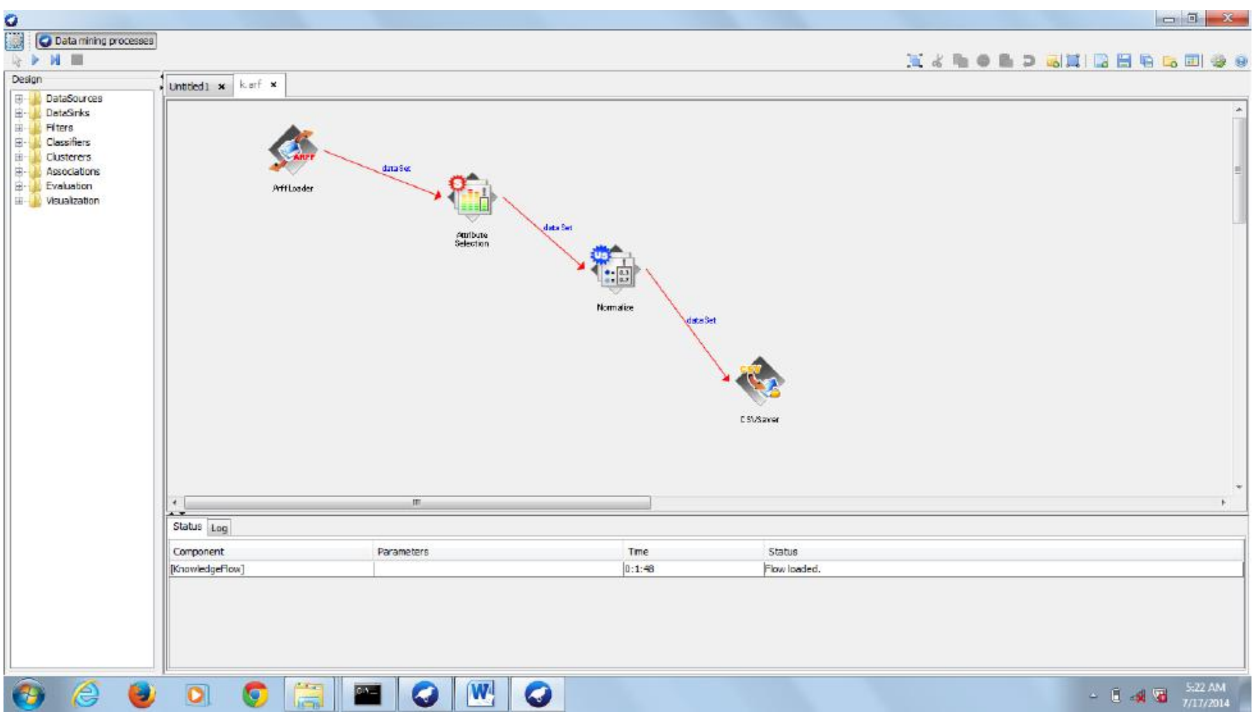
**Procedure:** Arranging the icons according to above give components



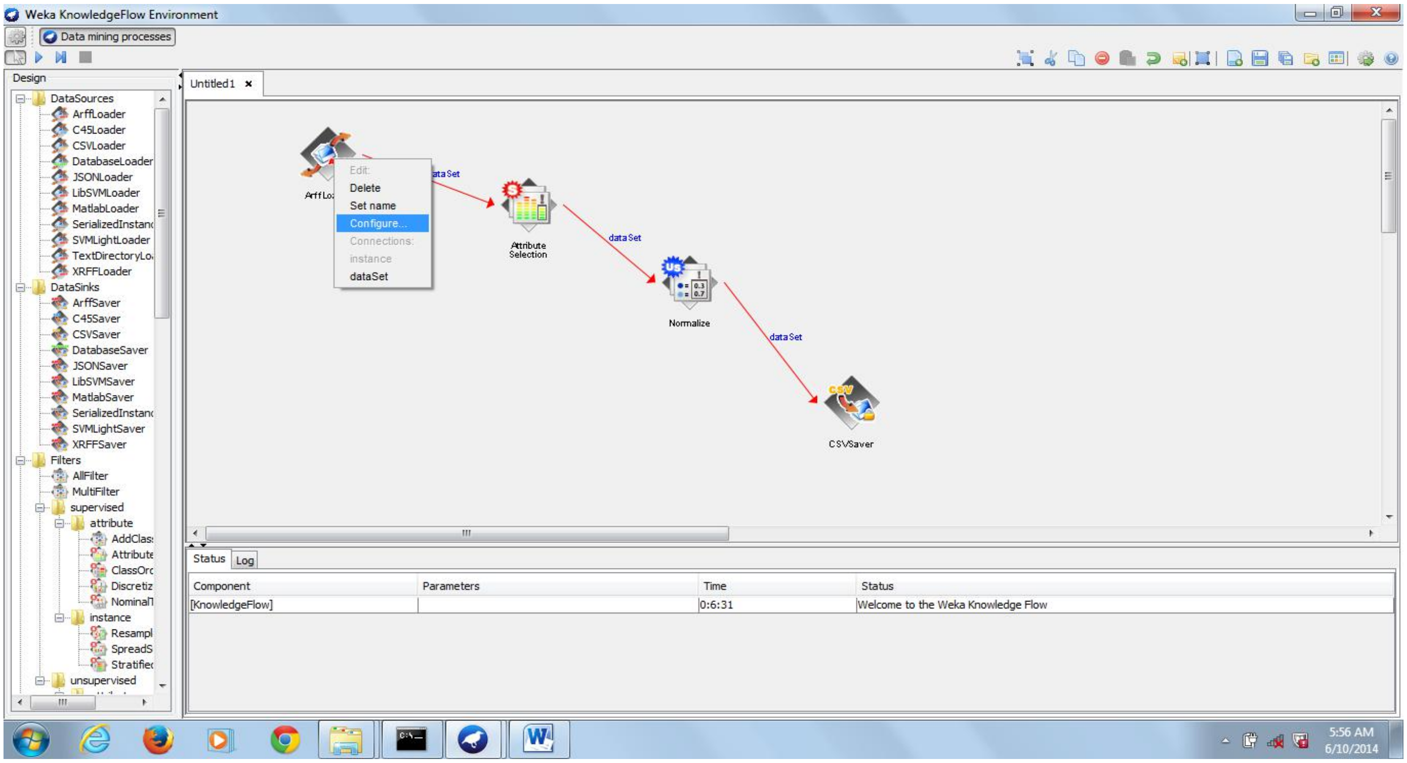

Link the icons using dataset



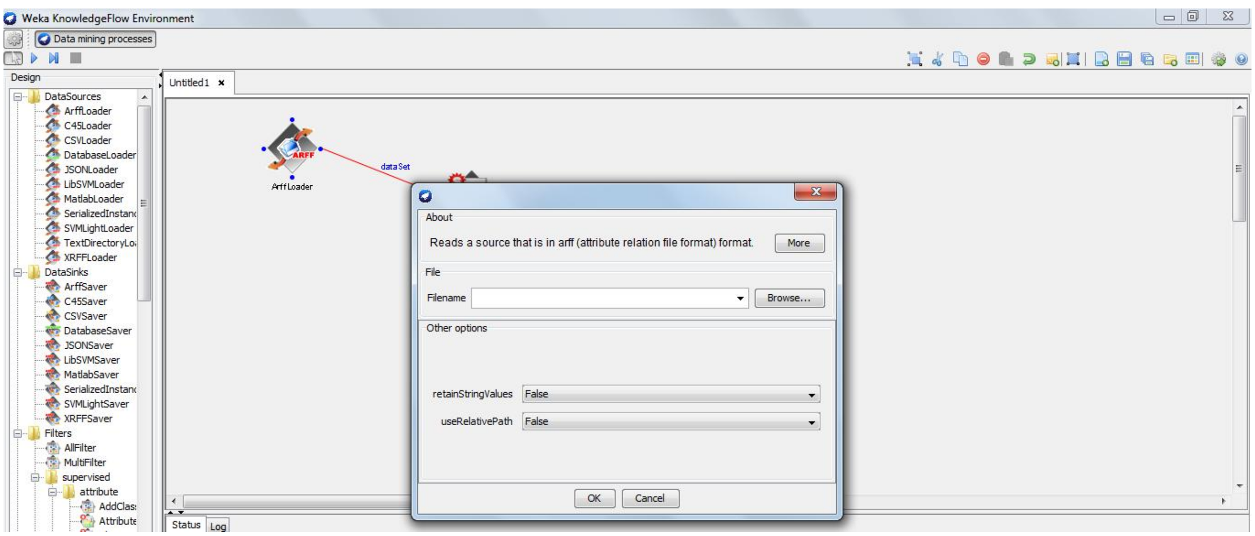

After linking all the icon the window shows below



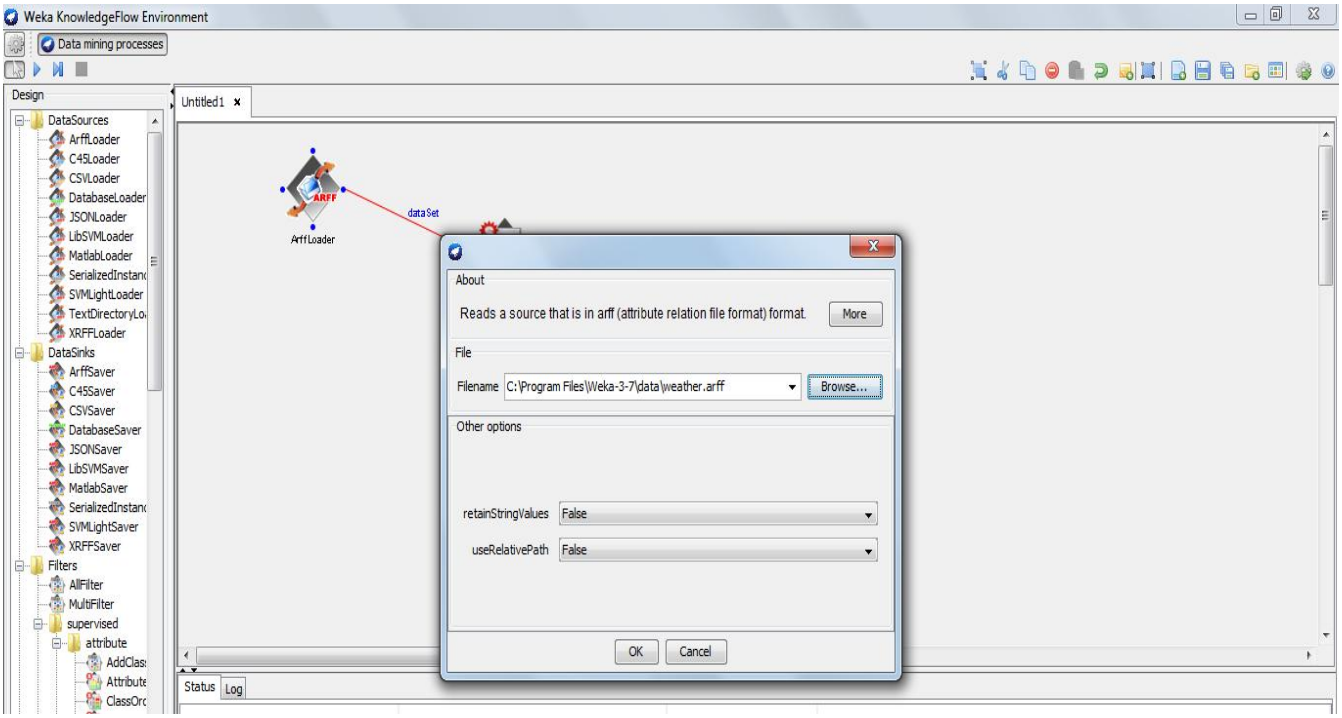

Load the file by right click over **arffloader**in configure tab.



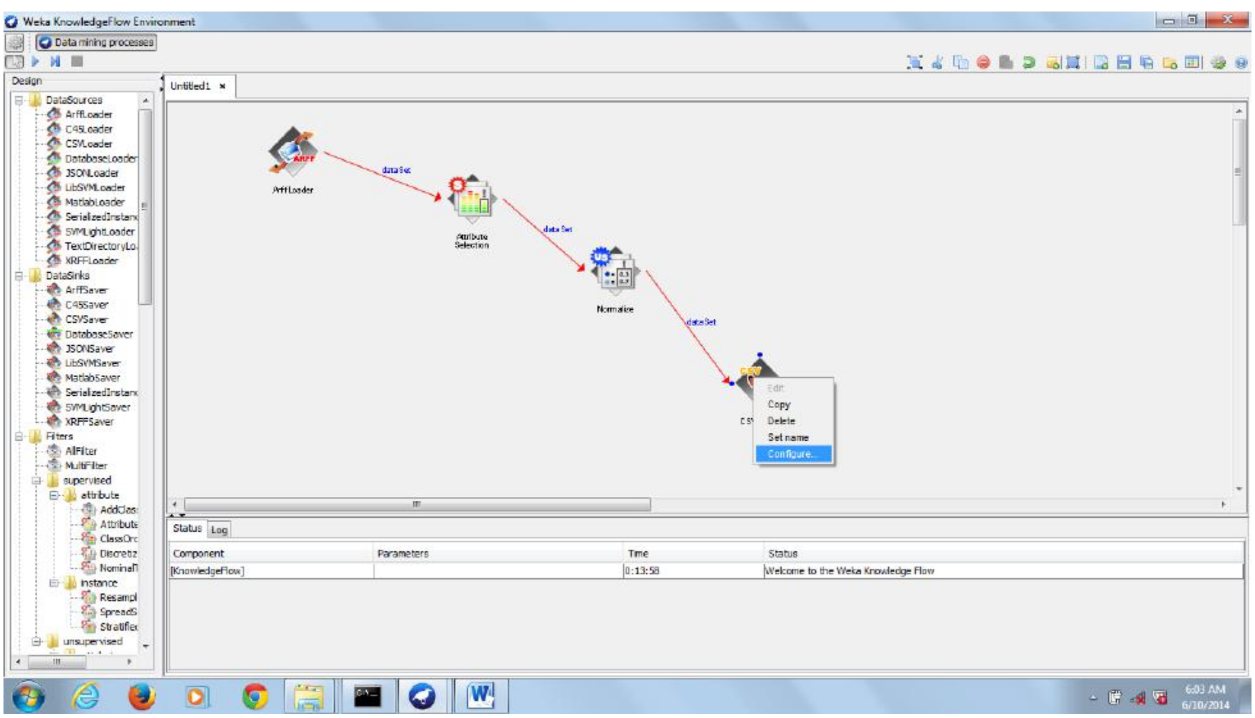
After selecting the configure tab a new window is opened in that we have to load aarff file.



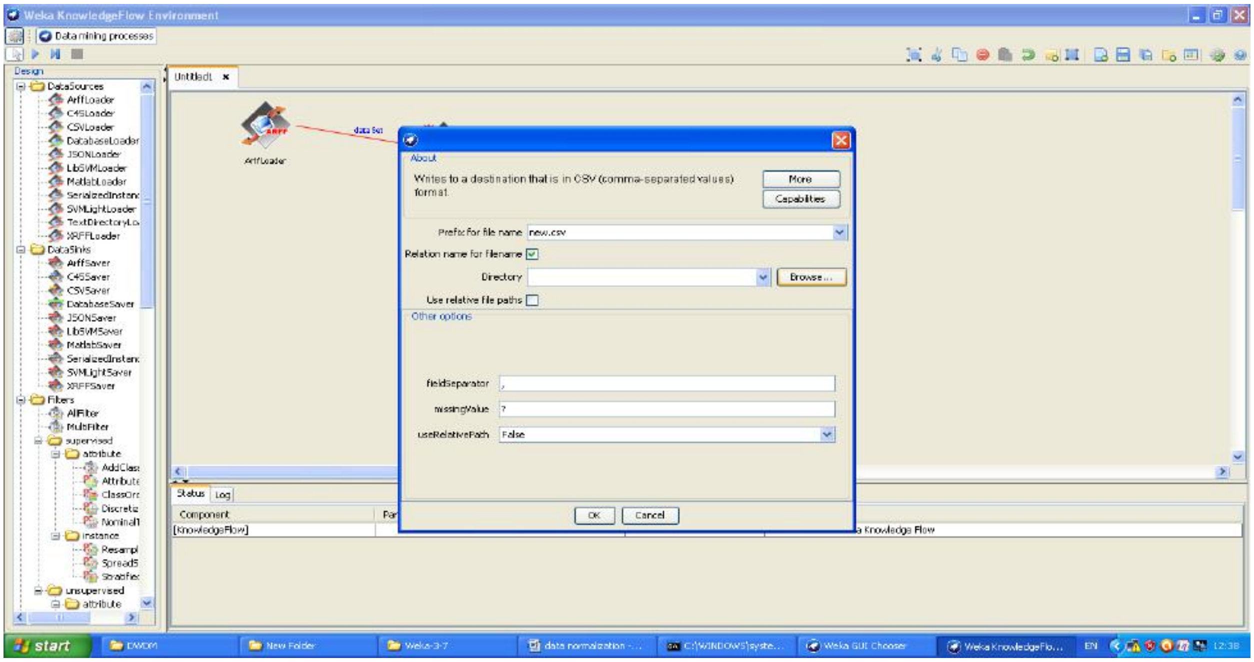
After browse the arff file from data and click on ok.



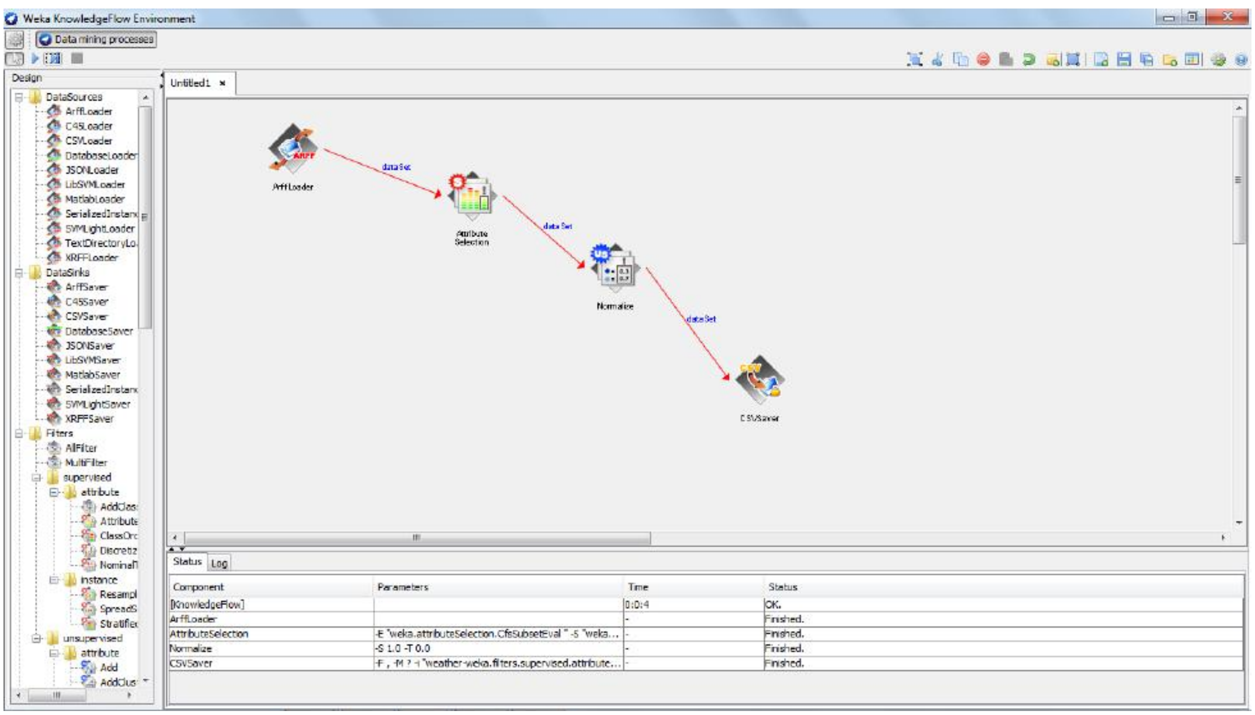
Load the file by right click over **csvsaver** in configure tab.



Give the file name as **.csv** and select the directory.

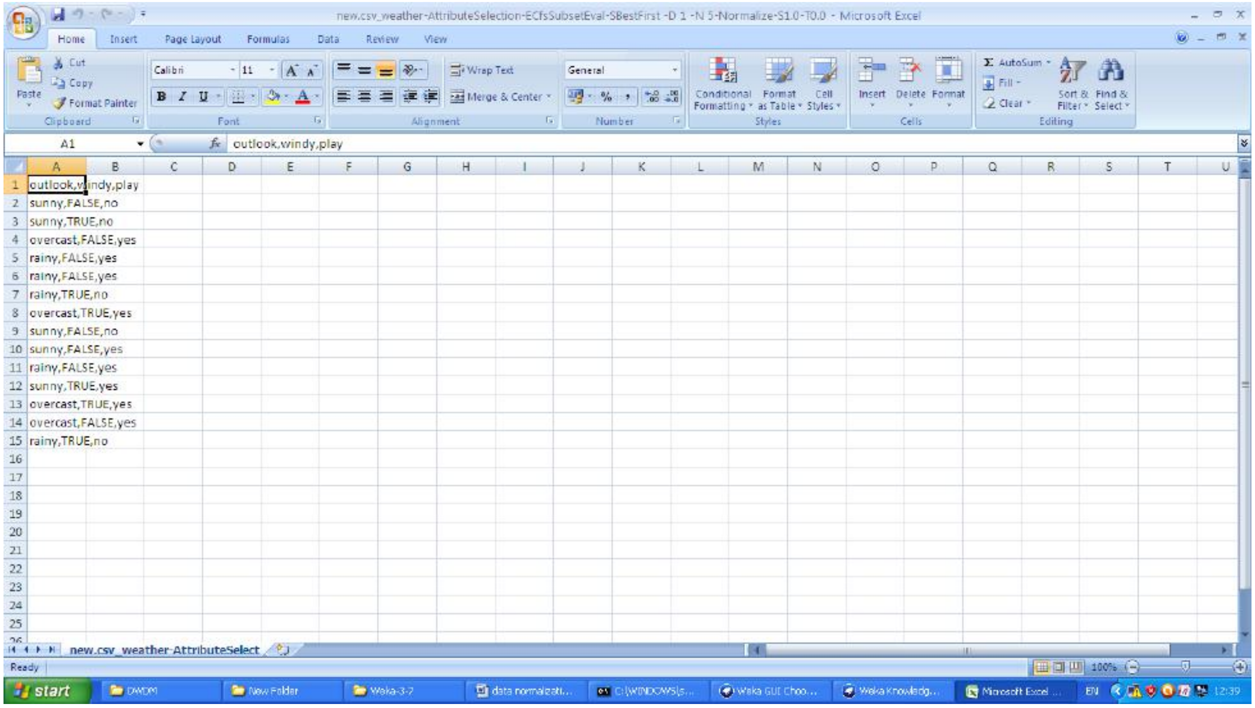


Run the file.



**Output:**

So result can viewed as double over the resultant csv file that opens in MS-EXCEL WORK SHEET.

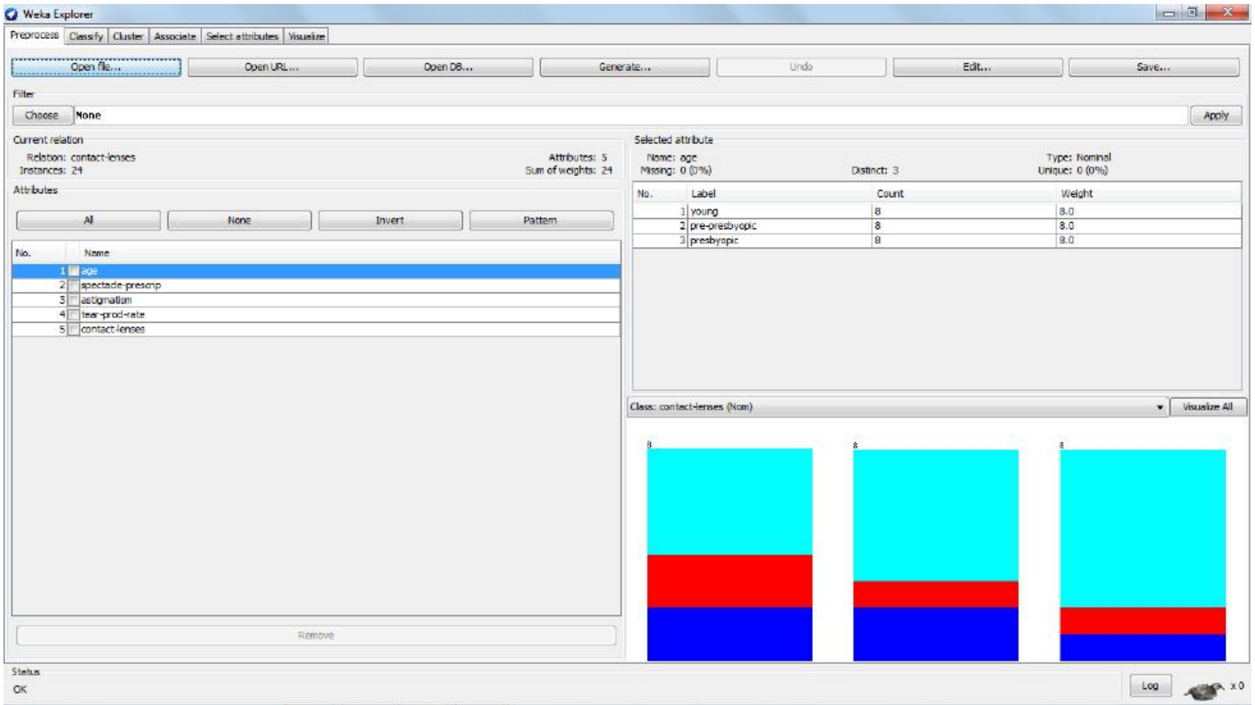

new.csv_weather-AttributeSelection-ECfsSubsetEval-SBestFirst -D 1 -N 5-Normalize-S1.0-T0.0 - Microsoft Excel
outlook,windy,play
sunny,FALSE,no
sunny,TRUE,no
overcast,FALSE,yes
rainy,FALSE,yes
rainy,FALSE,yes
rainy,TRUE,no
overcast,TRUE,yes
sunny,FALSE,no
sunny,FALSE,yes
rainy,FALSE,yes
sunny,TRUE,yes
overcast,TRUE,yes
overcast,FALSE,yes
rainy,TRUE,no
new.csv_weather-AttributeSelect

5. Decision Tree Induction using J48 Classifier

Input file (CLASSIFY.CSV)

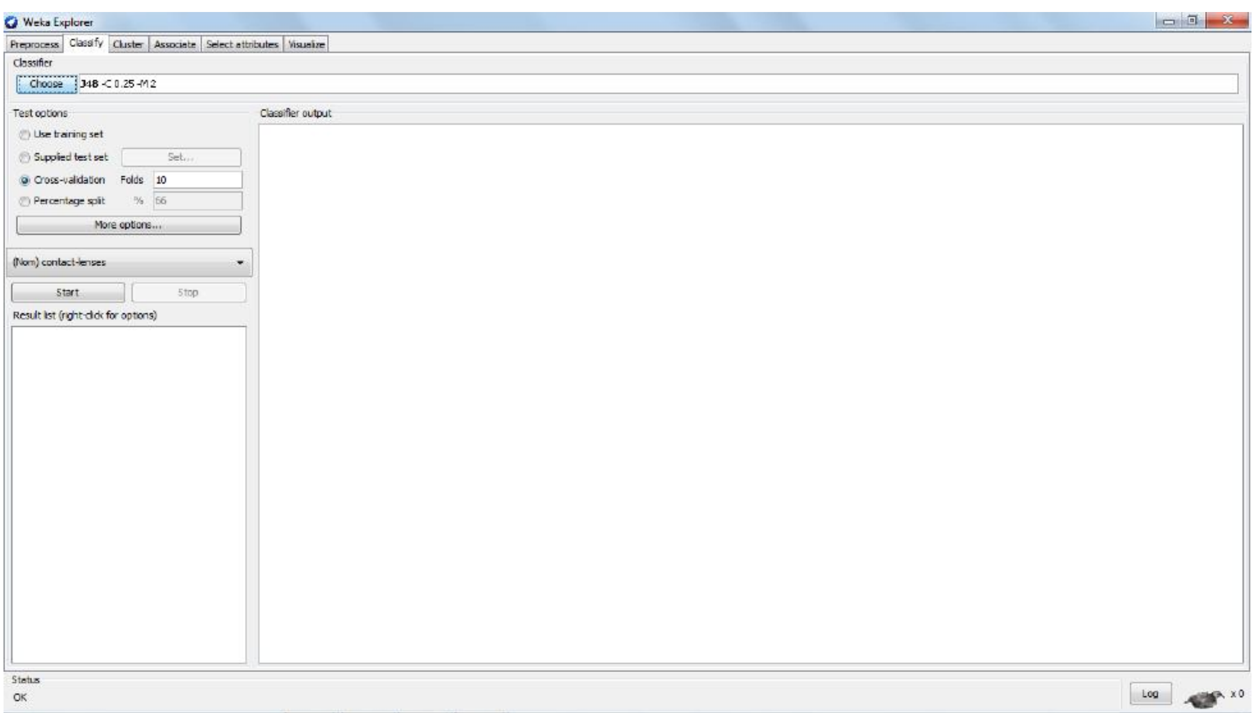
Procedure:

In preprocessor tab, choose the input file

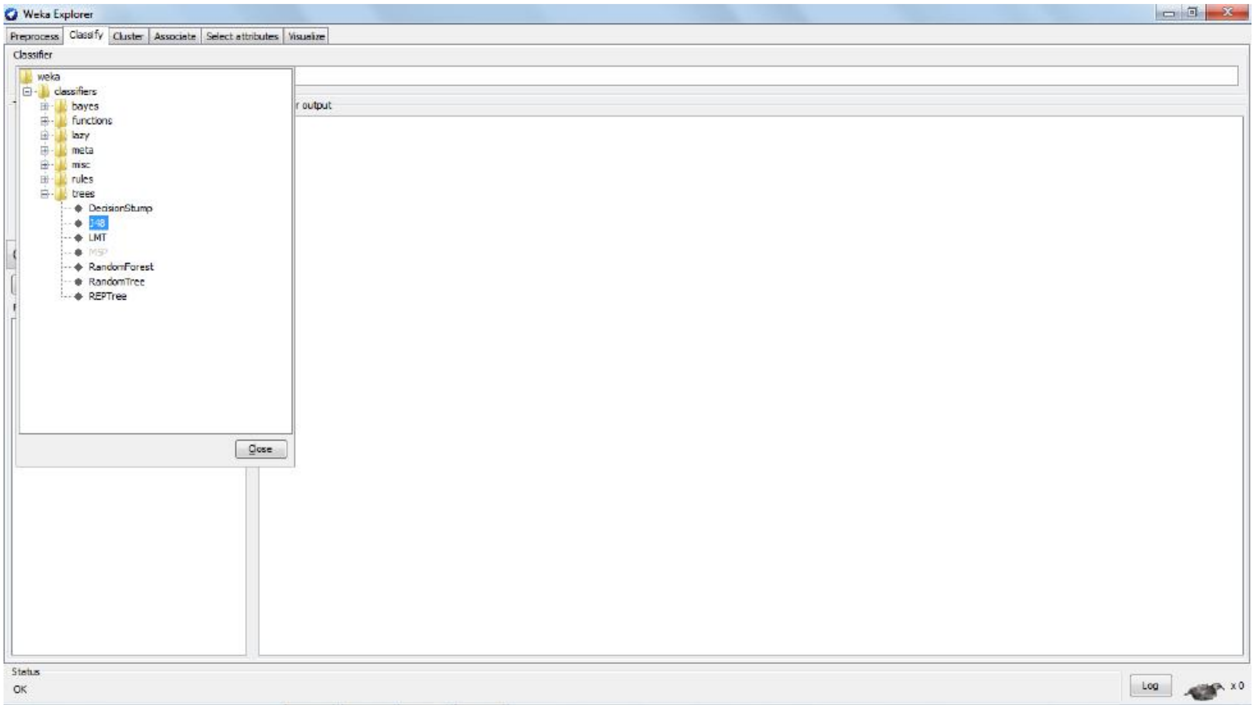



LOAD the file classify.csv

Choose the classify tab in the weka explorer window. Under the classify tab click on the choose button and select the j48 under tree as shown in the following.

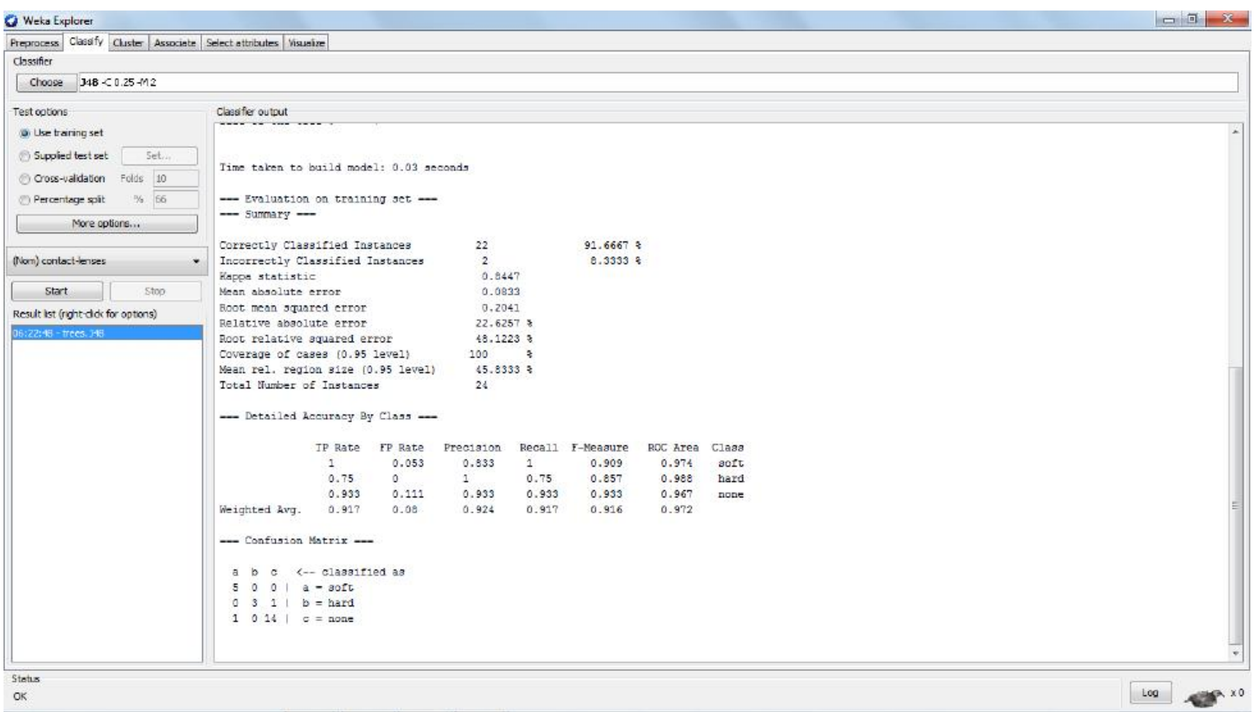


Select j48 algorithm (decision tree algorithm)



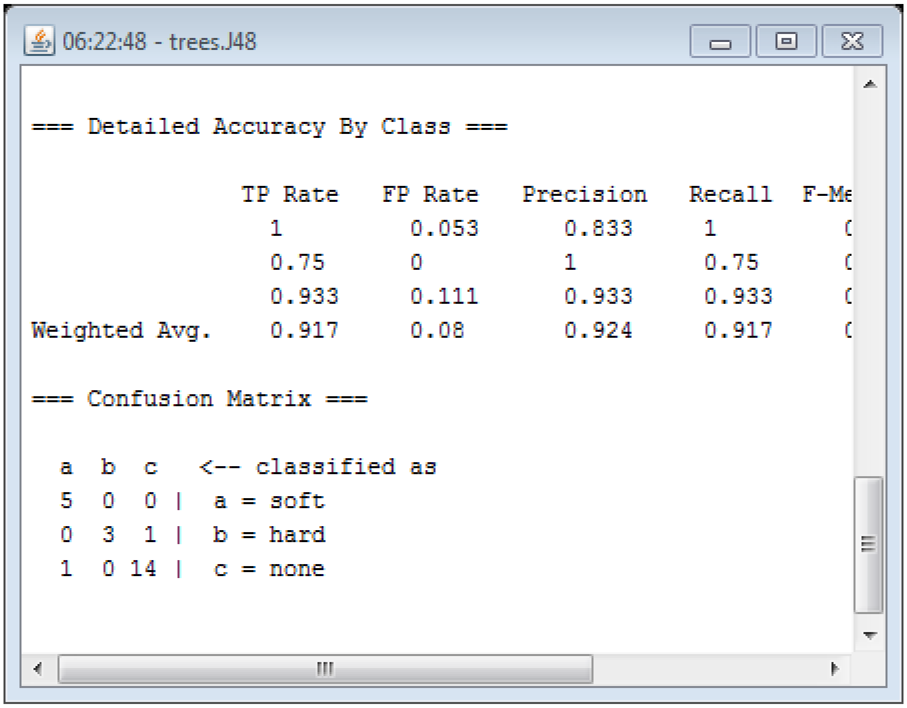

Now select the “use training set “ under the test option located at the left of the weka explorer window and click on the on start button.

The output is presented in the classifier output window in weka explorer window.

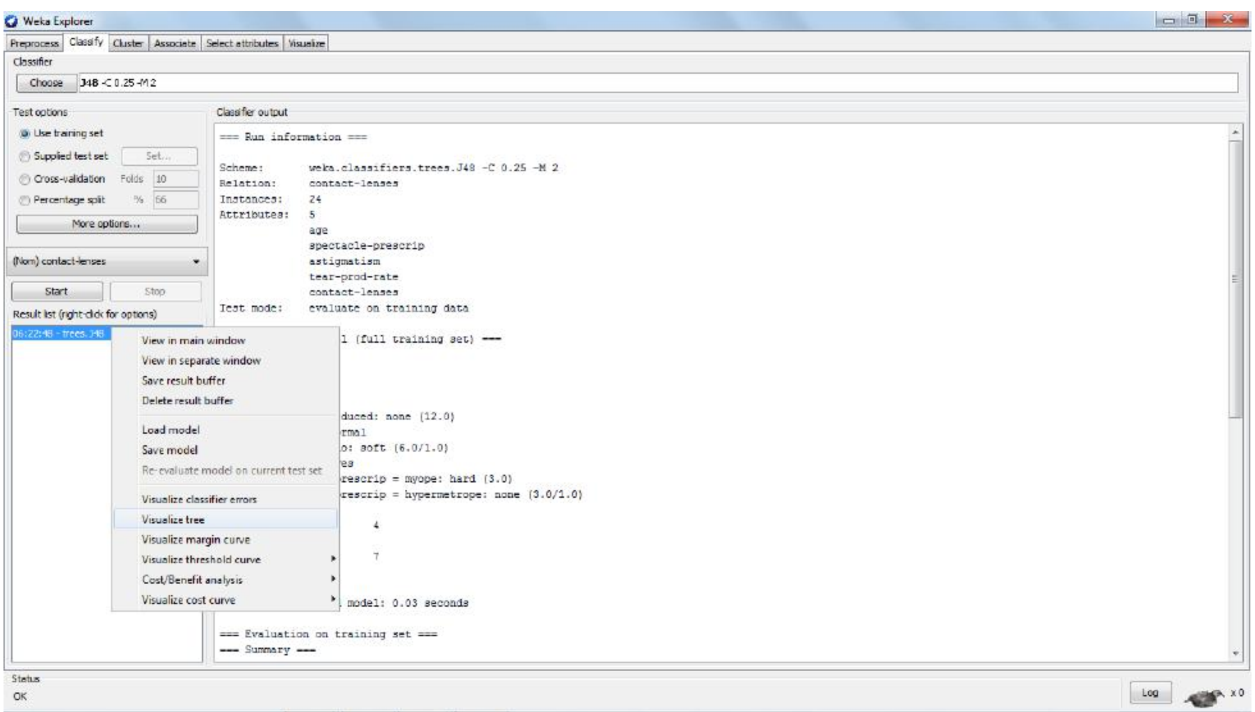


Shows output in classifier output window in weka explorer window.

We can also view the output in a separate window by right clicking on the option in result list clicking on “view in separate window”

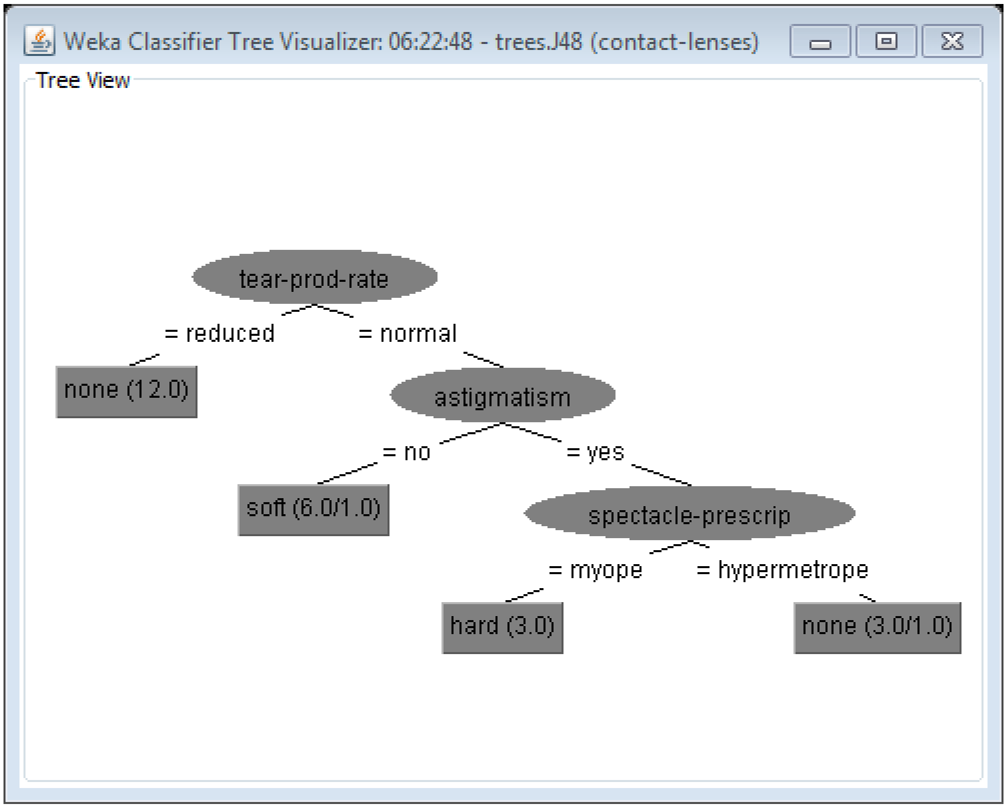



Under the result list right click on the item to get the options as shown and select the option “visualilize tree” option.



**Output screen shows how to select visualize as tree option**

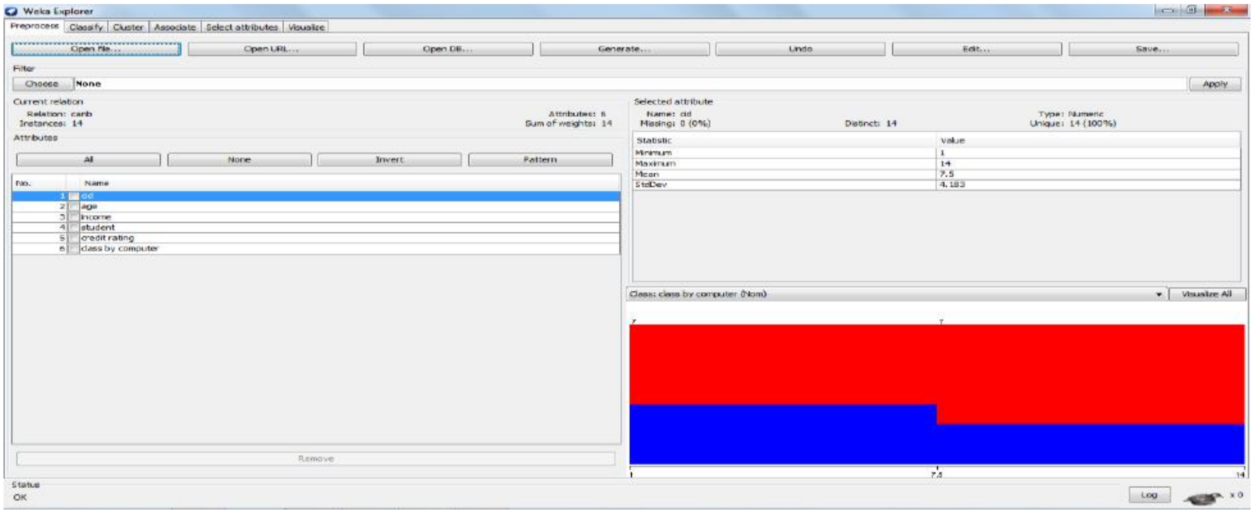
After selecting the "visualize tree " option the output is represented as tree in a separate window shown



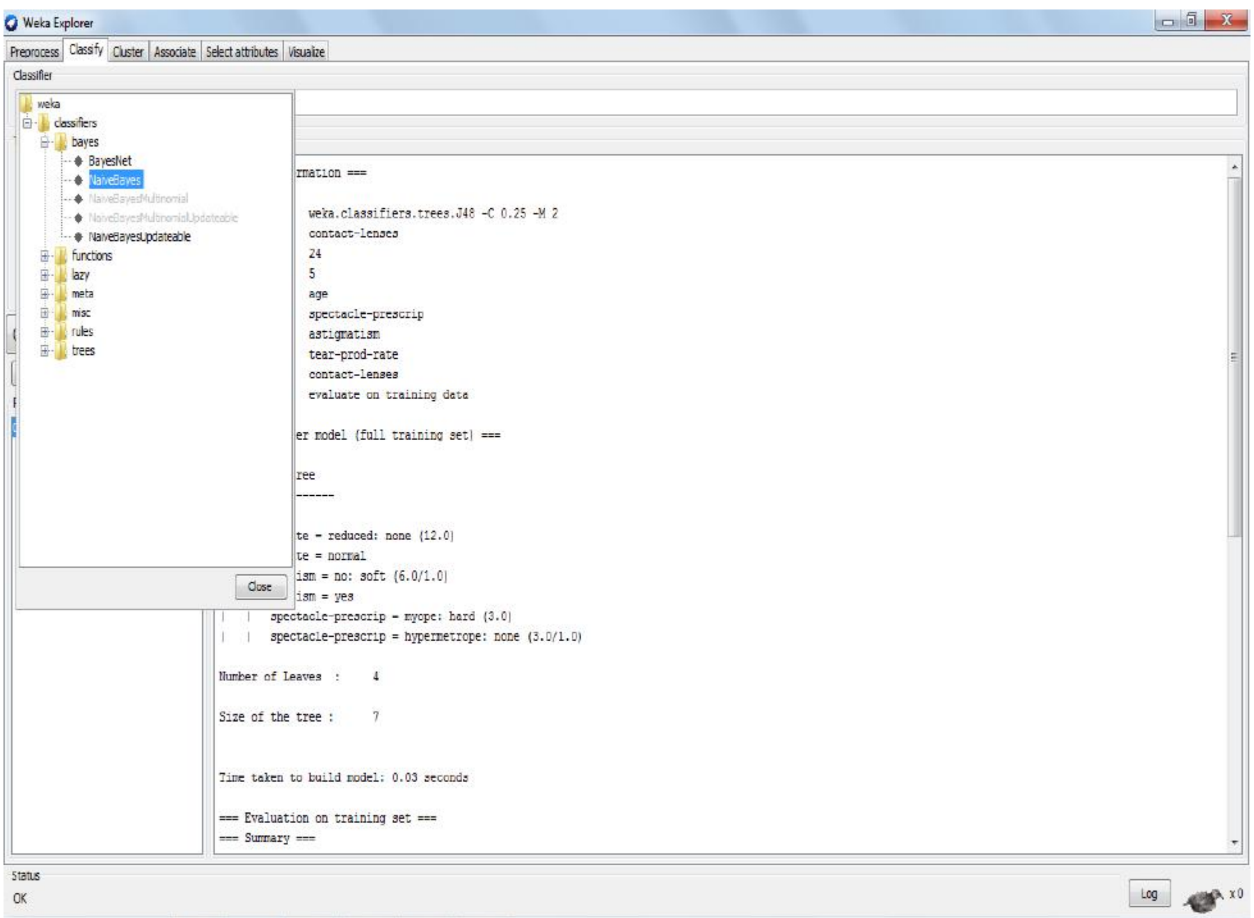
6. Classification Using Naïve Bayes Classifier

**Input file (CANB.CSV)**

Loading the input file into the explorer to perform the classification as shown in the below figure

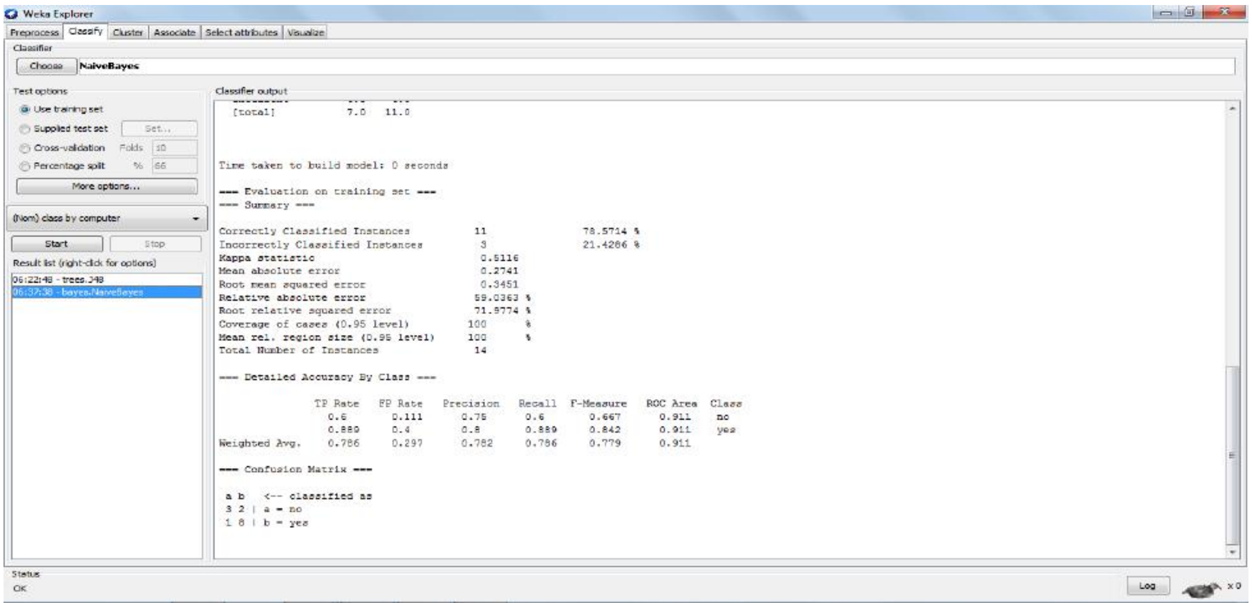


After loading the input file named canb.csv as shown in fig, choose the classify tab in the WEKA explorer window. Under the classify tab click on choose button and select the NaïveBayes under Bayes as shown ,

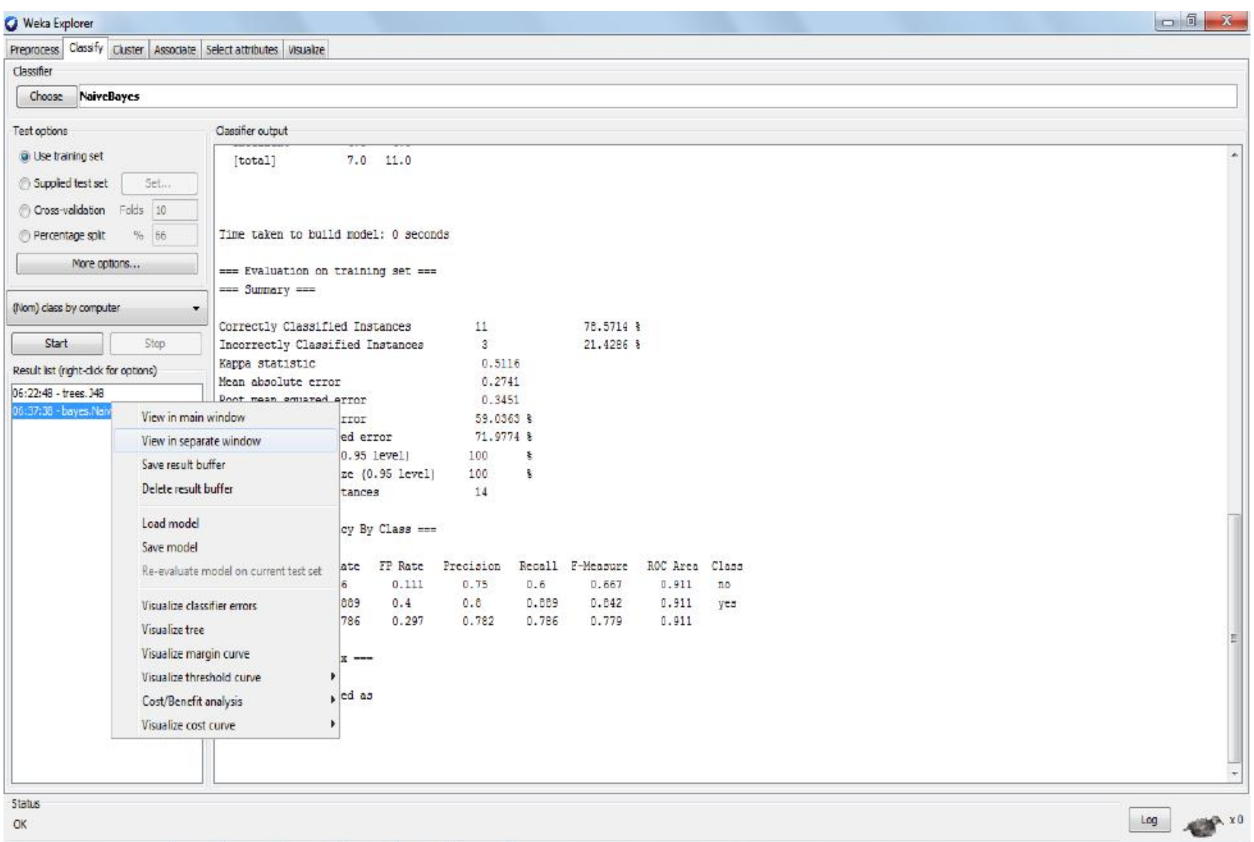


Select the “use training set “ under the test option located at the left of the weka explorer and click on start button.

The output is represented in  the classifier output window in weka explorer window,




Now we are able to view the output in a separate window by right clicking on the option in result list and clicking on “view in separate window “



```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     canb
Instances:    14
Attributes:   6
cid
age
income
student
credit rating
class by computer
Test mode:    evaluate on training data

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

Naive Bayes Classifier

               Class
Attribute          no    yes
              (0.38) (0.63)
===============================
cid
mean              6.2 8.2222
std. dev.      4.6648 3.4247
weight sum          5      9
precision           1      1

age
youth             4.0    3.0
middle            1.0    5.0
senior            3.0    4.0
  [total]         8.0   12.0

income
high              3.0    3.0
medium            3.0    5.0
low               2.0    4.0
  [total]         8.0   12.0

student
no                5.0    4.0
yes               2.0    7.0
  [total]         7.0   11.0

credit rating
fair              3.0    7.0
excellent         4.0    4.0
  [total]         7.0   11.0



Time taken to build model: 0 seconds

=== Evaluation on training set ===
```

=== Summary ===

```
Correctly Classified Instances          11               78.5714 %
Incorrectly Classified Instances         3               21.4286 %
Kappa statistic                          0.5116
Mean absolute error                      0.2741
Root mean squared error                  0.3451
Relative absolute error                 59.0363 %
Root relative squared error             71.9774 %
Coverage of cases (0.95 level)         100      %
Mean rel. region size (0.95 level)     100      %
Total Number of Instances               14
```

=== Detailed Accuracy By Class ===

```
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.6       0.111      0.75      0.6       0.667      0.911    no
                 0.889     0.4        0.8       0.889     0.842      0.911    yes
Weighted Avg.    0.786     0.297      0.782     0.786     0.779      0.911
```

=== Confusion Matrix ===

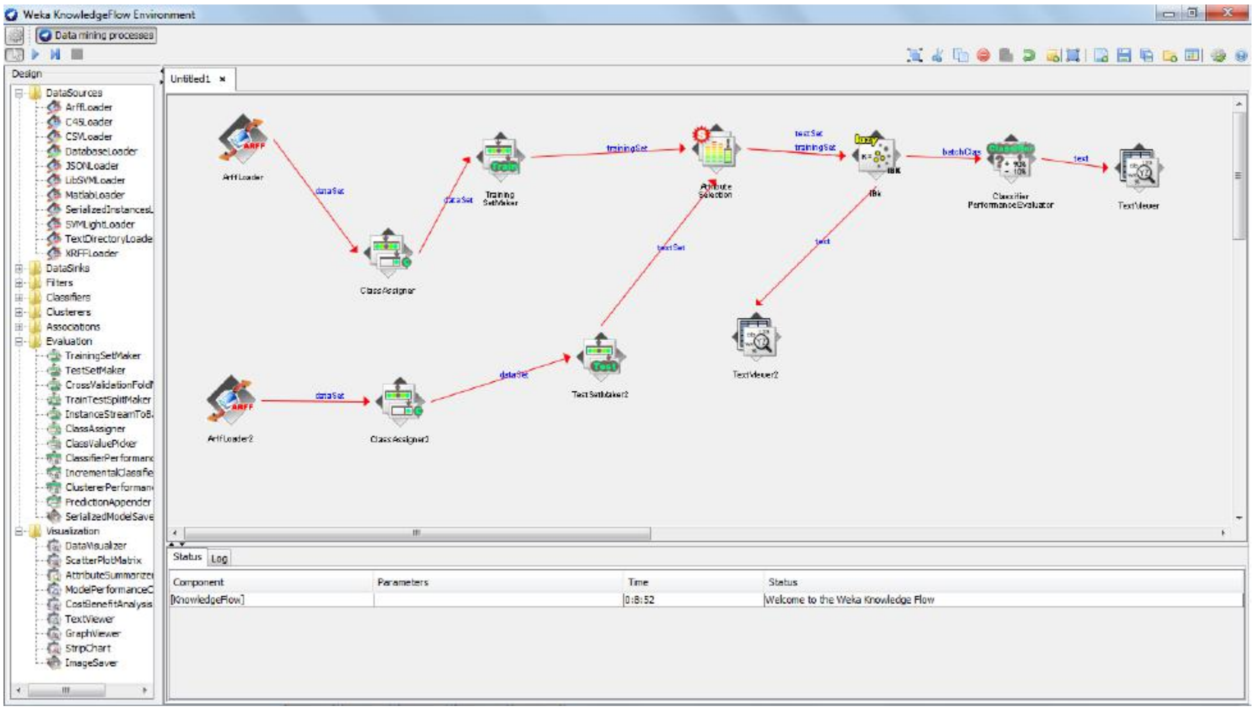
```
a b   <-- classified as
 3 2 | a = no
 1 8 | b = yes
```

7. To Evaluate the Performance of a Classifier

**Input:** weather.arff

**Procedure:**

The arrangement and linking of icons for: " Training set and test set"




=== Evaluation result ===

Scheme: IBk

Options: -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""

Relation: weather-weka.filters.supervised.attribute.AttributeSelection-Eweka.attributeSelection.CfsSubsetEval-Sweka.attributeSelection.BestFirst -D 1 -N 5

Correctly Classified Instances 12 85.7143 %

Incorrectly Classified Instances 2 14.2857 %

Kappa statistic 0.6889

Mean absolute error 0.1864

Root mean squared error 0.2897

Coverage of cases (0.95 level) 100 %

Total Number of Instances 14

=== Detailed Accuracy By Class ===

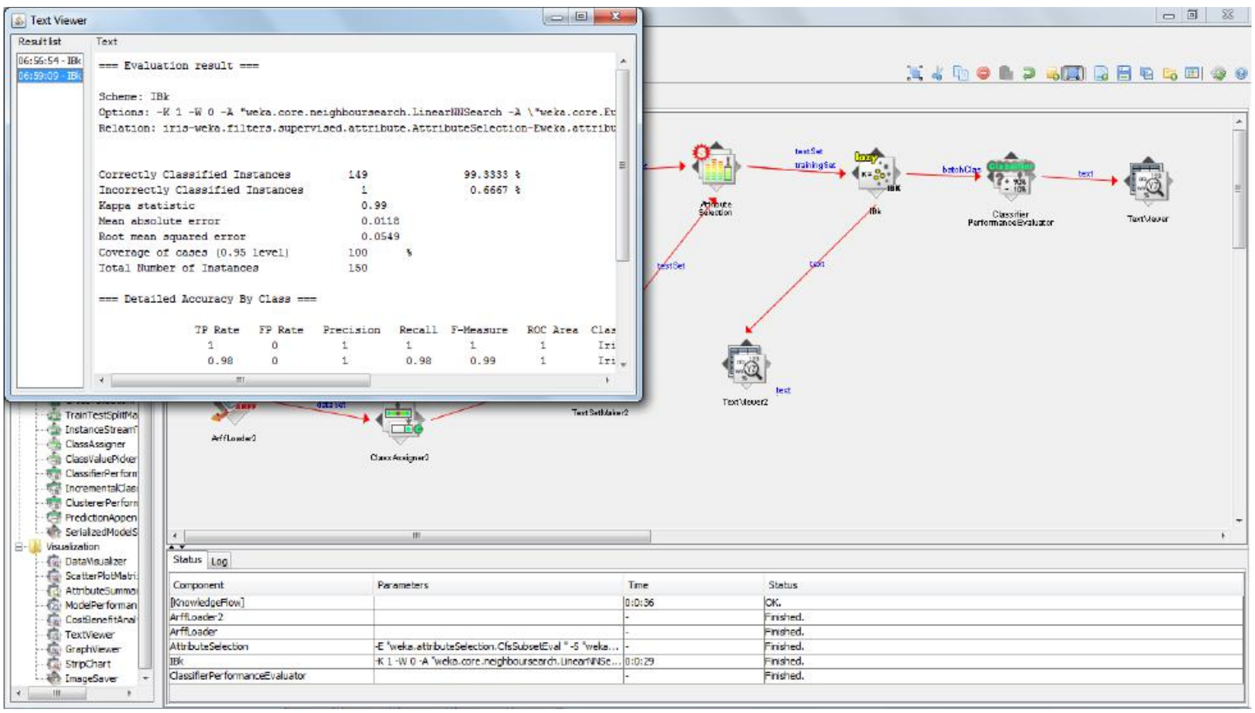
| | TP Rate | FP Rate | Precision | Recall | F-Measure | ROC Area | Class |
|---|---|---|---|---|---|---|---|
| | 0.889 | 0.2 | 0.889 | 0.889 | 0.889 | 0.944 | yes |
| | 0.8 | 0.111 | 0.8 | 0.8 | 0.8 | 0.944 | no |
| Weighted Avg. | 0.857 | 0.168 | 0.857 | 0.857 | 0.857 | 0.944 | |

=== Confusion Matrix ===

a b <-- classified as

8 1 | a = yes

1 4 | b = no



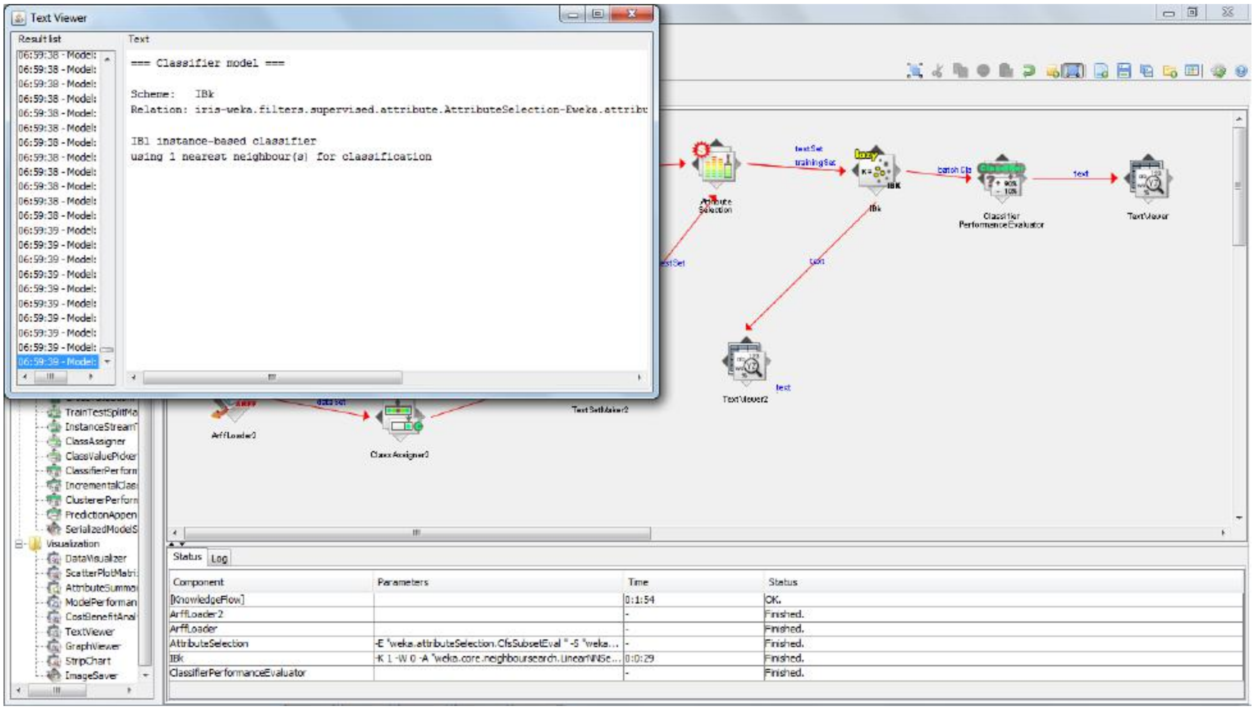
=== Classifier model ===

Scheme: IBk

Relation: iris-weka.filters.supervised.attribute.AttributeSelection-Eweka.attributeSelection.CfsSubsetEval-Sweka.attributeSelection.BestFirst -D 1 -N 5

IB1 instance-based classifier

using 1 nearest neighbour(s) for classification

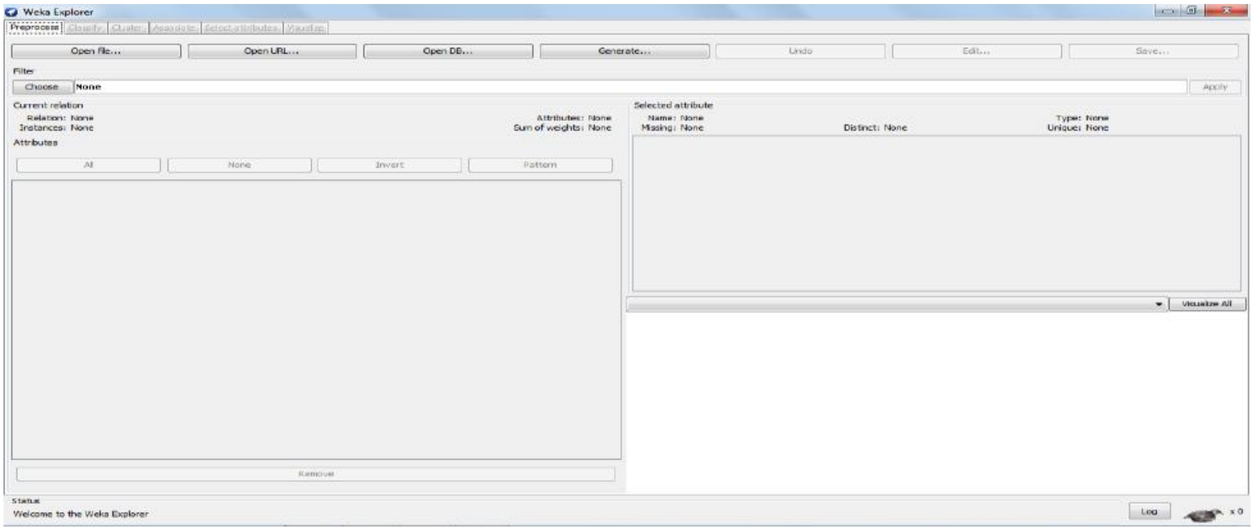


8. Clustering with k-means algorithm

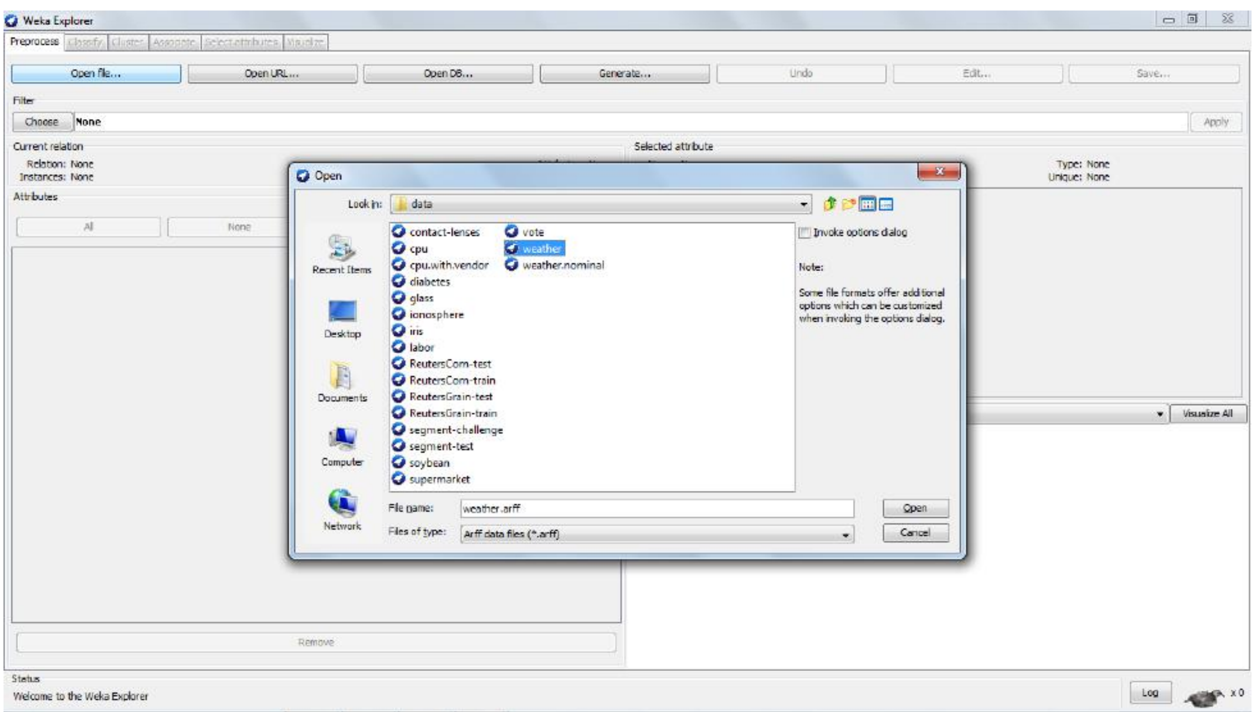
Input: weather.arff

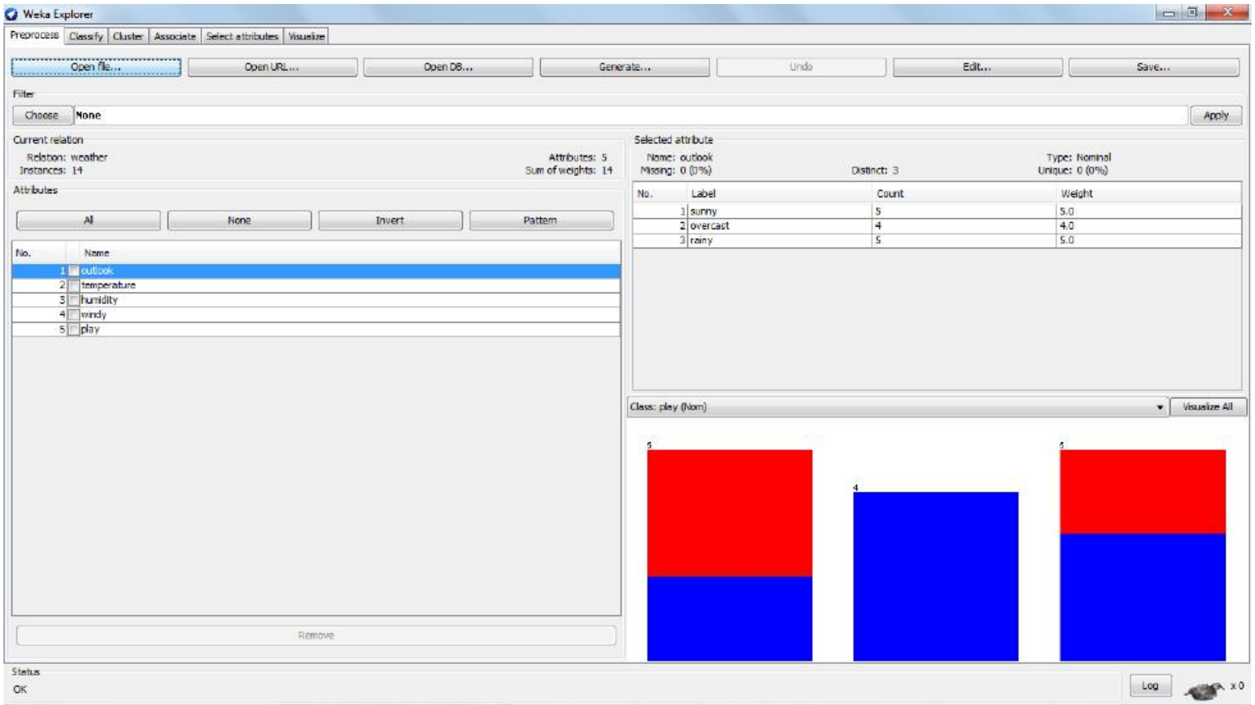
Procedure:

Go to weka  explorer environment.

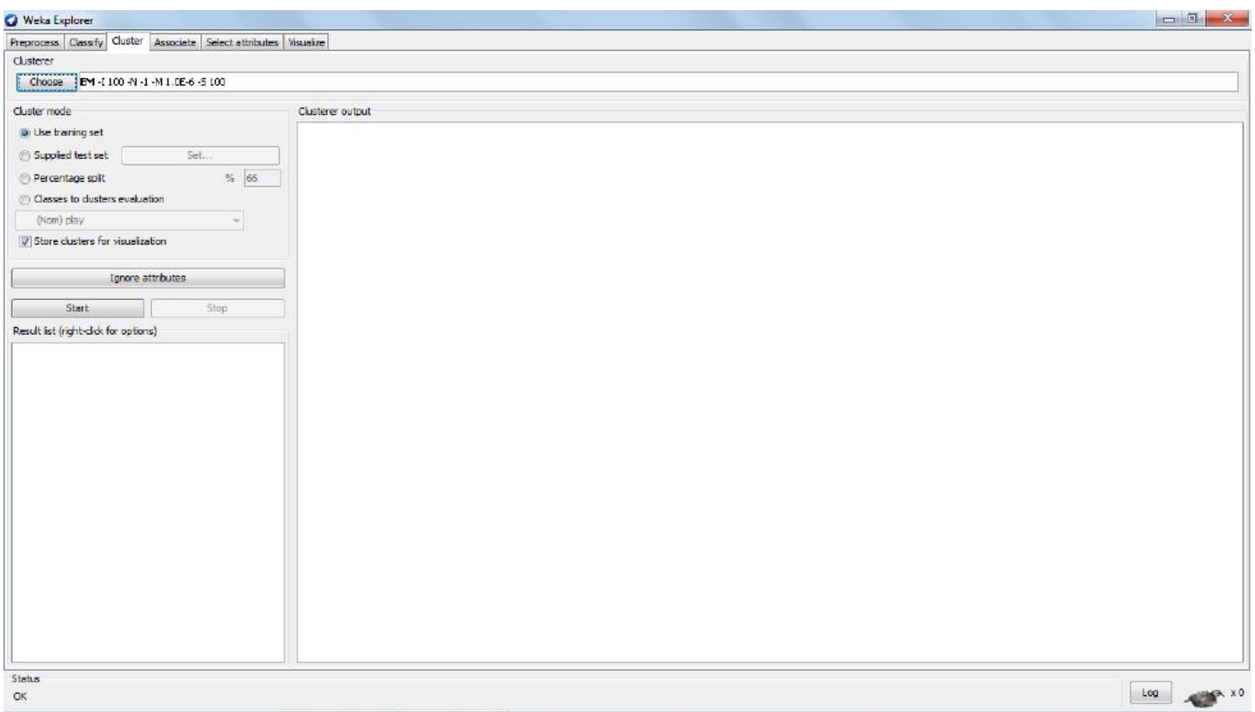



Load weather.arff  in preprocessor mode.

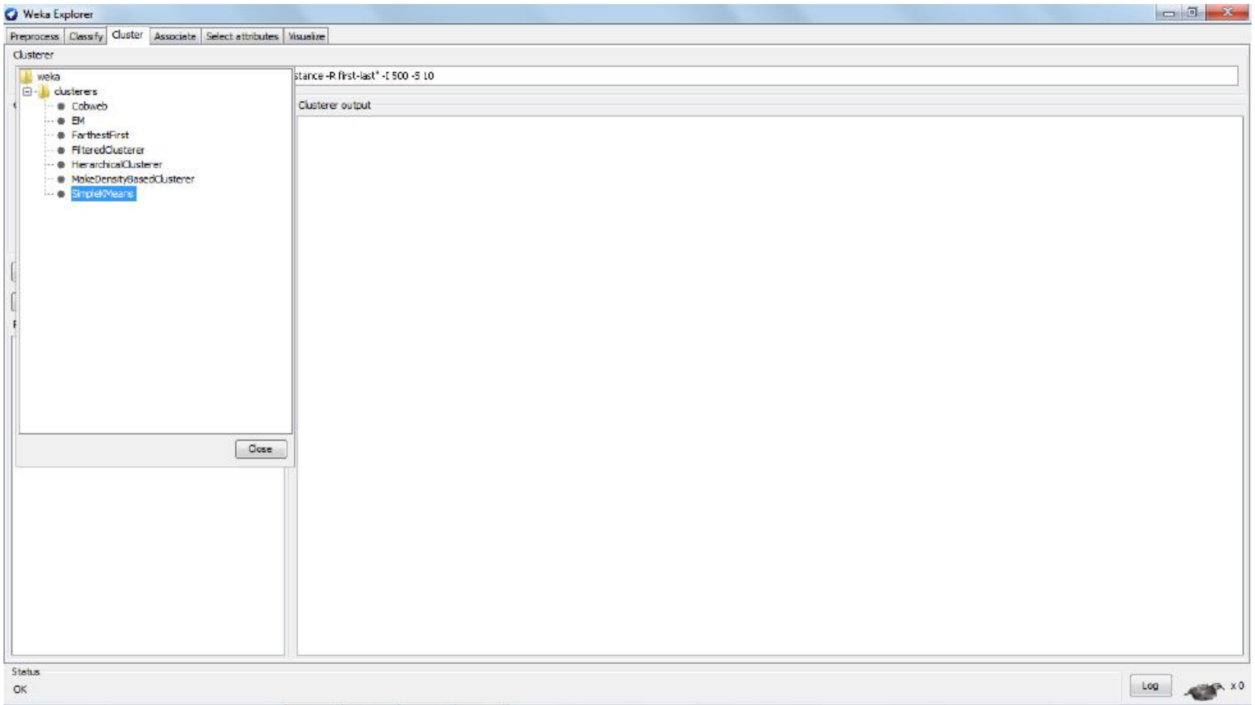




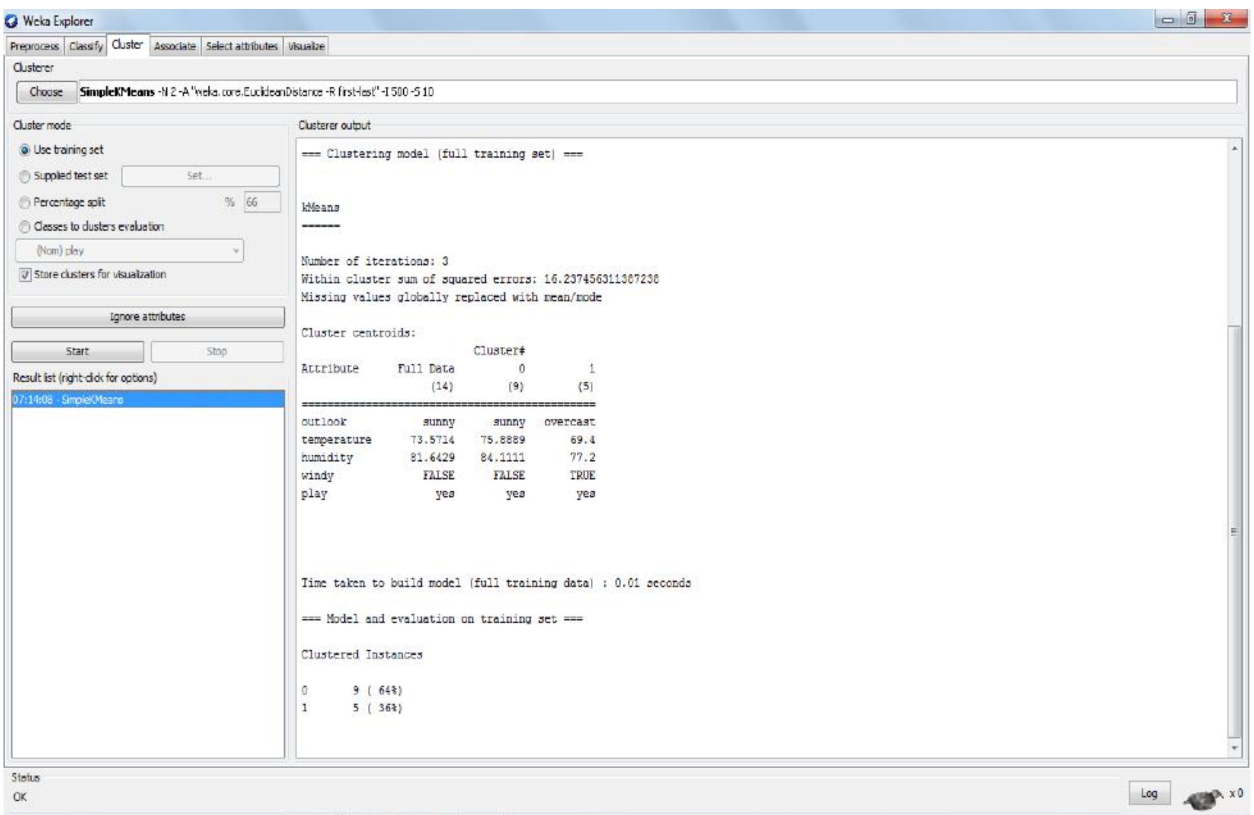
Click on cluster tab



Select a clustering algorithm (Use sample K-means)



Click on start button and get the clustering result in the output window.

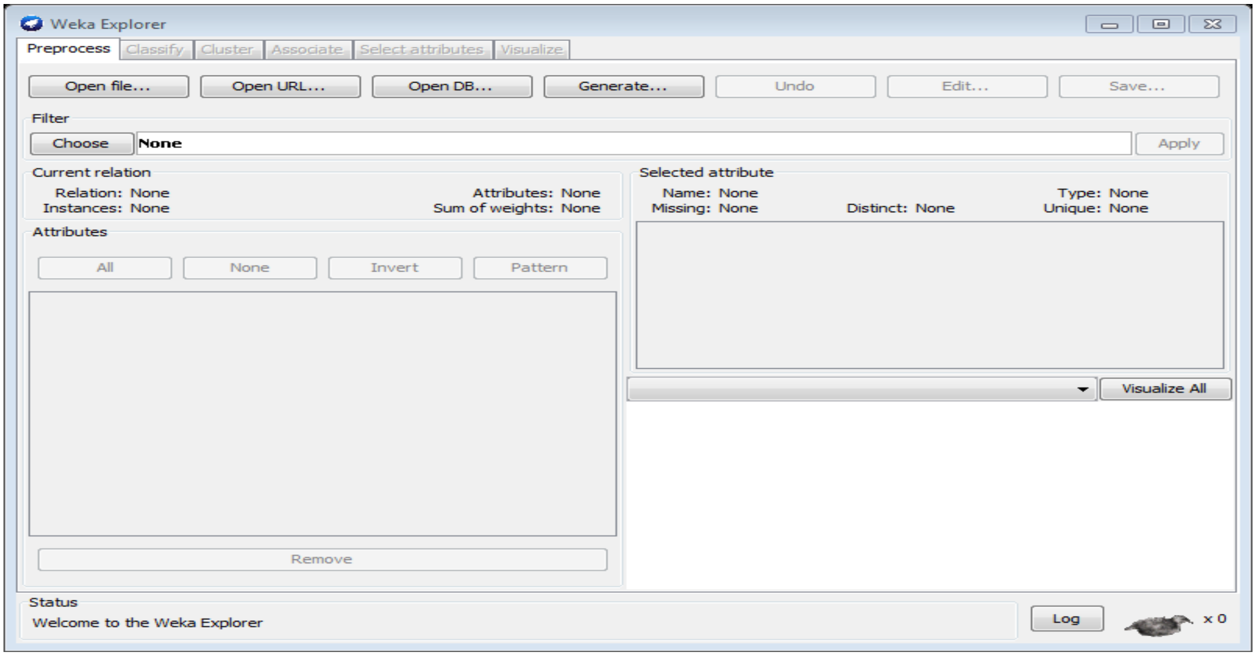


9. Clustering using EM

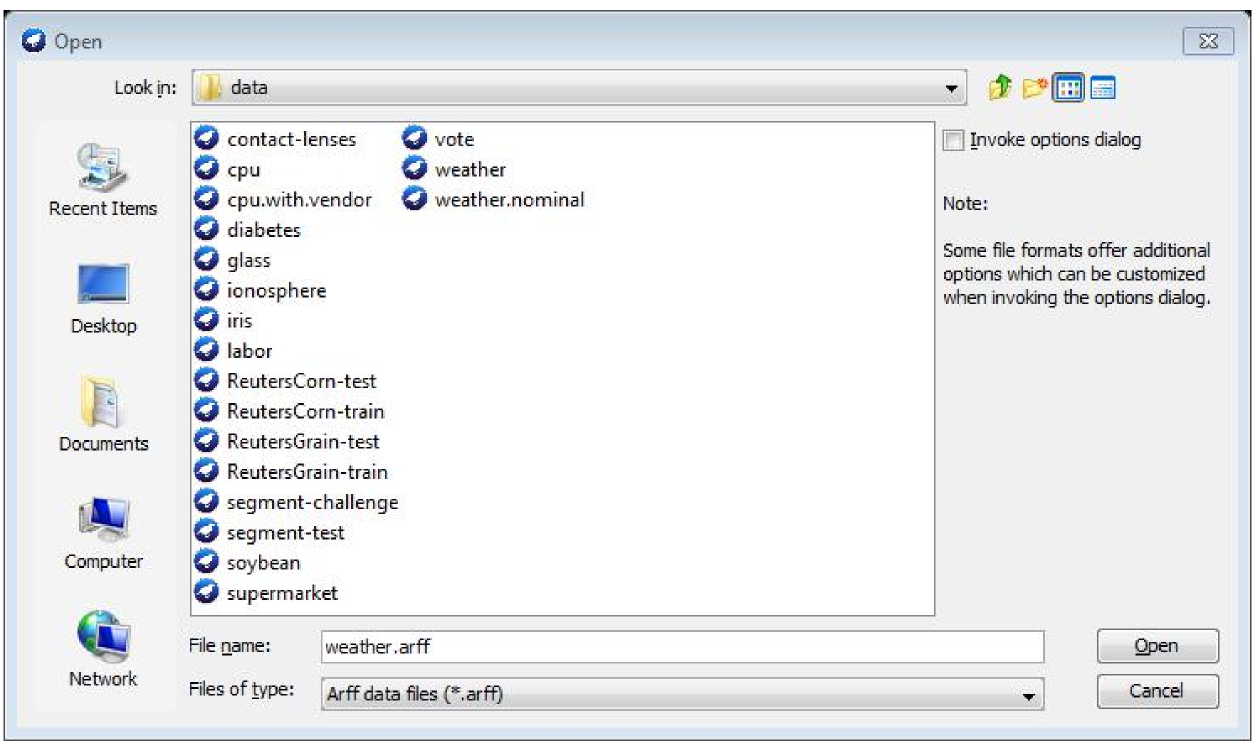
Input: weather.arff

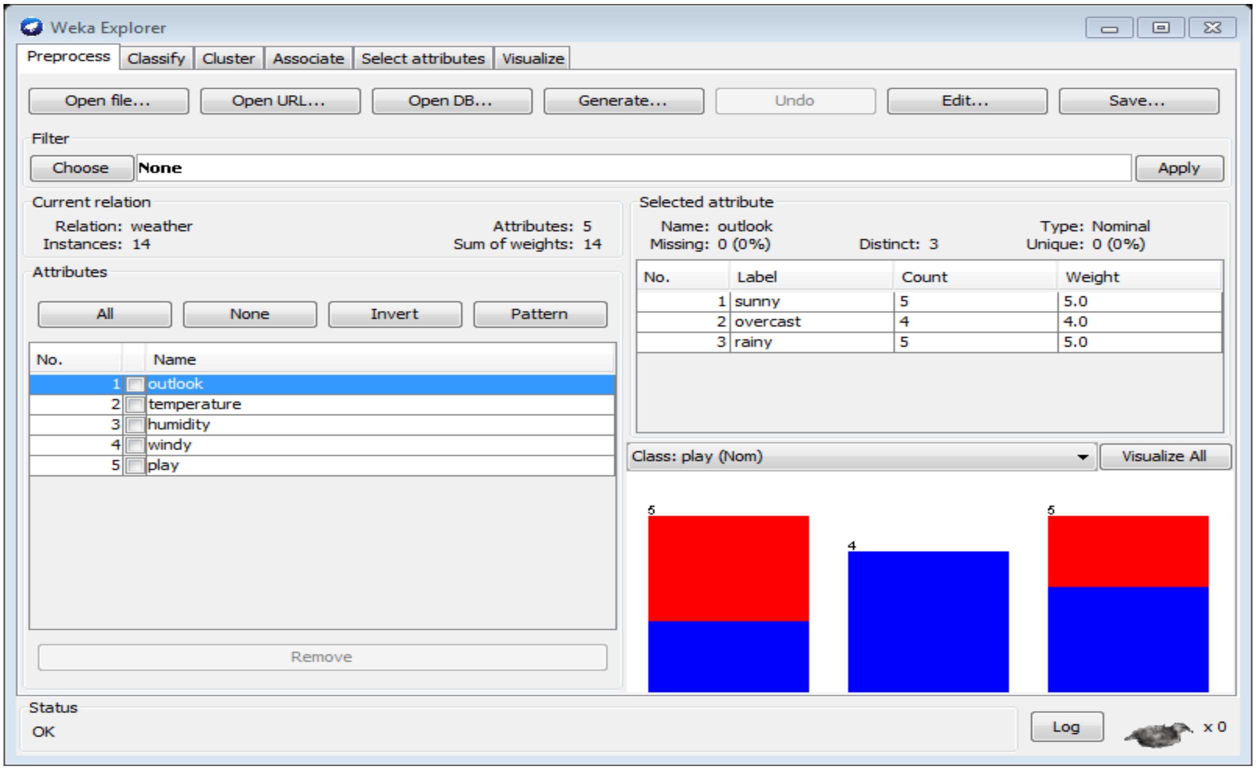
Procedure:

Go to weka explorer environment

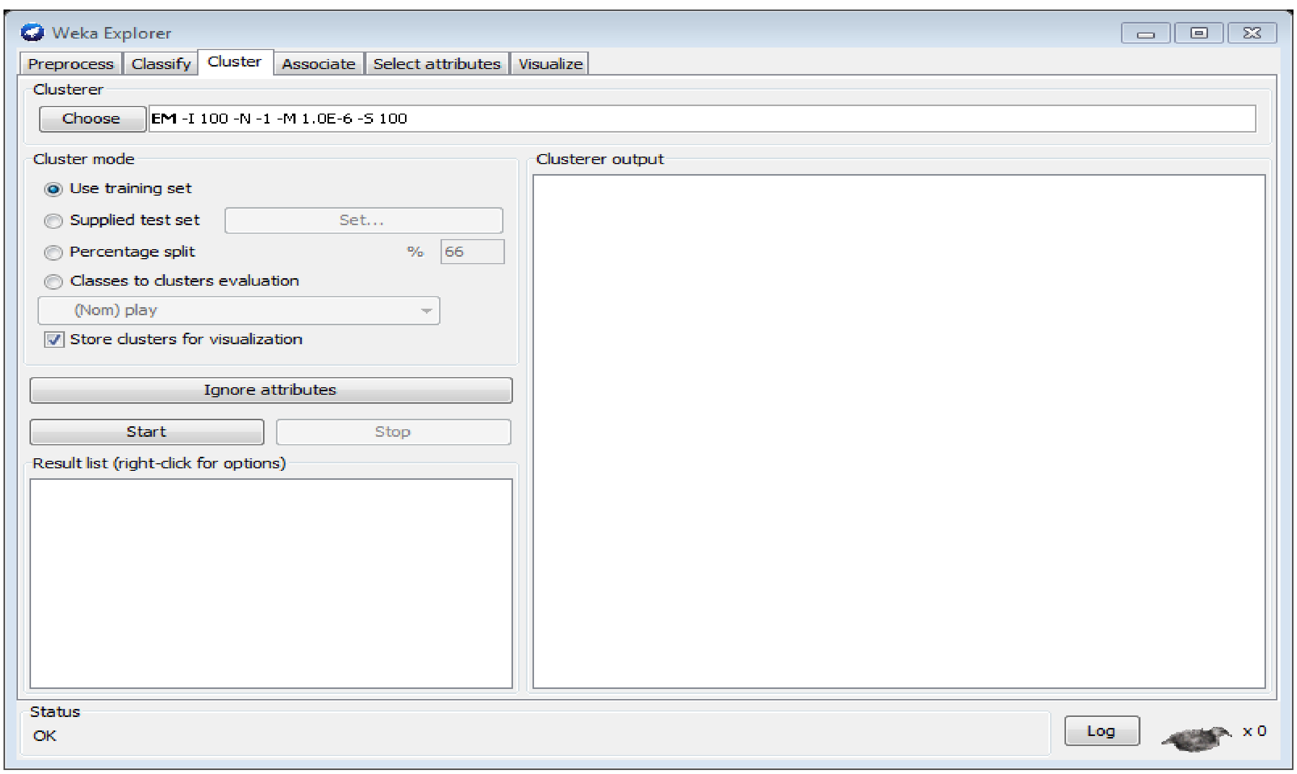



Load weather.arff in preprocess mode

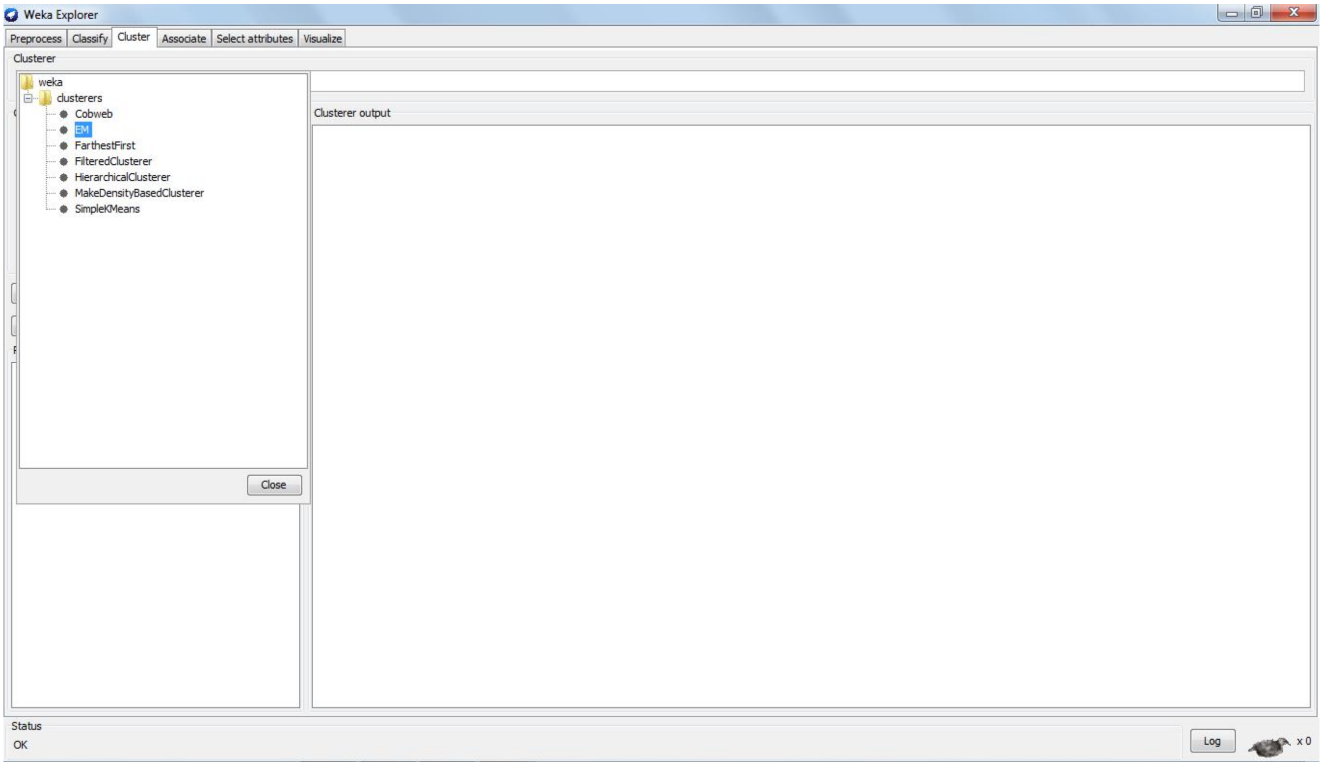




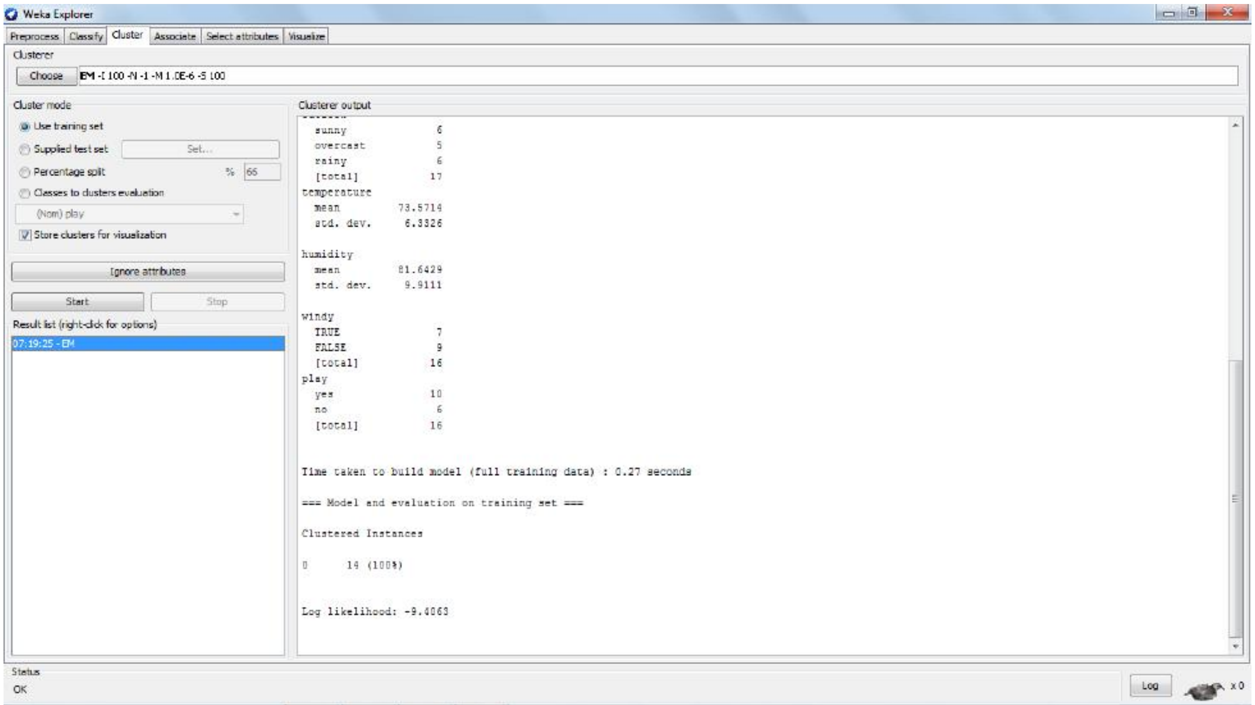

Click on cluster tab



Select a clustering algorithm (EM)



Click on start button and get clustering result in the output window.

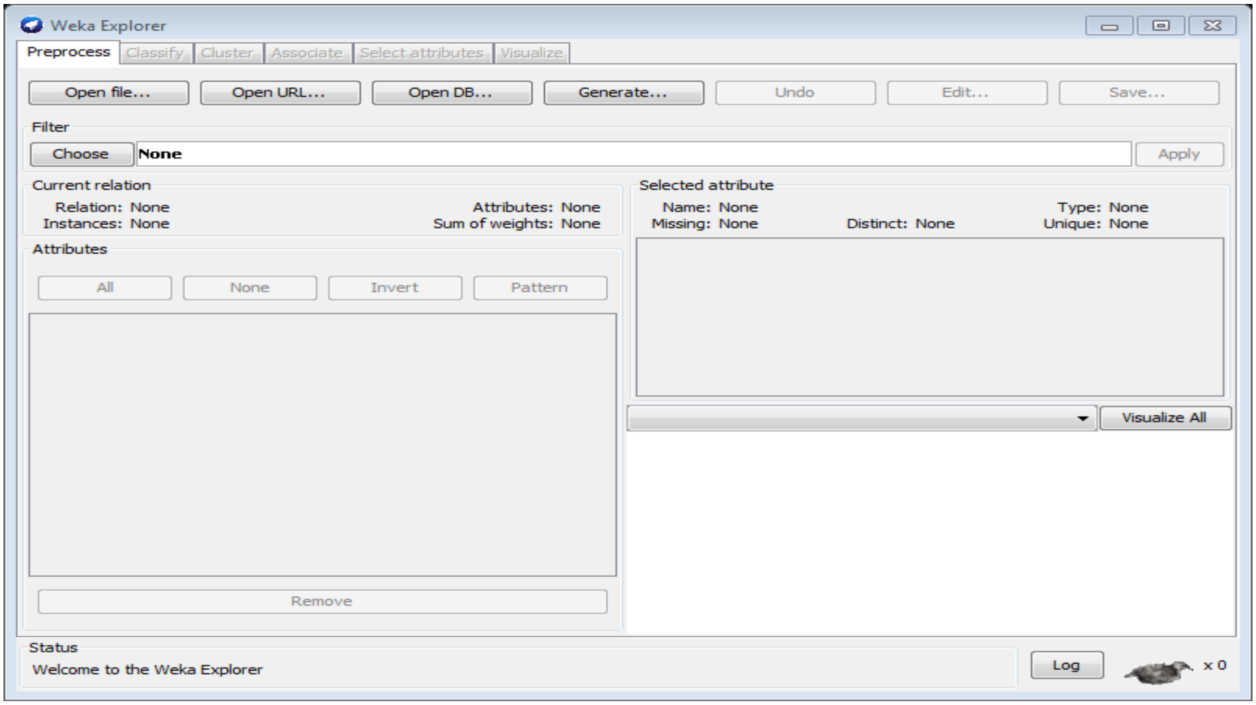


10. Clustering using COBWEB

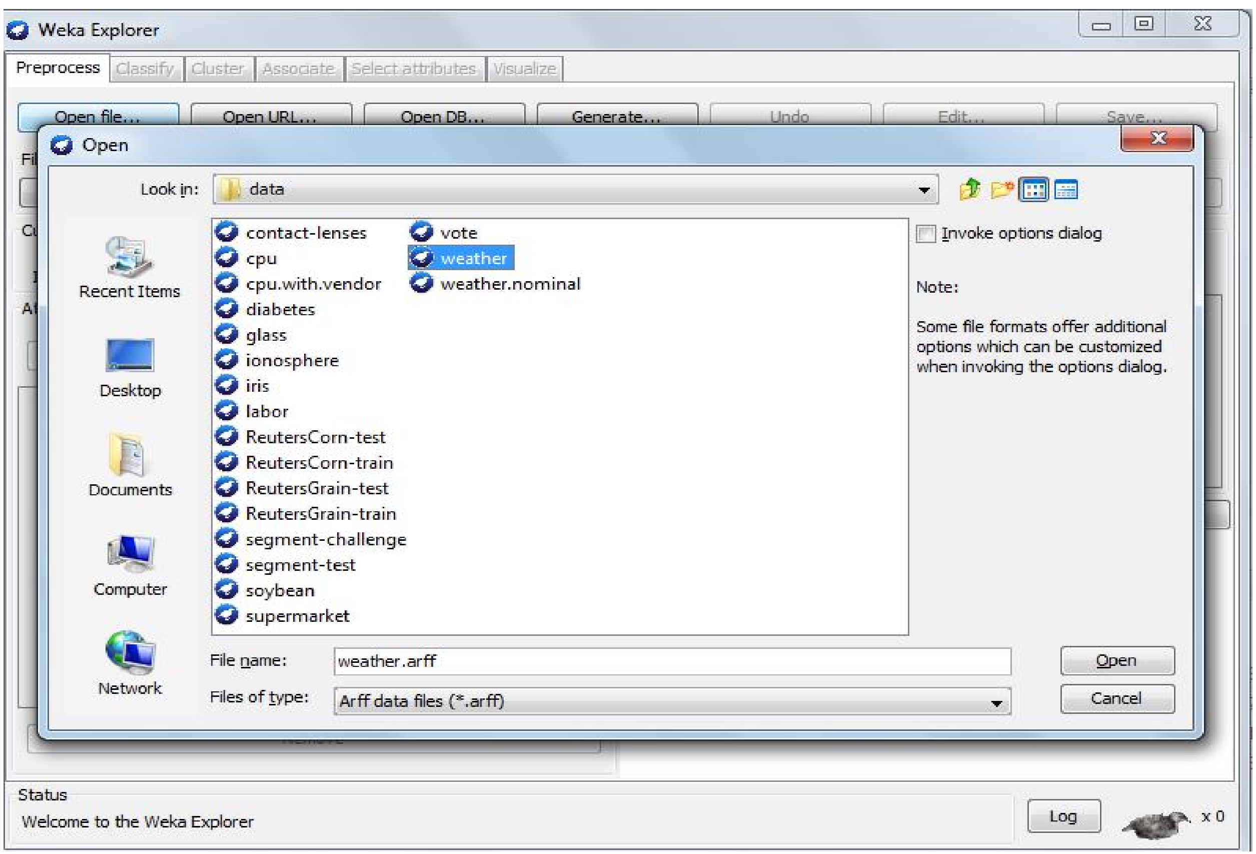
Input: weather.arff

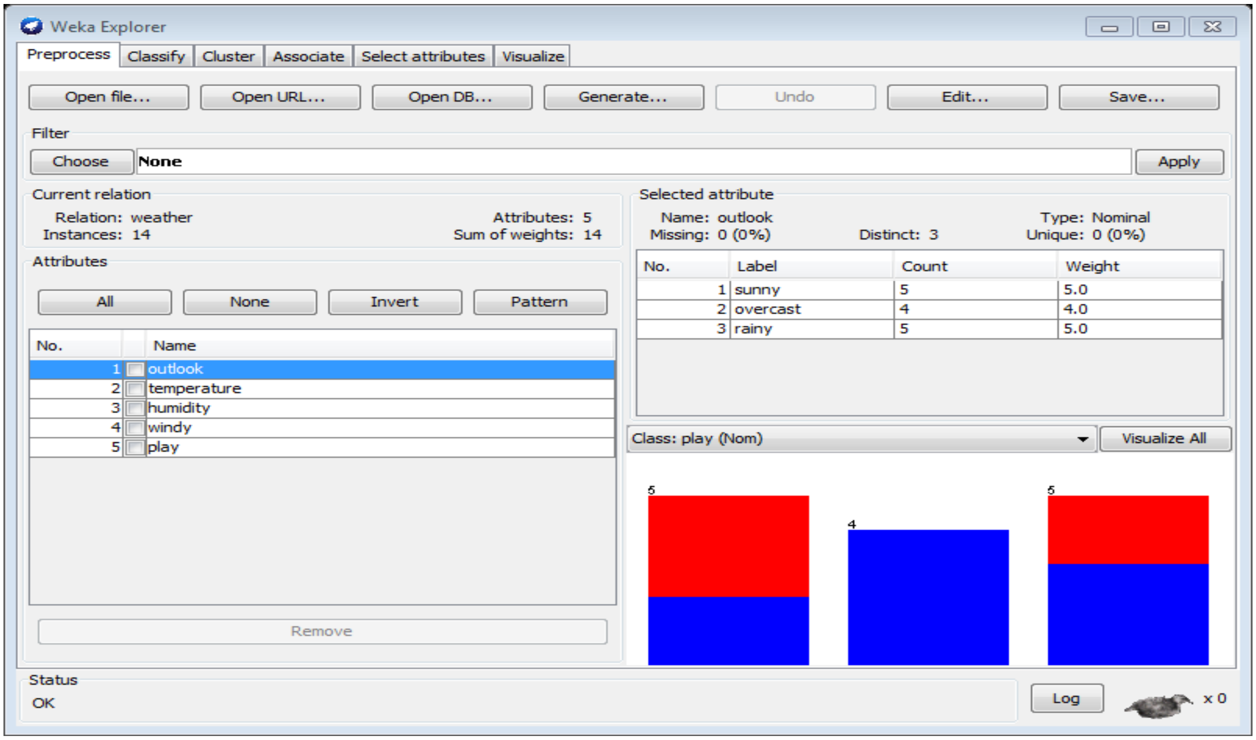
Procedure:

Go to explorer environment

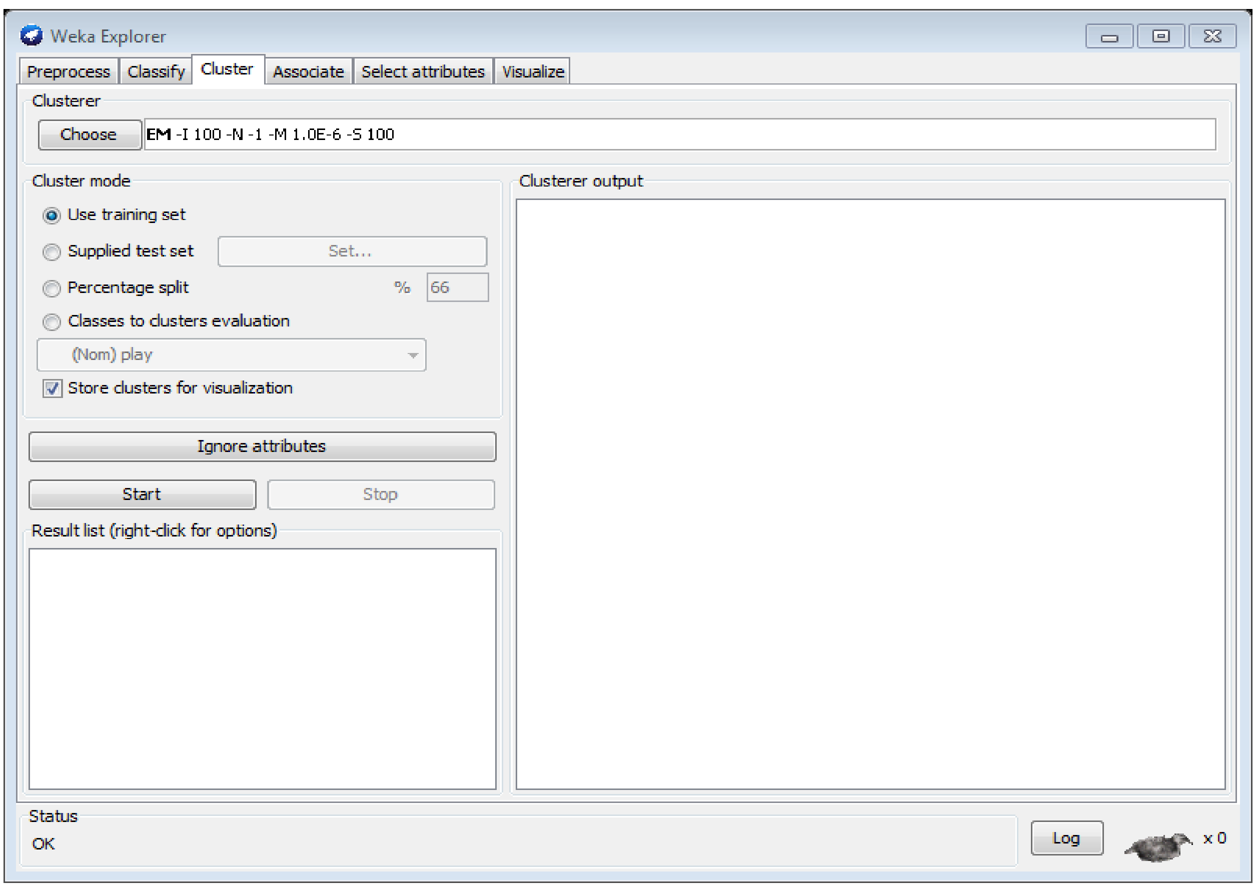



Load weather.arff in preprocess mode

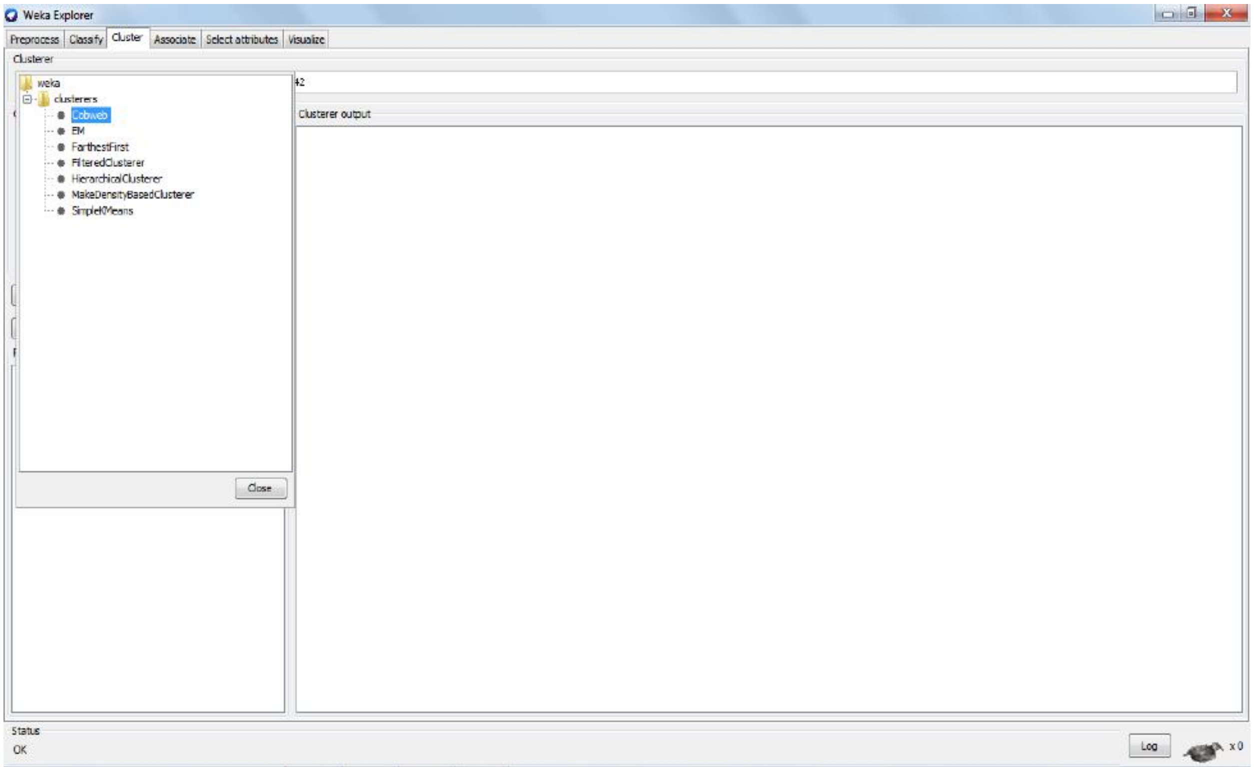


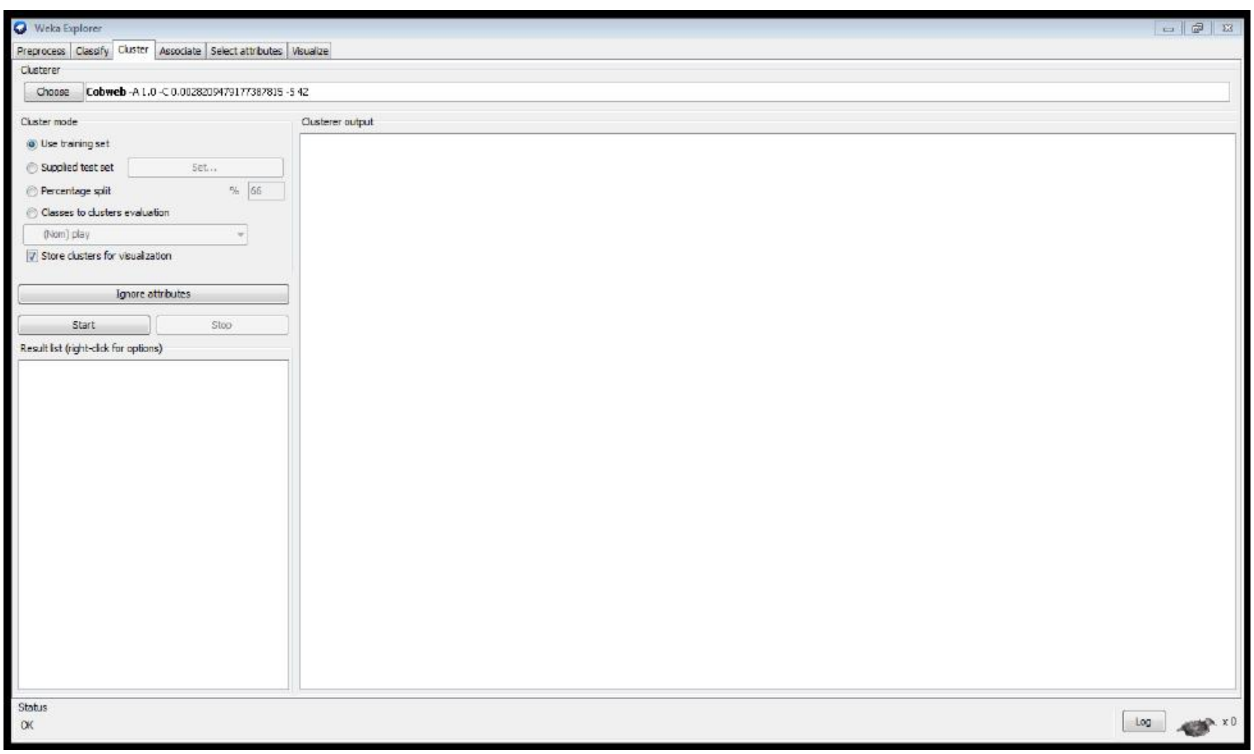



Click on cluster tab

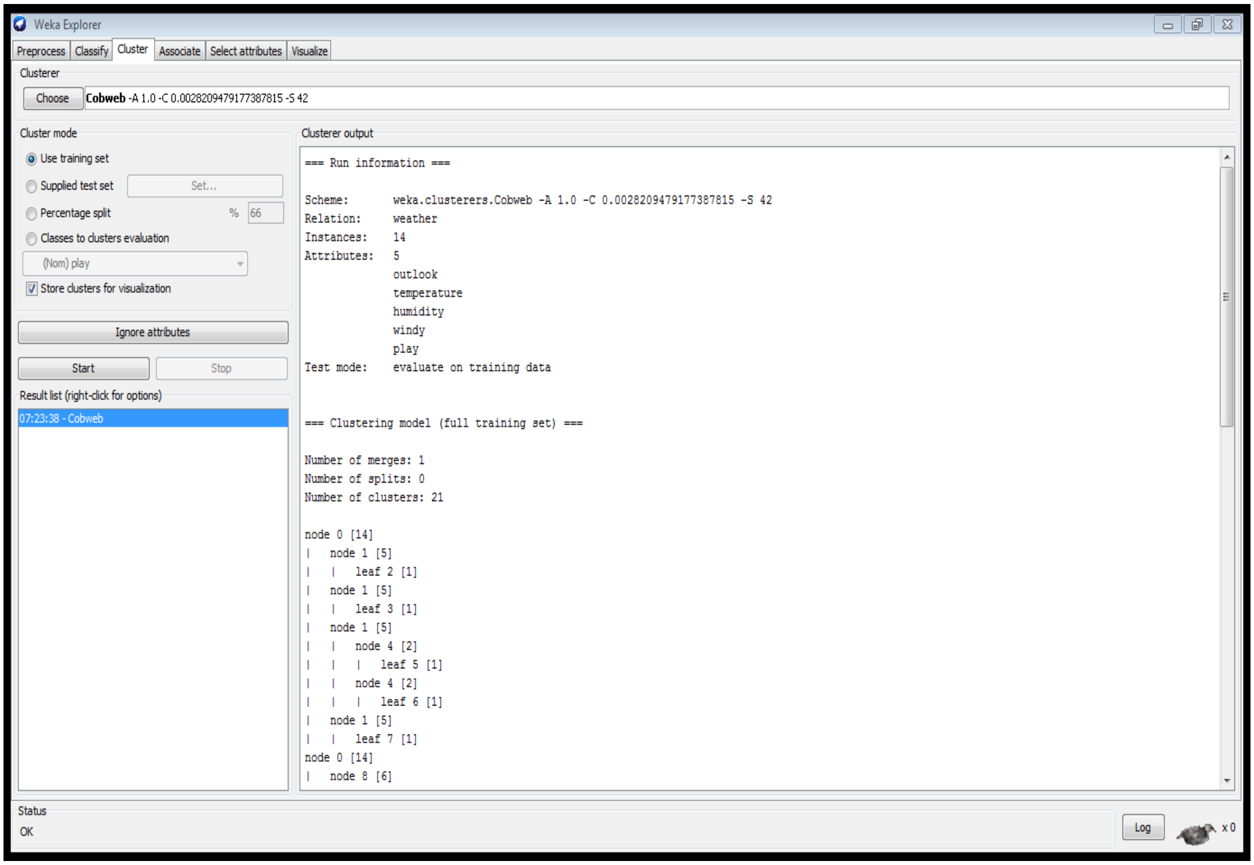


Select algorithm (COBWEB)clustering



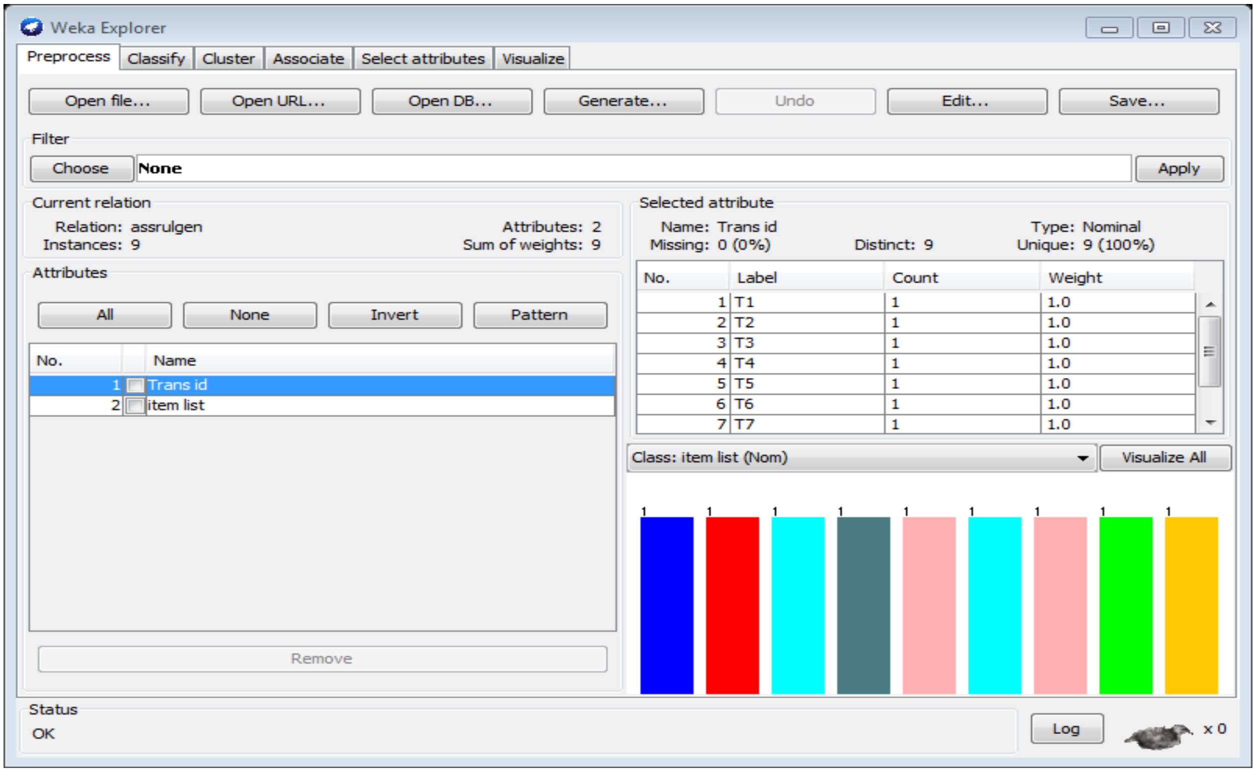
Click on start button and get the clustering result in the output window



11. To Generate Association Rules

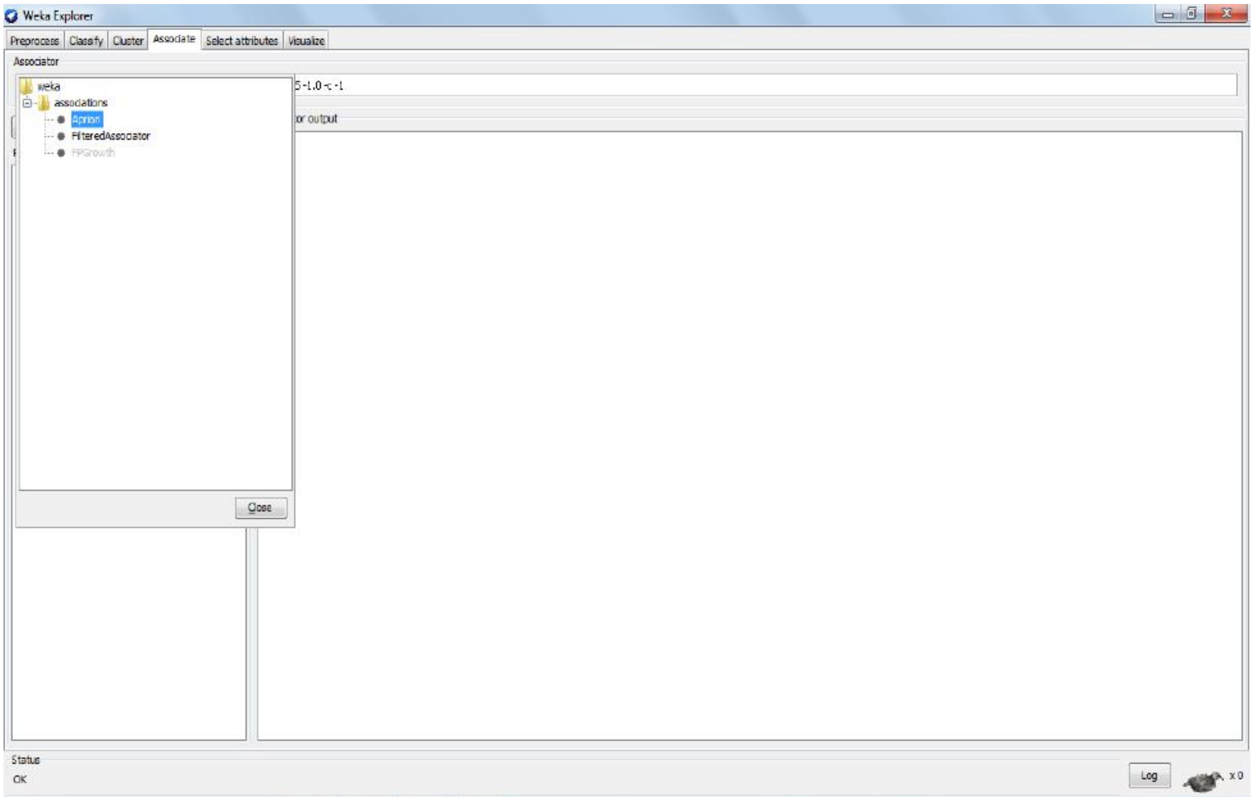
Input: assrulegen.arff

Load the input file into explorer to perform association as shown below.

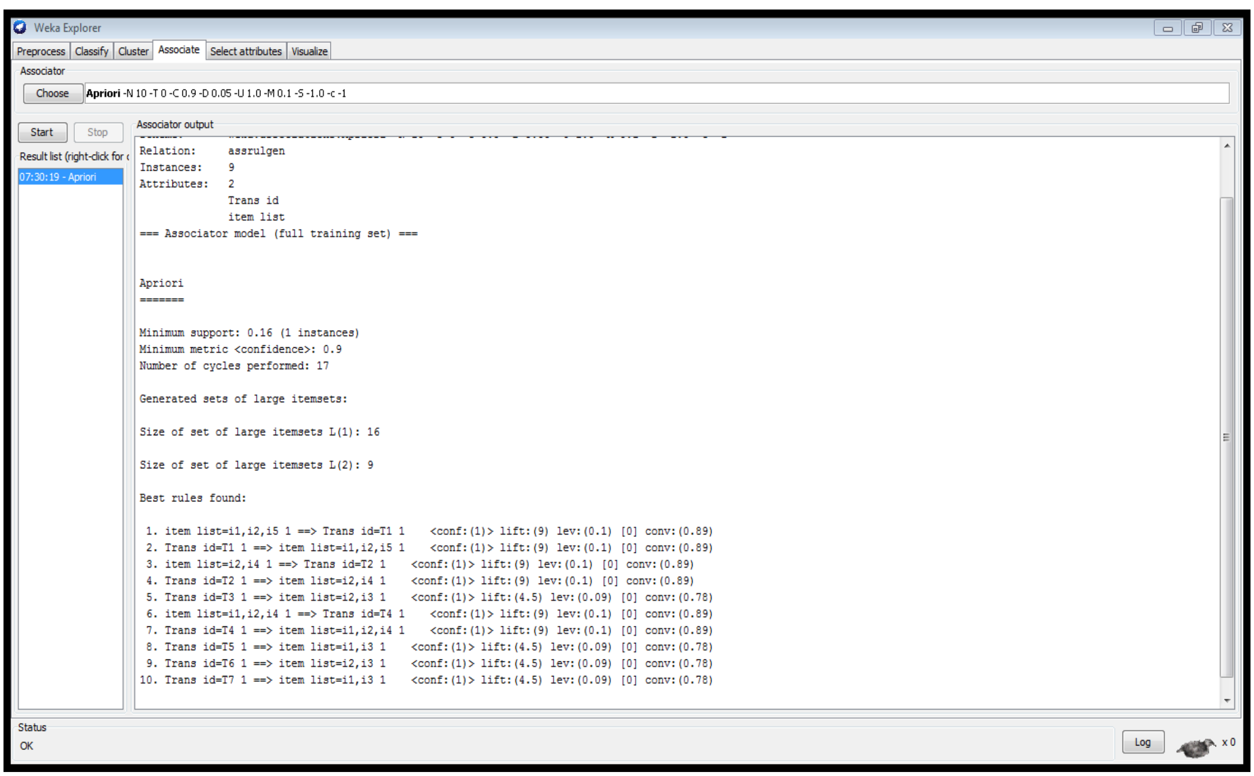



After loading, choose the associate tab in the weka explorer window.

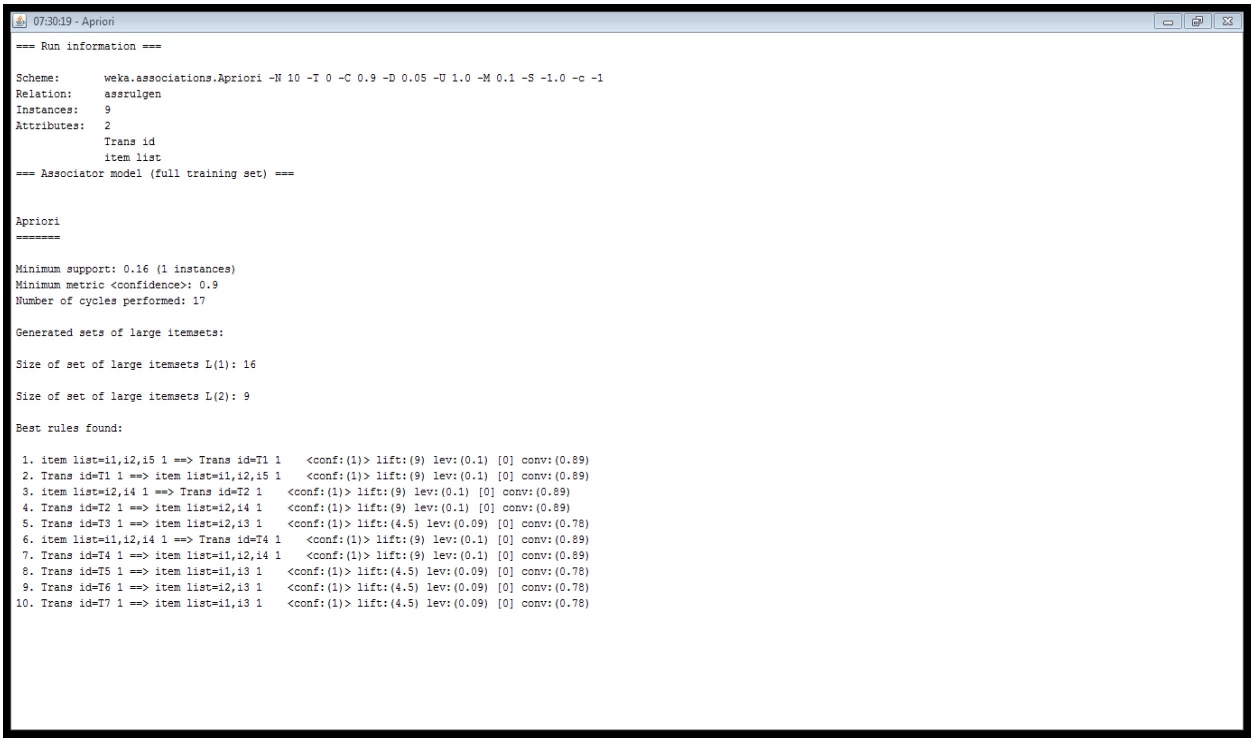
Under associate tab, click on button and select the apriori algorithm as shown below.




Select “use training set “ under the test options which is located at the left of the weka explorer window and the output is represented as shown below.



We can also view the output in a separate window by right clicking on the option in result list and clicking on “view in separate window” as shown below

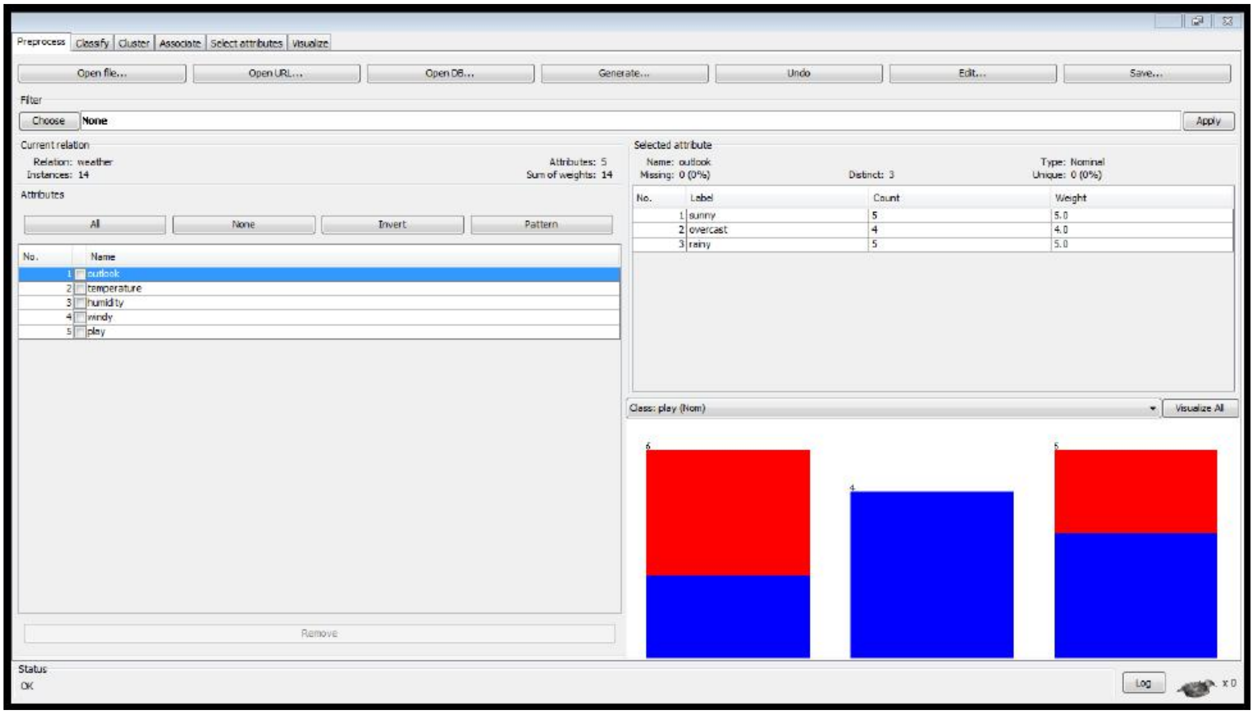


07:30:19 - Apriori

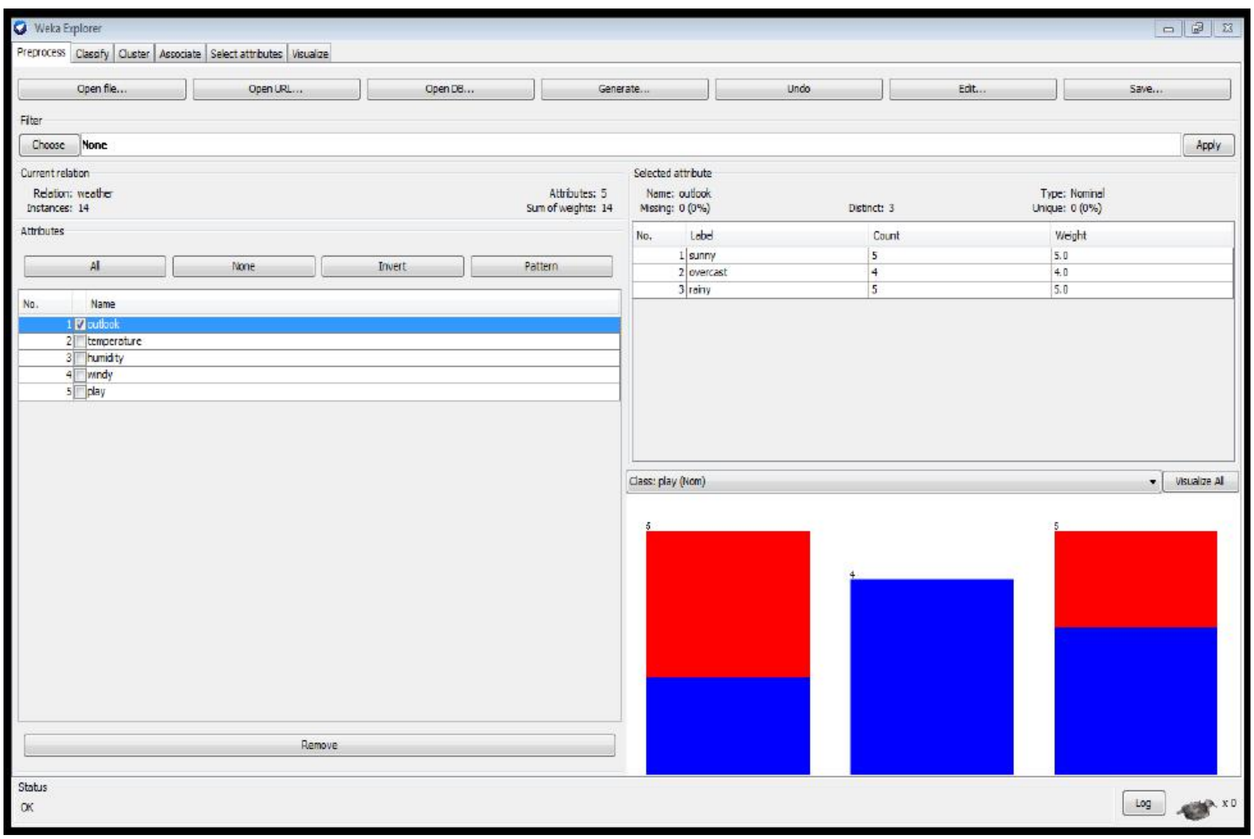
```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:     assrulgen
Instances:    9
Attributes:   2
              Trans id
              item list
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.16 (1 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

Size of set of large itemsets L(1): 16

Size of set of large itemsets L(2): 9

Best rules found:

 1. item list=i1,i2,i5 1 ==> Trans id=T1 1    <conf:(1)> lift:(9) lev:(0.1) [0] conv:(0.89)
 2. Trans id=T1 1 ==> item list=i1,i2,i5 1    <conf:(1)> lift:(9) lev:(0.1) [0] conv:(0.89)
 3. item list=i2,i4 1 ==> Trans id=T2 1    <conf:(1)> lift:(9) lev:(0.1) [0] conv:(0.89)
 4. Trans id=T2 1 ==> item list=i2,i4 1    <conf:(1)> lift:(9) lev:(0.1) [0] conv:(0.89)
 5. Trans id=T3 1 ==> item list=i2,i3 1    <conf:(1)> lift:(4.5) lev:(0.09) [0] conv:(0.78)
 6. item list=i1,i2,i4 1 ==> Trans id=T4 1    <conf:(1)> lift:(9) lev:(0.1) [0] conv:(0.89)
 7. Trans id=T4 1 ==> item list=i1,i2,i4 1    <conf:(1)> lift:(9) lev:(0.1) [0] conv:(0.89)
 8. Trans id=T5 1 ==> item list=i1,i3 1    <conf:(1)> lift:(4.5) lev:(0.09) [0] conv:(0.78)
 9. Trans id=T6 1 ==> item list=i2,i3 1    <conf:(1)> lift:(4.5) lev:(0.09) [0] conv:(0.78)
10. Trans id=T7 1 ==> item list=i1,i3 1    <conf:(1)> lift:(4.5) lev:(0.09) [0] conv:(0.78)
```

12. Data Discretization

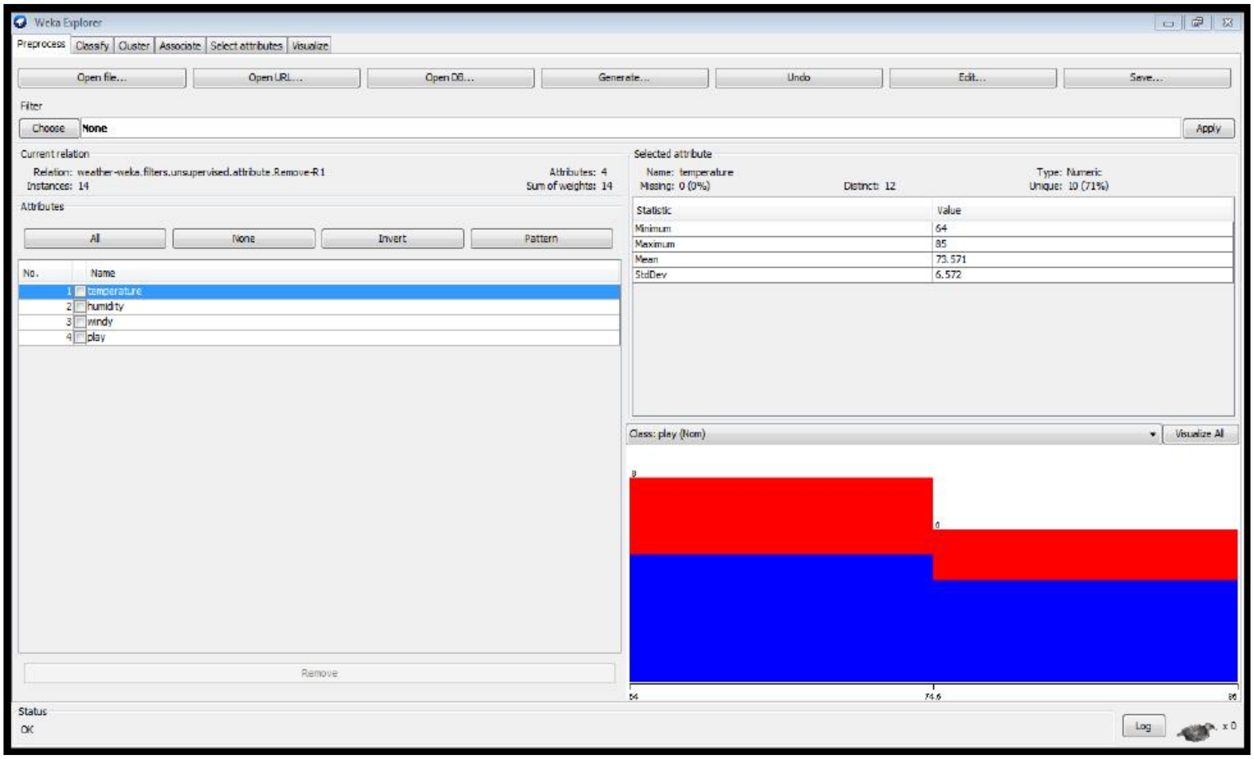
Open weka explorer and select weather.arff



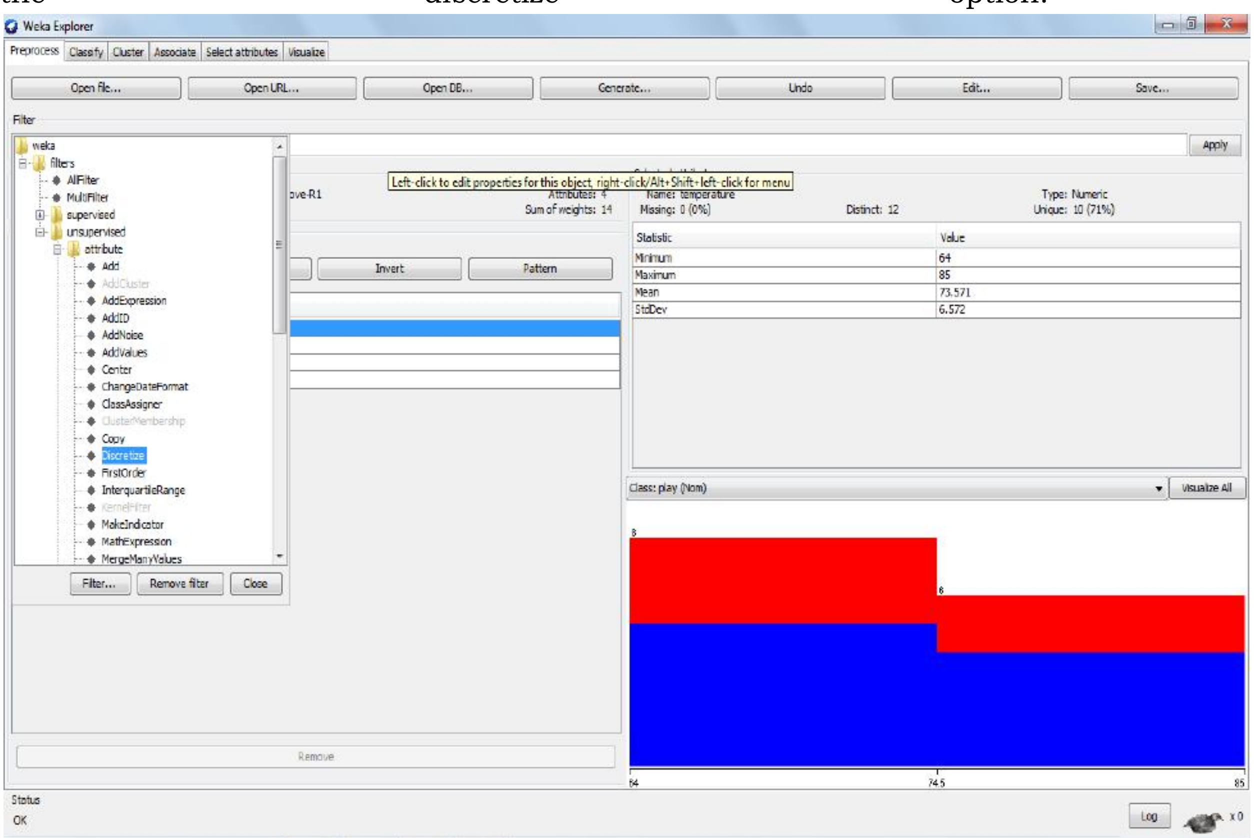

Select any attribute in the attributes section and click on remove button.



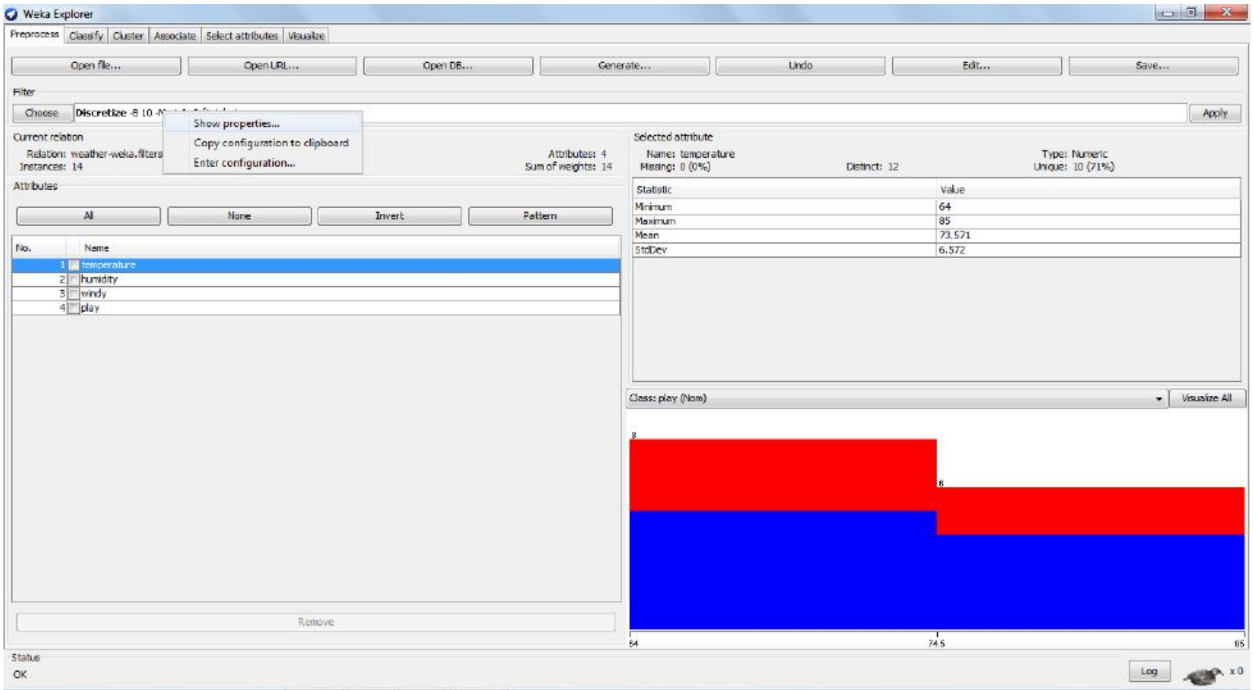
The below diagram shows the weather.arff after removing the attribute class.



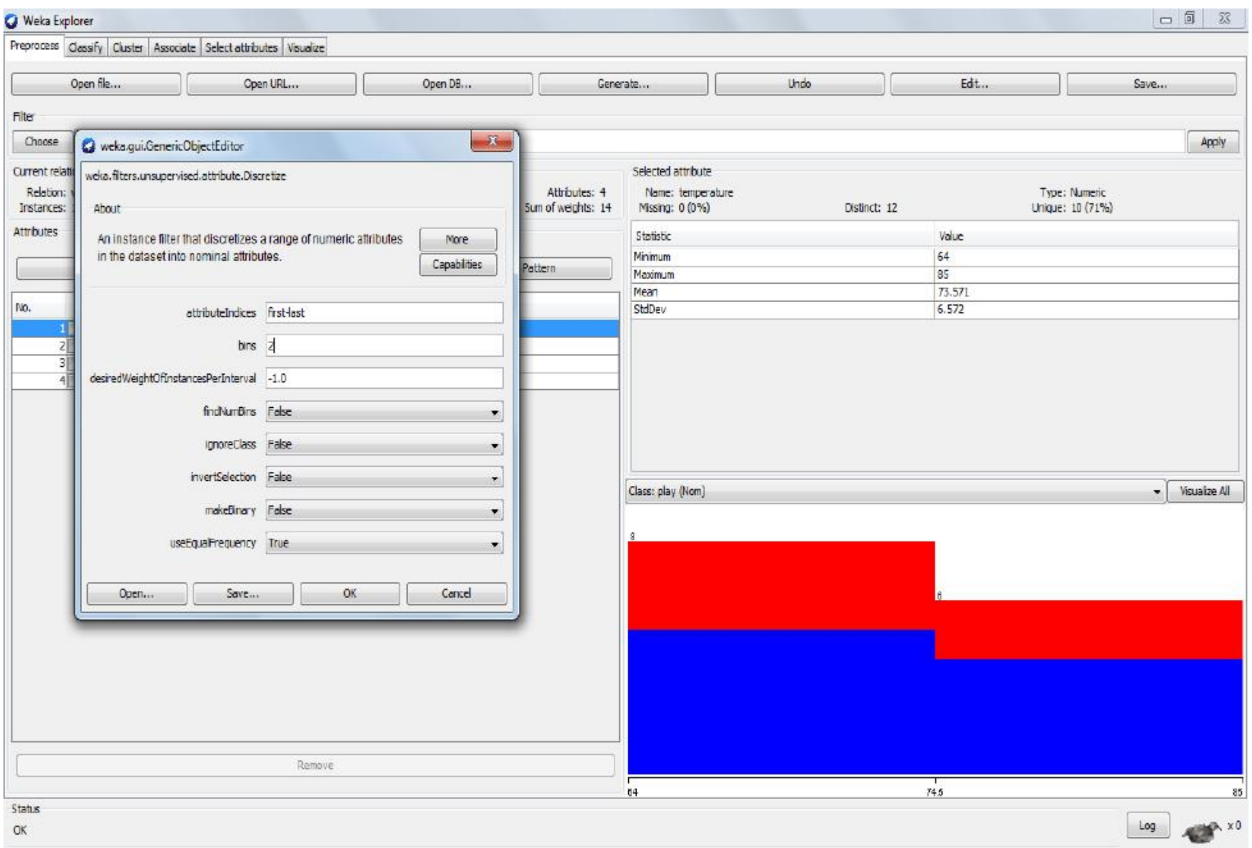

Now click on the “choose” button from the filter and expend the “unsupervised” option and select the “discretize” option.



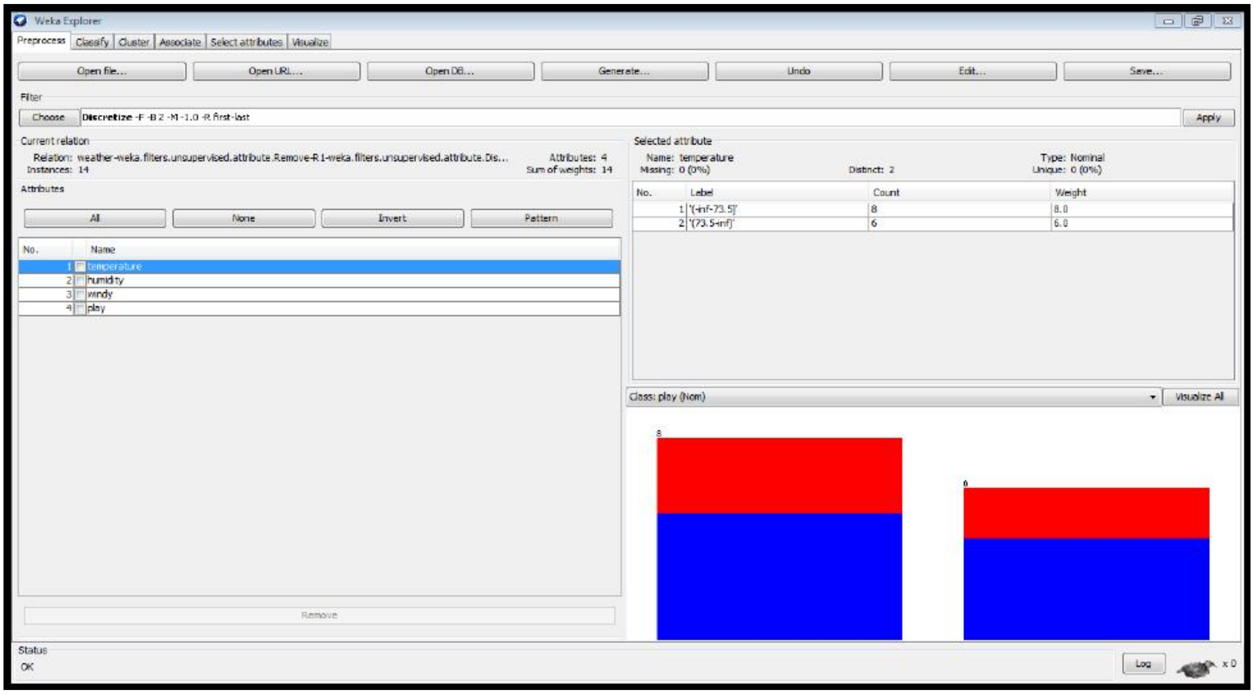
Now left click the object to edit the properties or right click and select show properties to edit the properties.



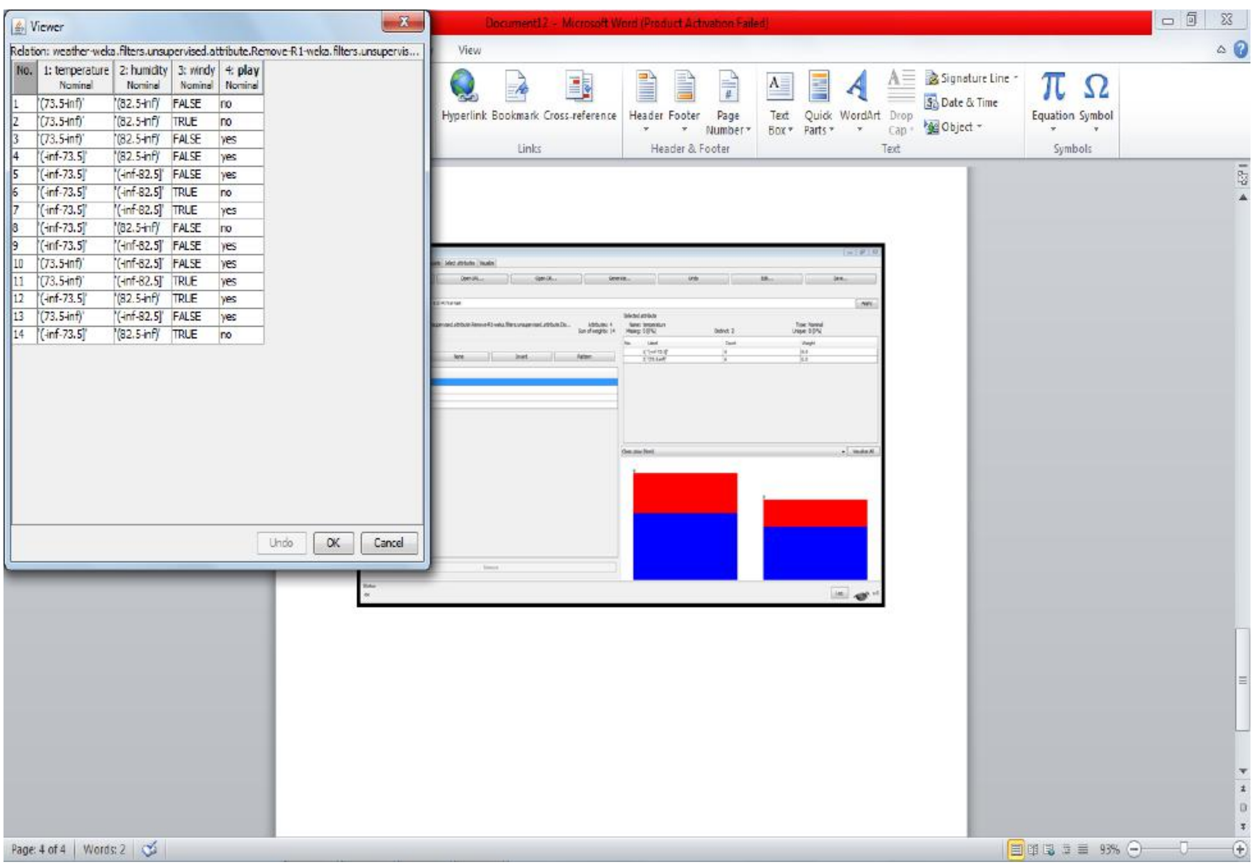
In the "generic ObjectEditor" change the bins value to either 2 or 3 or as our desire and make the "useEqualFrequency" option as "TRUE" and click on OK.



Now apply the prosperities by clicking on “apply” button in the filter where the discretize object contains bins and use EqualFrequency is set to TRUE

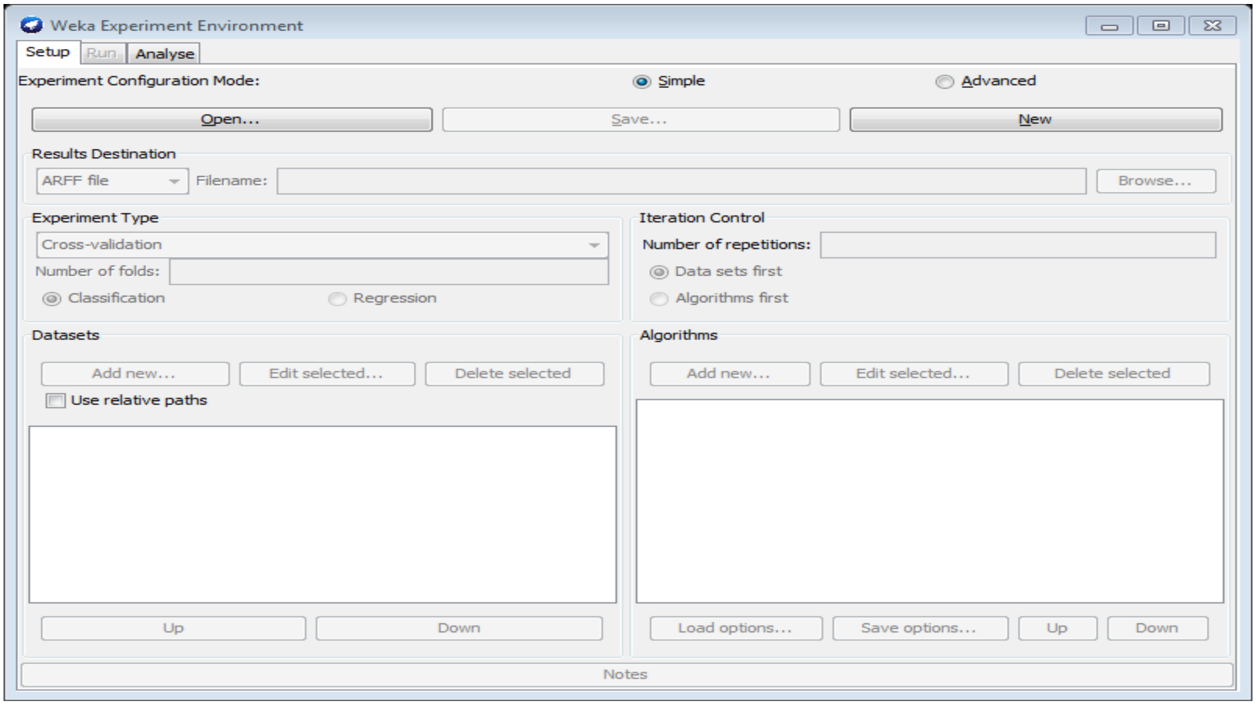



We can observe the change in the result in the visualize which is as shown in the figure by edit option.

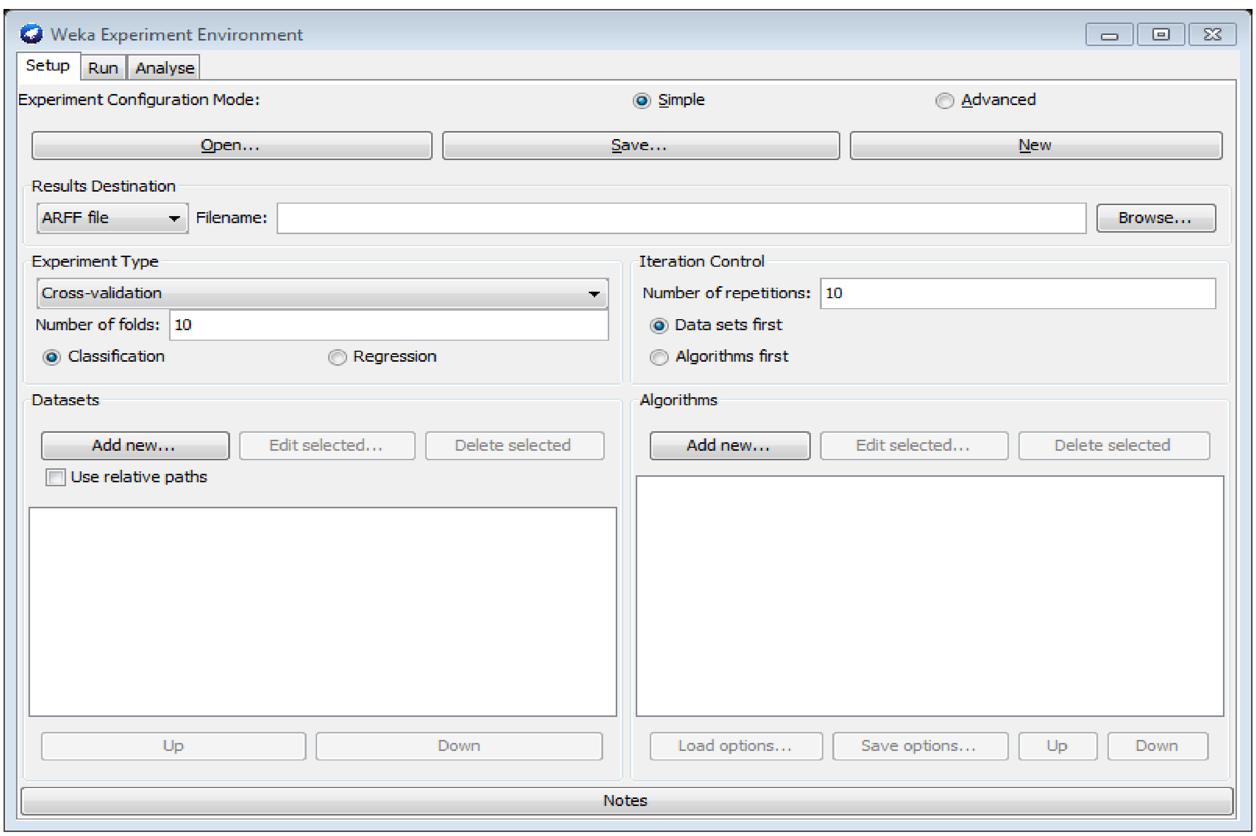


13. Weka's Experiment Environment usage in Simple Mode

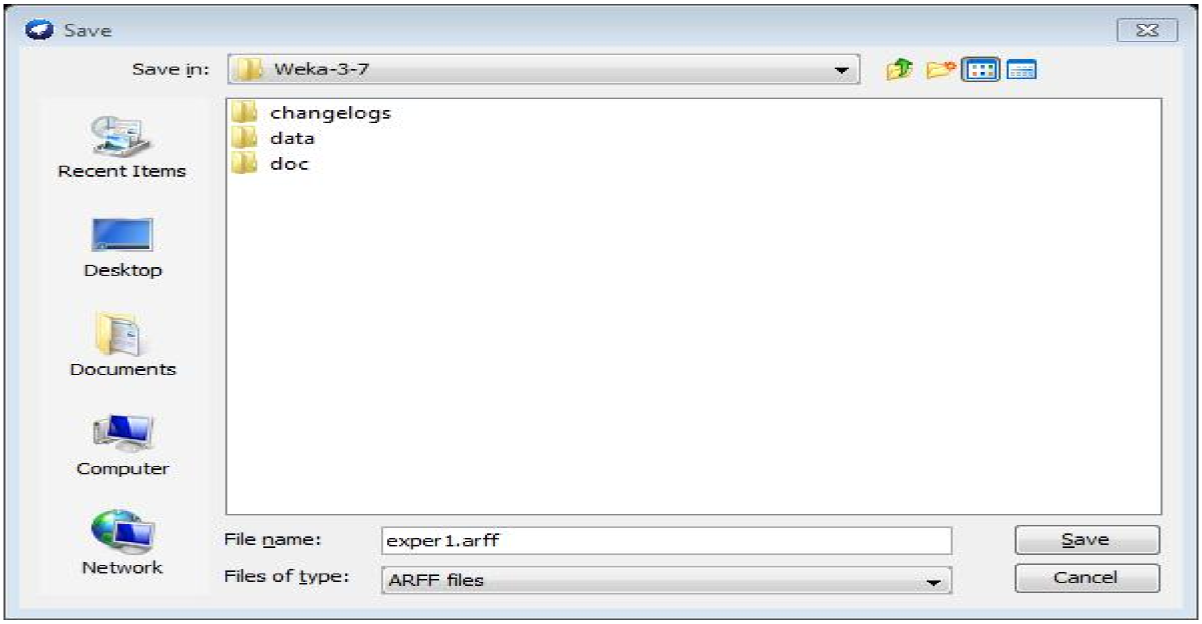
Open weka experiment environment.



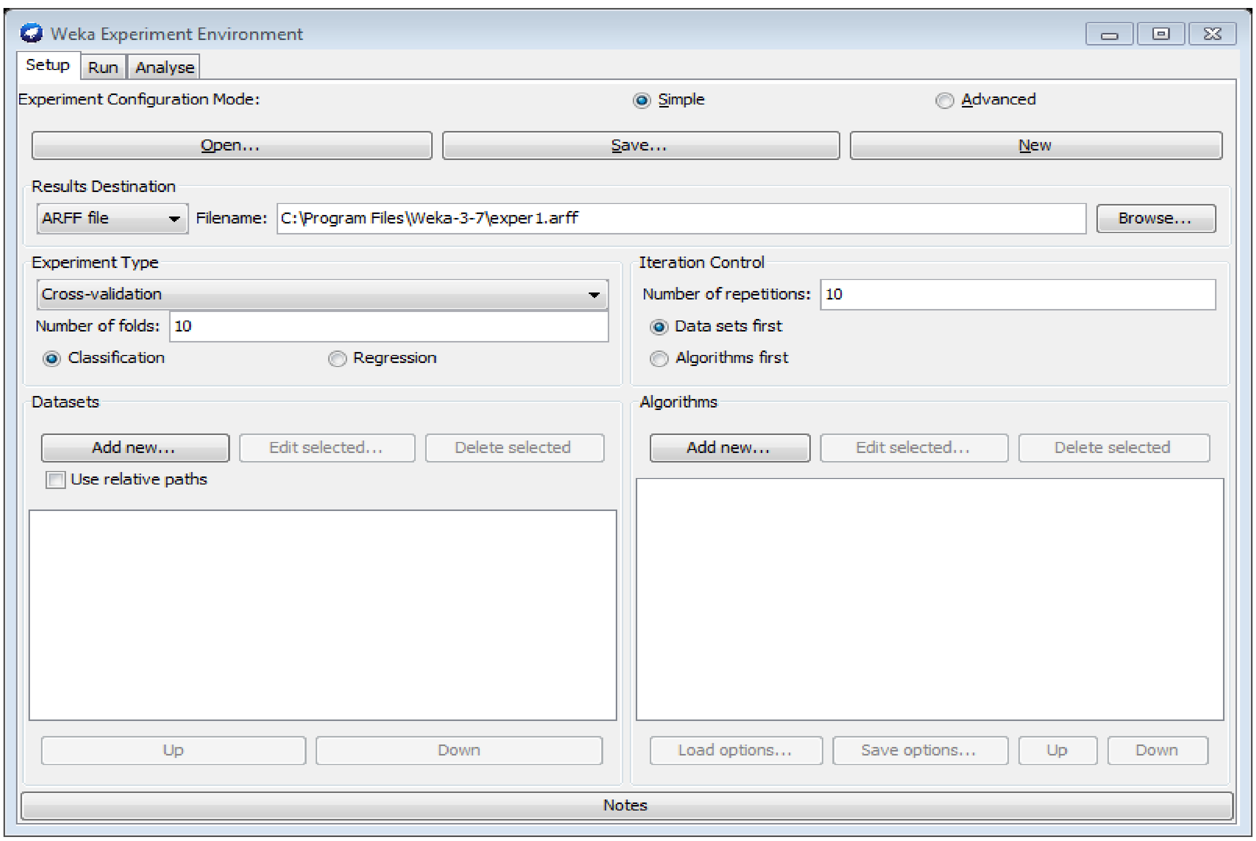

Click on new experiment, Click on results destination and select arff file.



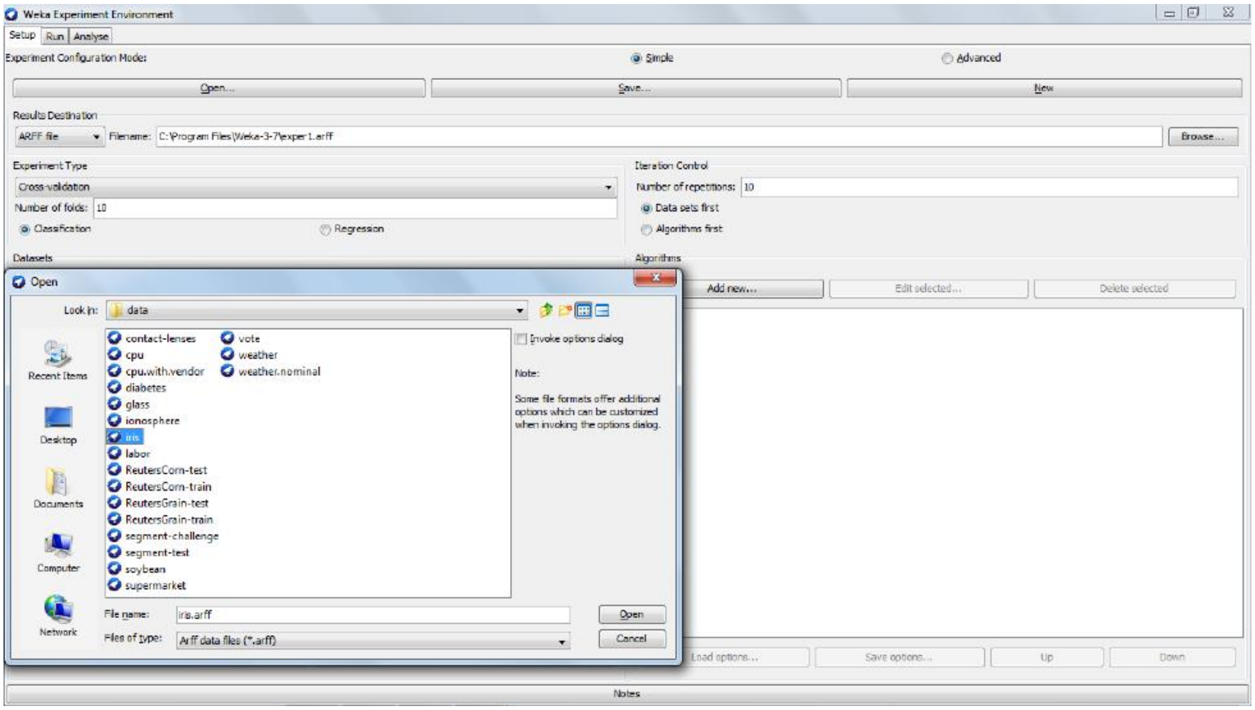
Click on browse and choose file name as exper1.arff




Click on experiment type choose cross validation with the default number of folds as 10 ,and click on use relative path check box.

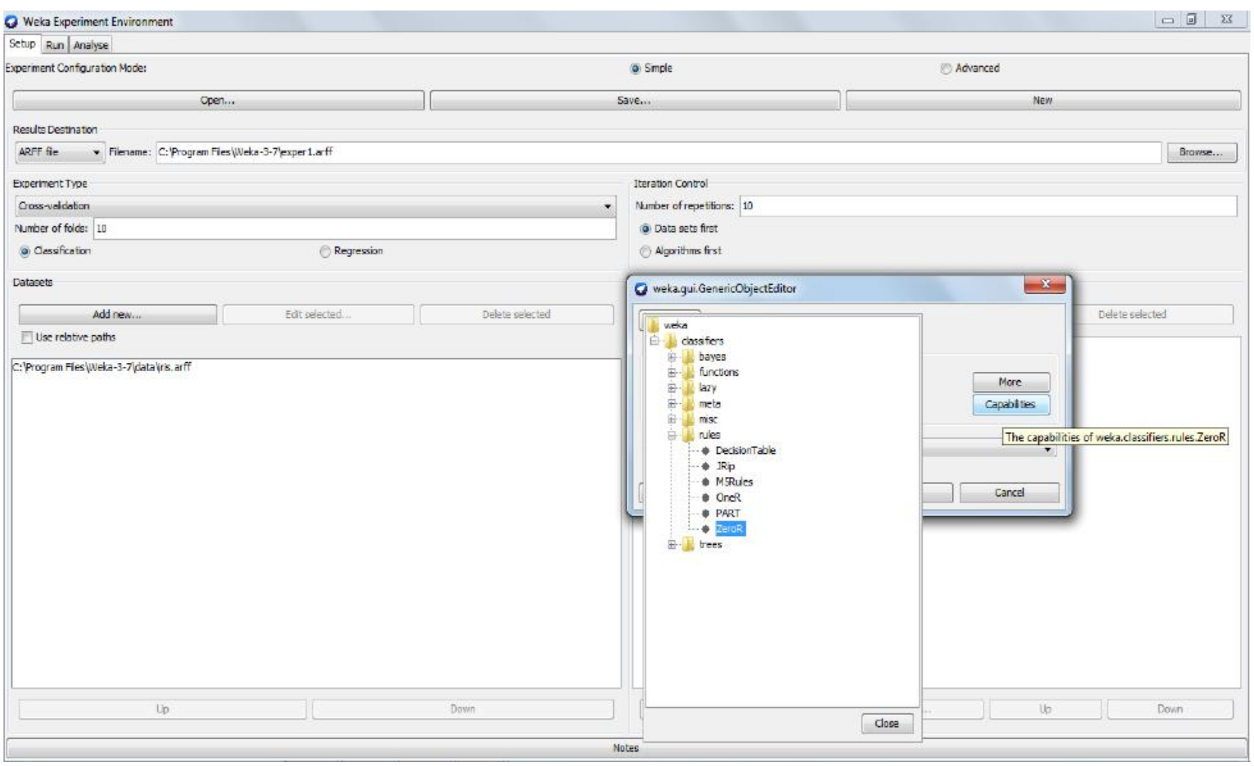


Click on add new button and select iris.arff, in iteration control tab by default there are 10 number of repetitions .either select data sets first or algorithms.

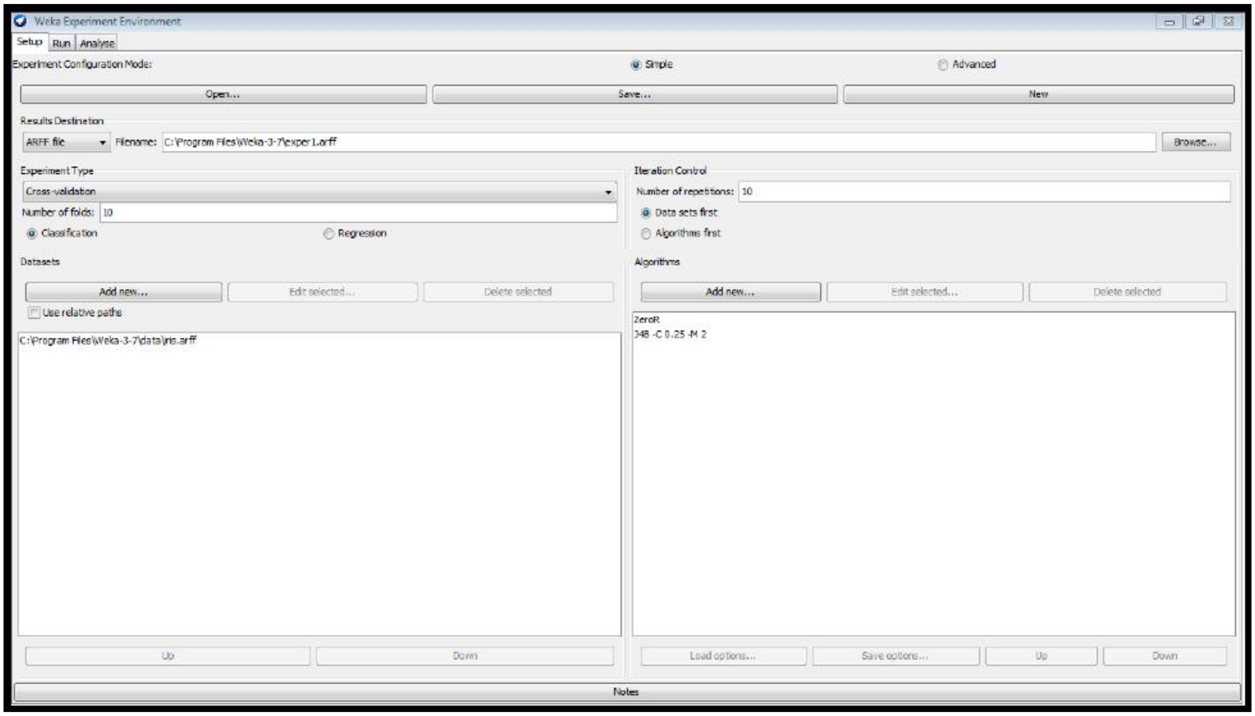



Click on add new button in algorithms and algorithms and choose an classifier, by default Zero-R classifier is selected, we can add many more classifiers using new buttons for example J48 classifier.

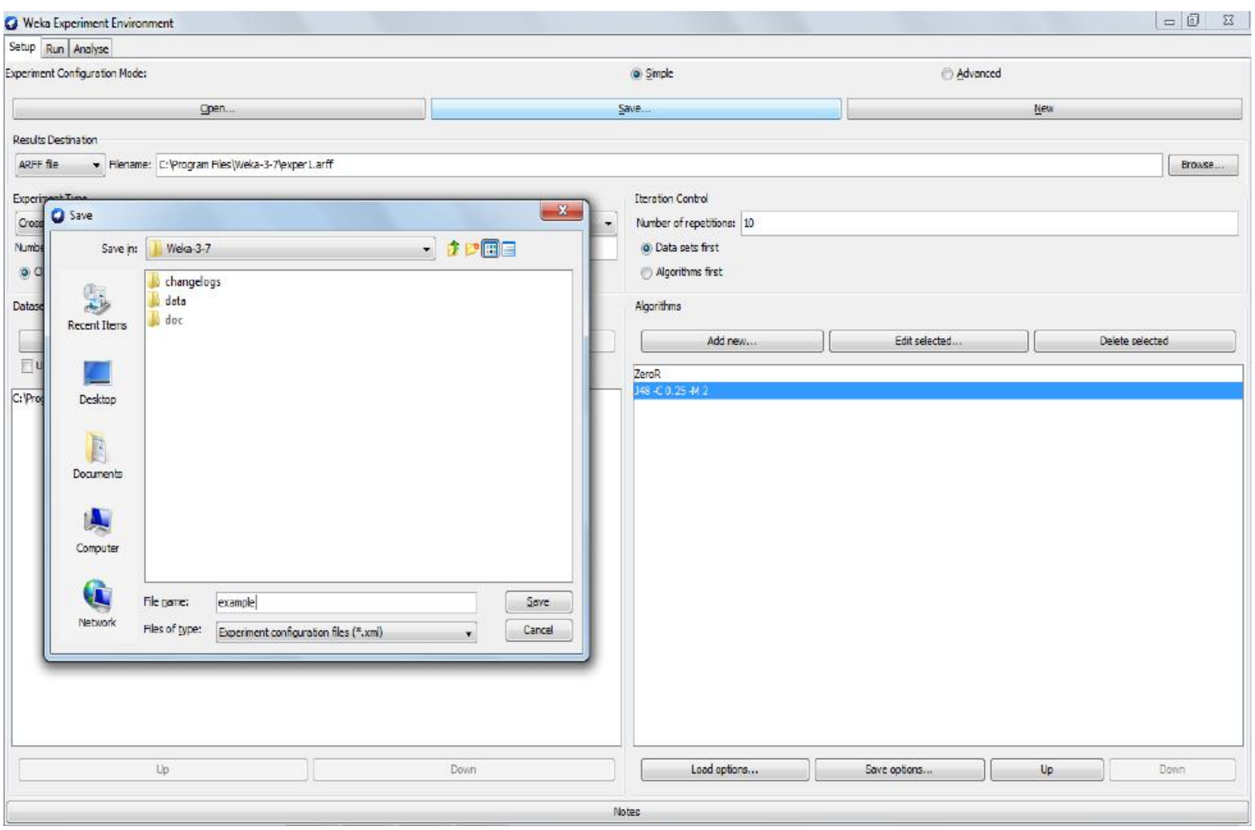
After selecting the classifier parameters click on ok to add it to the list of algorithms.



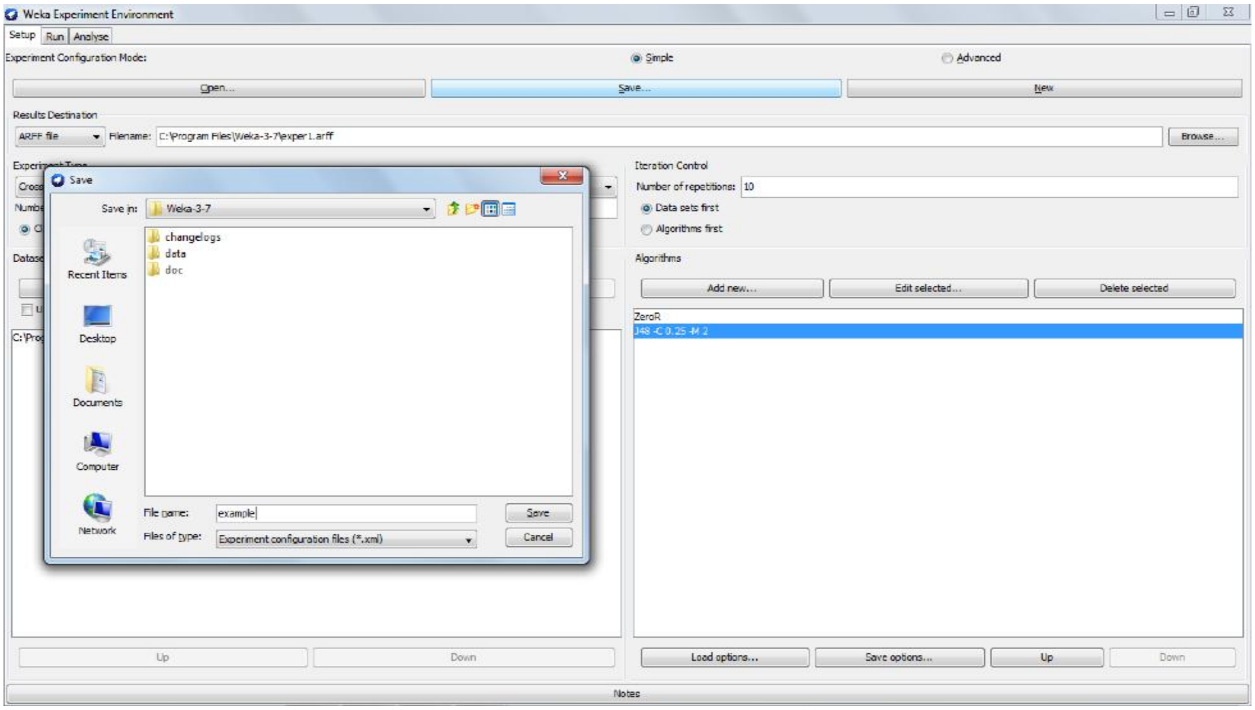
The algorithm will be stored in the list as shown in the below diagram.



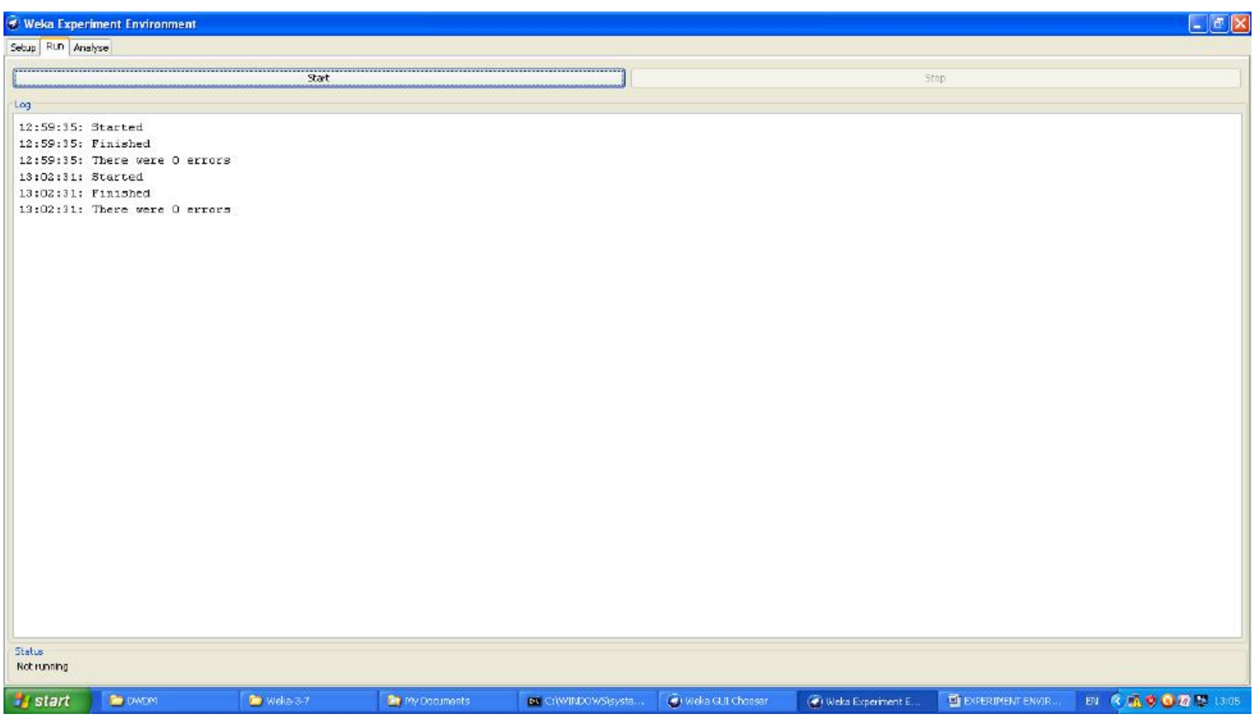

With the load and save options we can load and save setup of a select classifier and to XML.



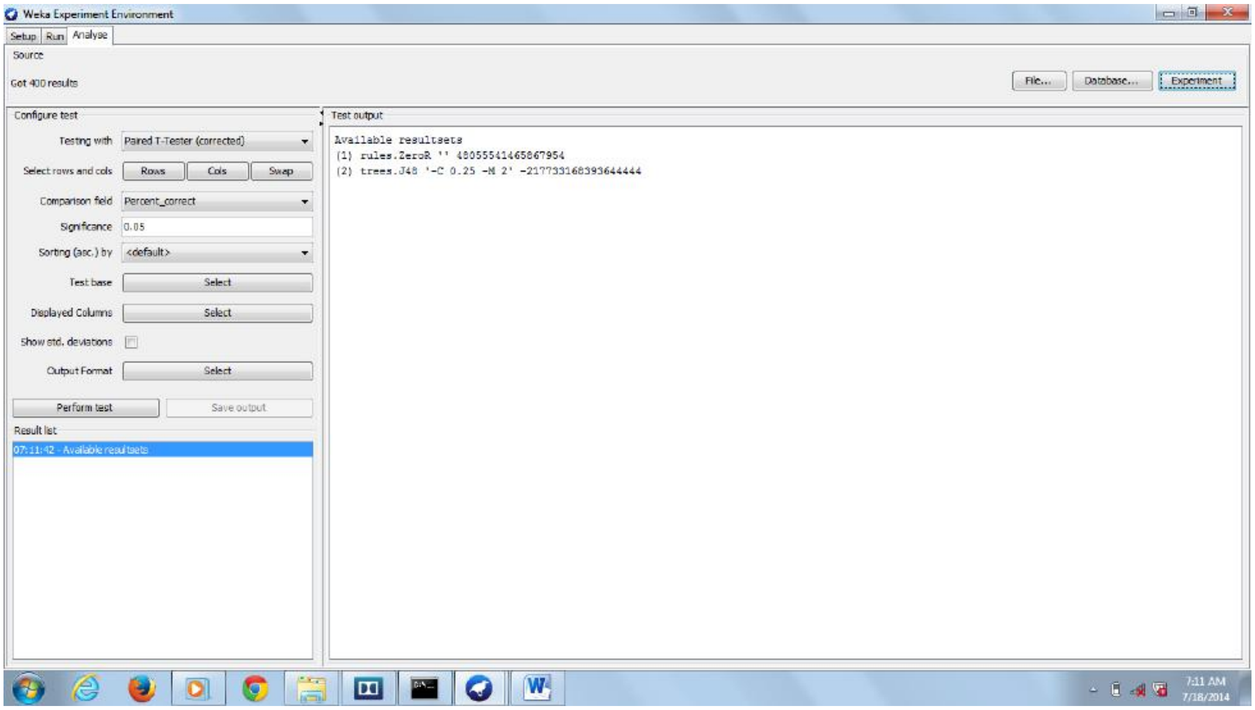
Save the current setup pf the experiment to a file by clicking on save at the top of the window, with extension.exp



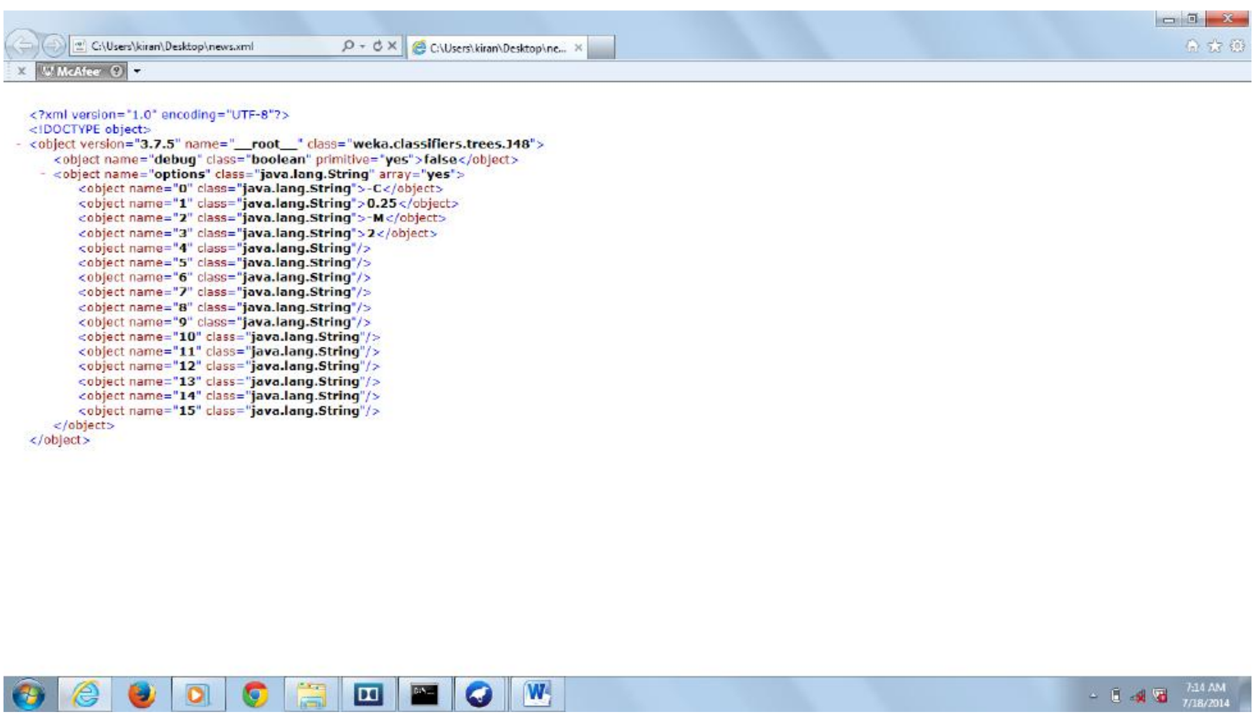
Run the current experiment by clicking on the RUN tab of the experiment environment window .click start to run the experiment. if the experiment was designed correctly, there will be ‘3’ messages in the log panel without errors.



Click on analyze tab at the top of the window and click on experiment tab and the output would be generated in the test output panel.

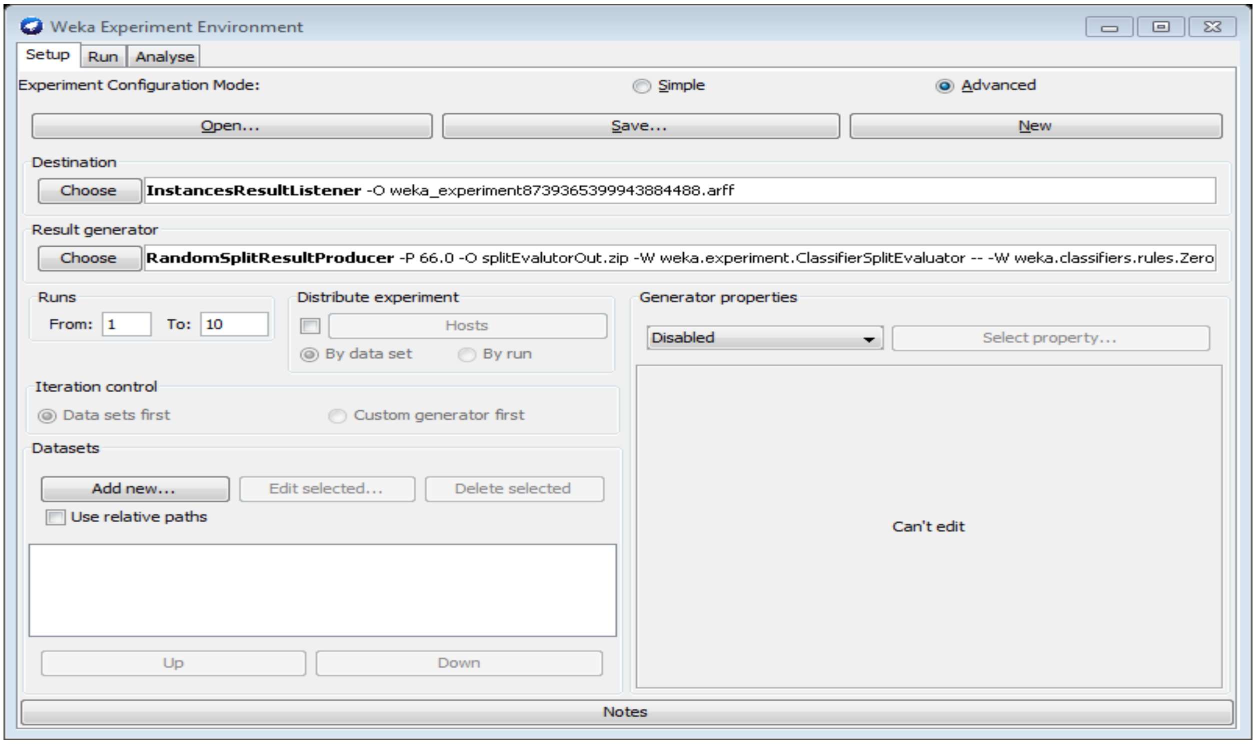


Check the output in an XML format open the sample.xml file.

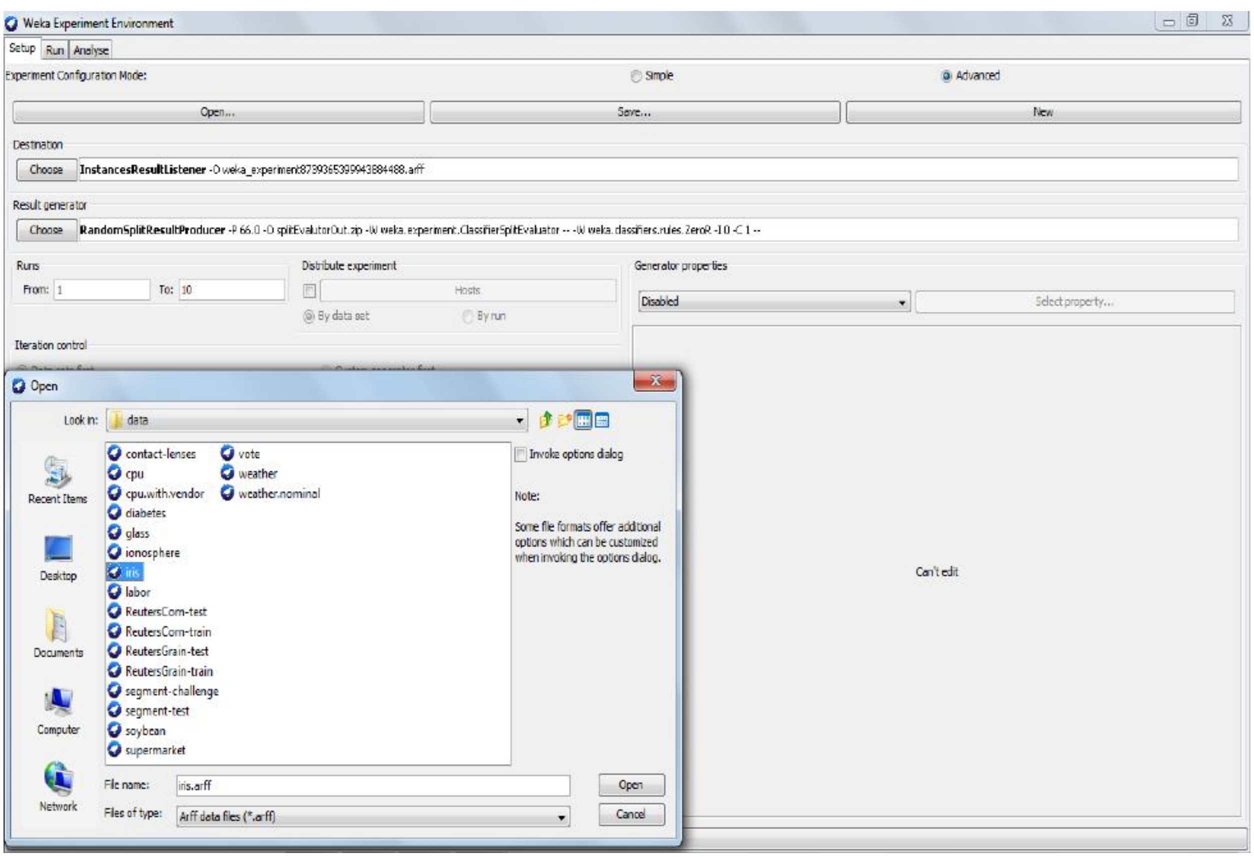


14. Weka experiment environment using advanced mode

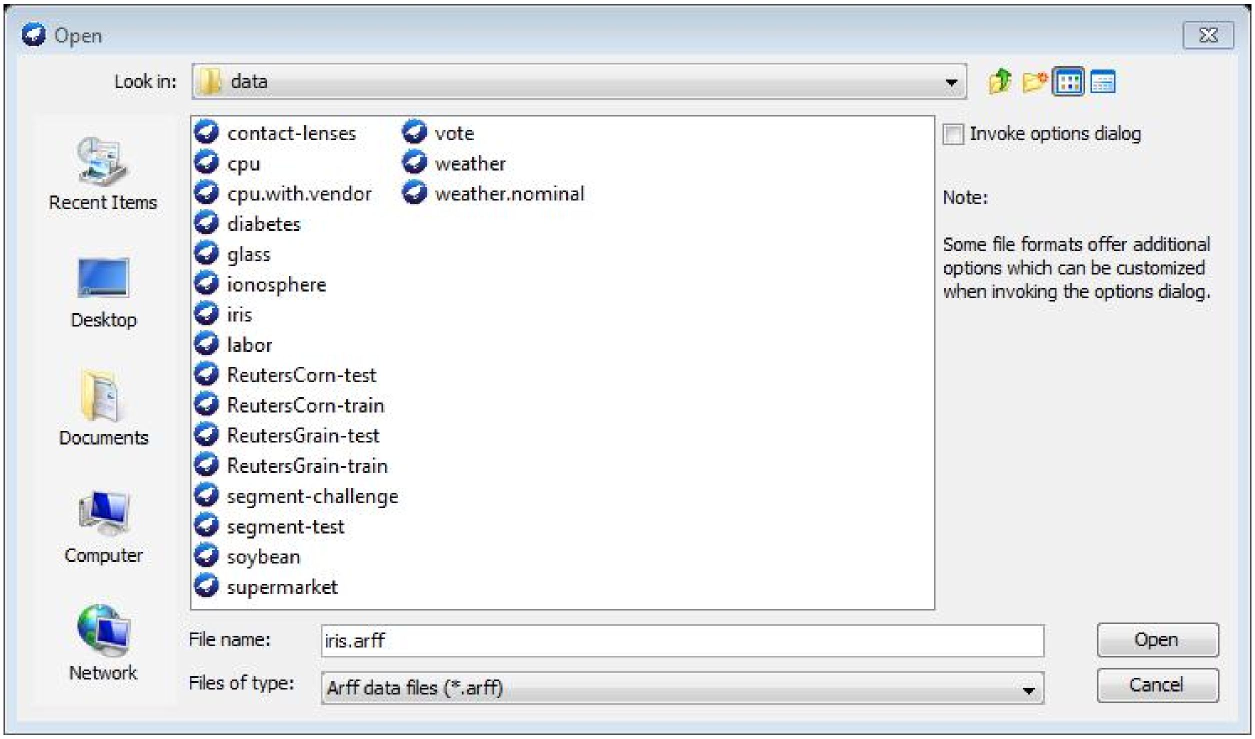
Open weka experiment environment. Click on Advanced Mode radio button.

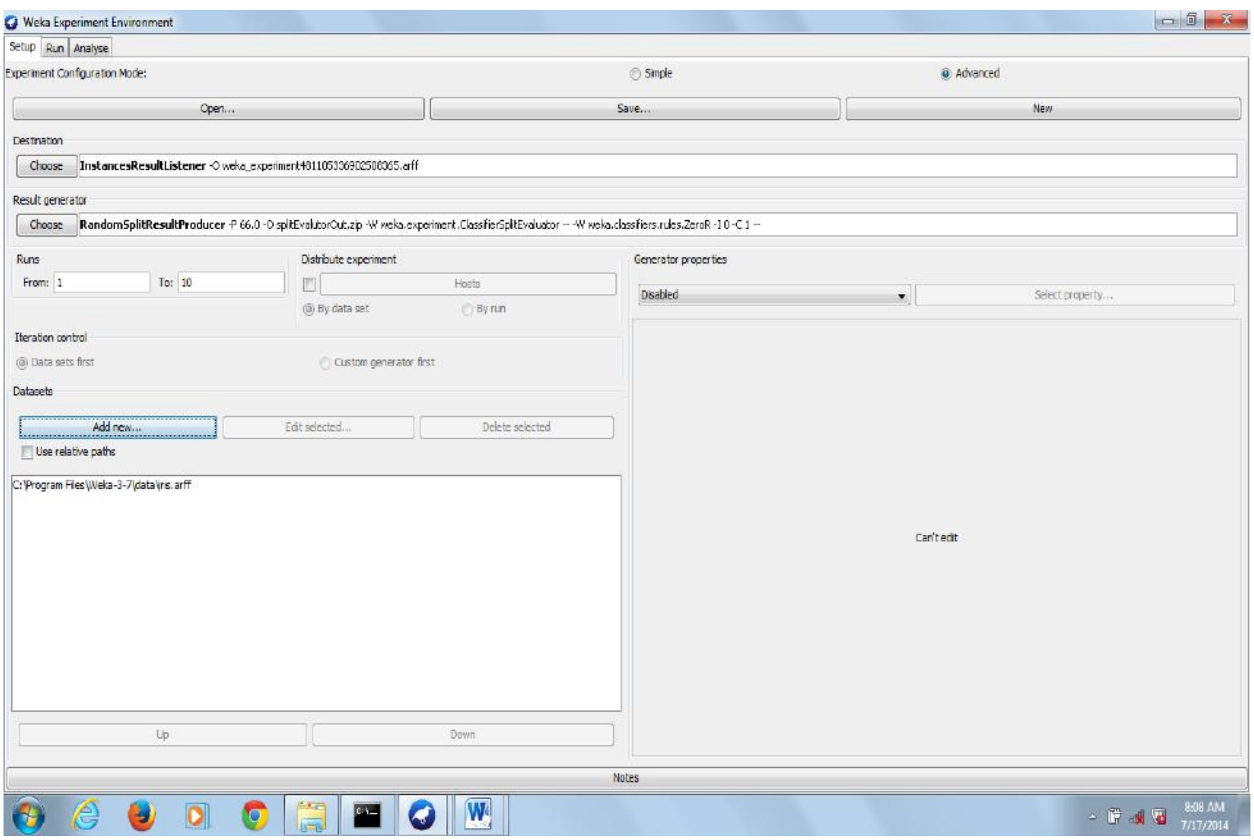



Select "userrelativepaths" in the datasets panel of the setup tab and click on add new to open a dialog window.

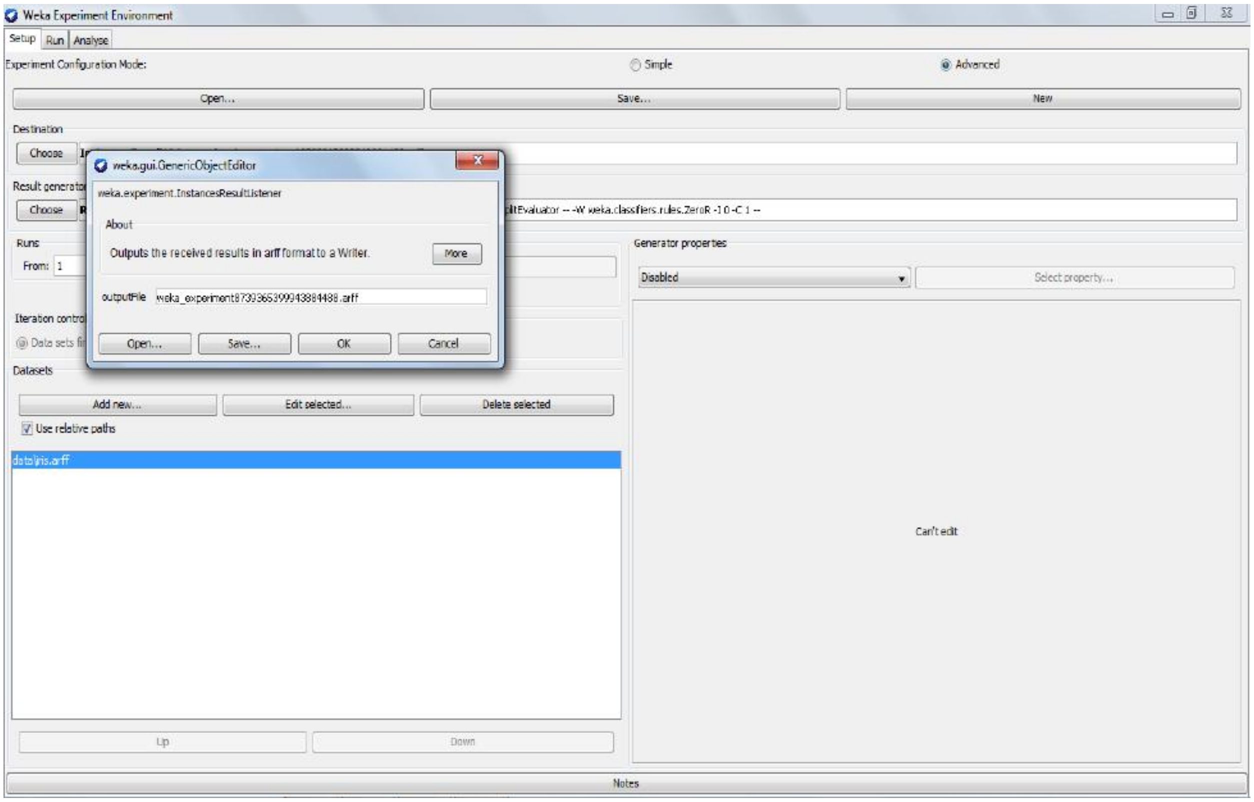


Double click on the data folder select iris.arff file.

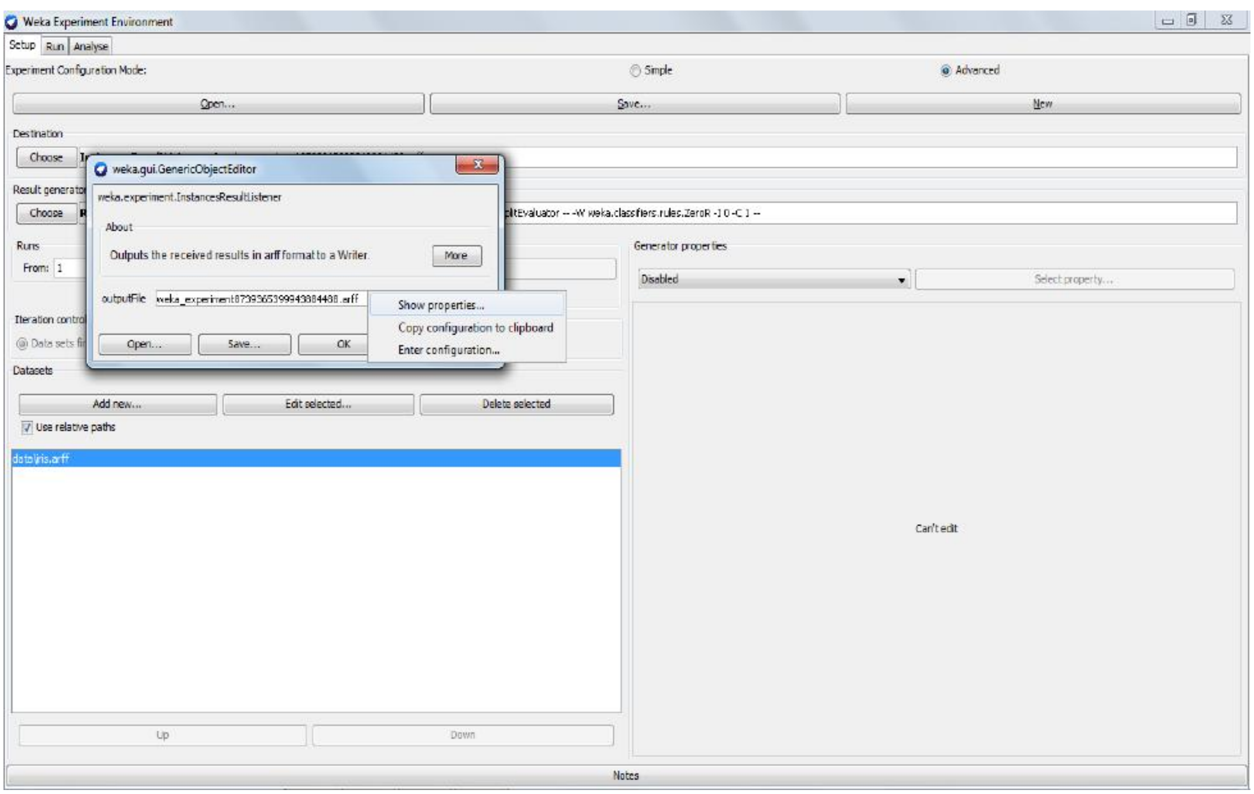


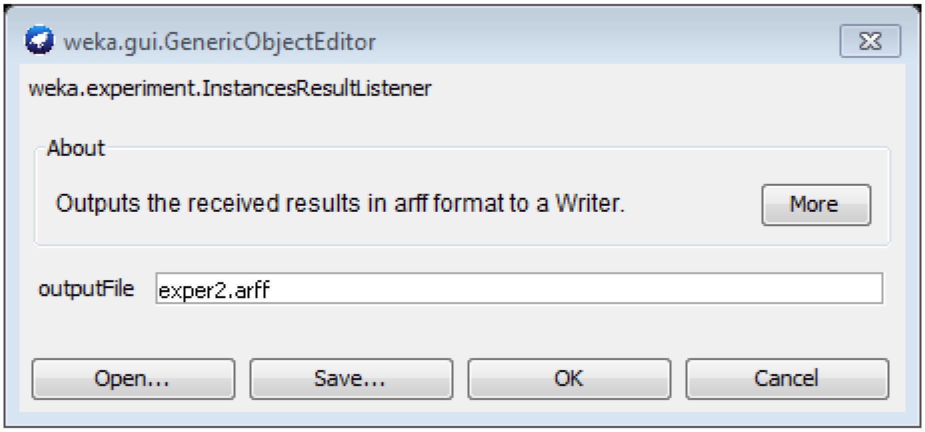
To identify a dataset to which the result are to be sent, click on the “instanceResultListener”entry in the destination panel.the output file parameter is near the bottom of the window beside the text output file.click on this parameter to display a file selection window.




The dataset name is displayed in the destination panel of the setup tab.

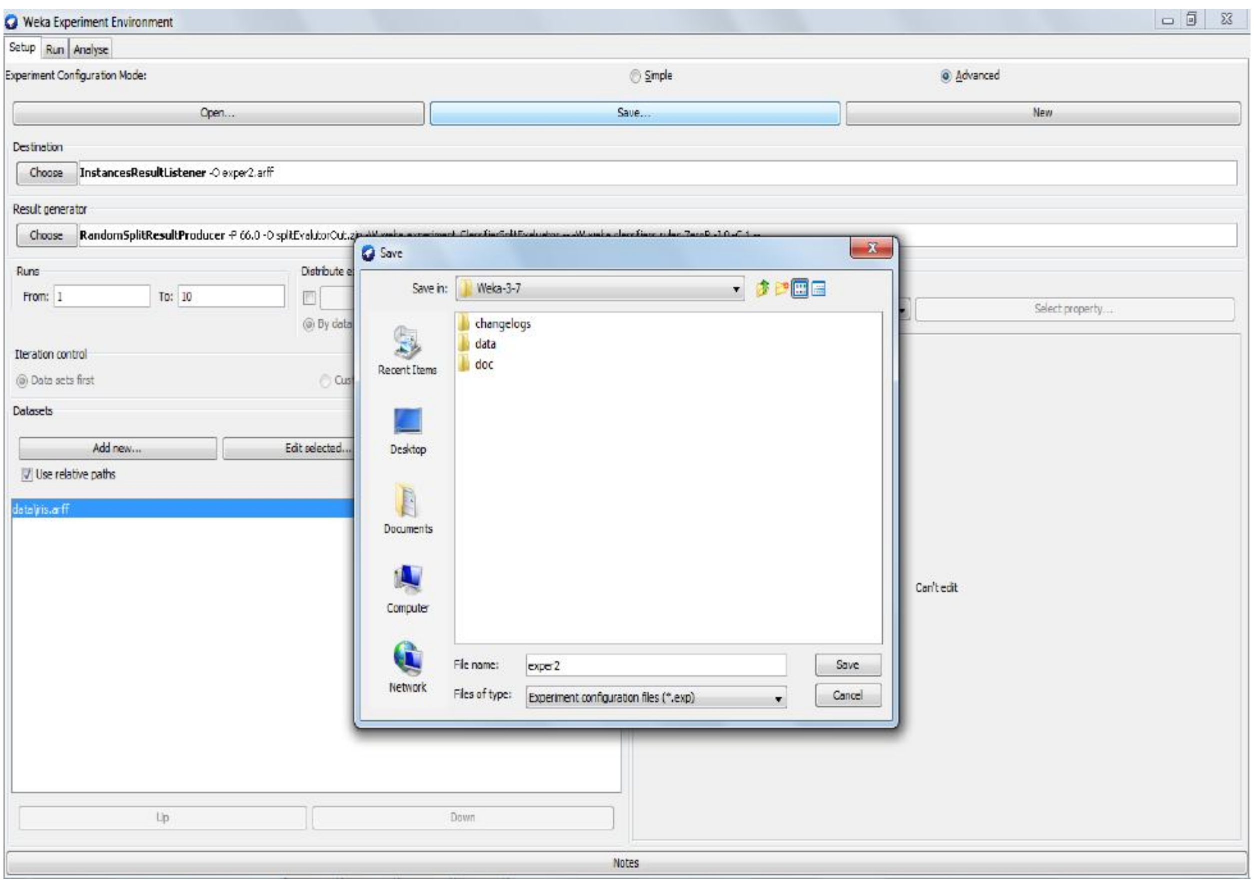


Type the name of the output file and click select.the file name is displayed in the output file panel.click on ok to close the window.




The dataset name is displayed in the destination panel of the setup tab

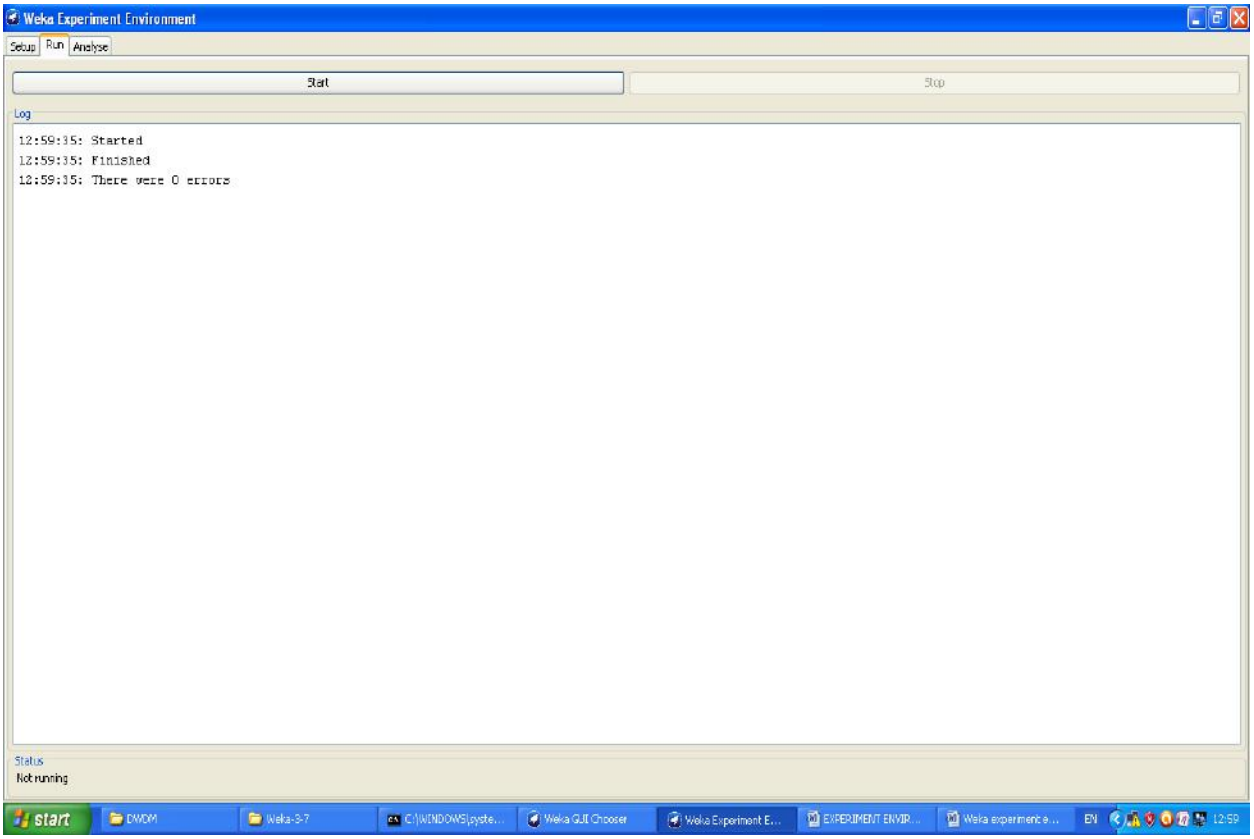
Select save at the top of the setup tab, type the dataset name with the extension exp for binary file for choose experiment configuration files for xml file type.



The experiment can be restored by selecting open in the setup tab and then selecting exper1.exp in the dialog box.

To run the current experiment ,click the run tab bat the top of the experiment window.

Click start to run the environment.




If the experiment was defined correctly, the three messages shown above will be displayed in the log panel.

**OUTPUT:**

@relation InstanceResultListener
@attribute Key_Dataset {iris}
@attribute Key_Run {1,2,3,4,5,6,7,8,9,10}
@attribute Key_Scheme {weka.classifiers.rules.ZeroR}
@attribute Key_Scheme_options {''}
@attribute Key_Scheme_version_ID {48055541465867954}
@attribute Date_time numeric
@attribute Number_of_training_instances numeric
@attribute Number_of_testing_instances numeric
@attribute Number_correct numeric
@attribute Number_incorrect numeric
@attribute Number_unclassified numeric
@attribute Percent_correct numeric
@attribute Percent_incorrect numeric
@attribute Percent_unclassified numeric
@attribute Kappa_statistic numeric
@attribute Mean_absolute_error numeric
@attribute Root_mean_squared_error numeric

@attribute Relative_absolute_error numeric
@attribute Root_relative_squared_error numeric
@attribute SF_prior_entropy numeric
@attribute SF_scheme_entropy numeric
@attribute SF_entropy_gain numeric
@attribute SF_mean_prior_entropy numeric
@attribute SF_mean_scheme_entropy numeric
@attribute SF_mean_entropy_gain numeric
@attribute KB_information numeric
@attribute KB_mean_information numeric
@attribute KB_relative_information numeric
@attribute True_positive_rate numeric
@attribute Num_true_positives numeric
@attribute False_positive_rate numeric
@attribute Num_false_positives numeric
@attribute True_negative_rate numeric
@attribute Num_true_negatives numeric
@attribute False_negative_rate numeric
@attribute Num_false_negatives numeric
@attribute IR_precision numeric
@attribute IR_recall numeric
@attribute F_measure numeric
@attribute Area_under_ROC numeric
@attribute Weighted_avg_true_positive_rate numeric
@attribute Weighted_avg_false_positive_rate numeric
@attribute Weighted_avg_true_negative_rate numeric
@attribute Weighted_avg_false_negative_rate numeric
@attribute Weighted_avg_IR_precision numeric
@attribute Weighted_avg_IR_recall numeric
@attribute Weighted_avg_F_measure numeric
@attribute Weighted_avg_area_under_ROC numeric
@attribute Unweighted_macro_avg_F_measure numeric
@attribute Unweighted_micro_avg_F_measure numeric
@attribute Elapsed_Time_training numeric
@attribute Elapsed_Time_testing numeric
@attribute UserCPU_Time_training numeric
@attribute UserCPU_Time_testing numeric
@attribute Serialized_Model_Size numeric
@attribute Serialized_Train_Set_Size numeric
@attribute Serialized_Test_Set_Size numeric
@attribute Coverage_of_Test_Cases_By_Regions numeric
@attribute Size_of_Predicted_Regions numeric
@attribute Summary string

```
@data

iris,1,weka.classifiers.rules.ZeroR,'',48055541465867954,20140716.0732,99,51,17,34,0,33.333333,66.666667,0,0,0.444444,0.471405,100,100,80.833088,80.833088,0,1.584963,1.584963,0,0,0,0,1,17,1,34,0,0,0,0,0.333333,1,0.5,0.5,0.333333,0.333333,0.666667,0.666667,0.111111,0.333333,0.166667,0.5,0.166667,0.333333,0,0,0,0,1157,8434,5122,100,100,?
iris,2,weka.classifiers.rules.ZeroR,'',48055541465867954,20140716.0732,99,51,17,34,0,33.333333,66.666667,0,0,0.444444,0.471405,100,100,80.833088,80.833088,0,1.584963,1.584963,0,0,0,0,1,17,1,34,0,0,0,0,0.333333,1,0.5,0.5,0.333333,0.333333,0.666667,0.666667,0.111111,0.333333,0.166667,0.5,0.166667,0.333333,0,0,0,0,1157,8434,5122,100,100,?
iris,3,weka.classifiers.rules.ZeroR,'',48055541465867954,20140716.0732,99,51,17,34,0,33.333333,66.666667,0,0,0.444444,0.471405,100,100,80.833088,80.833088,0,1.584963,1.584963,0,0,0,0,1,17,1,34,0,0,0,0,0.333333,1,0.5,0.5,0.333333,0.333333,0.666667,0.666667,0.111111,0.333333,0.166667,0.5,0.166667,0.333333,0,0,0,0,1157,8434,5122,100,100,?
iris,4,weka.classifiers.rules.ZeroR,'',48055541465867954,20140716.0732,99,51,17,34,0,33.333333,66.666667,0,0,0.444444,0.471405,100,100,80.833088,80.833088,0,1.584963,1.584963,0,0,0,0,1,17,1,34,0,0,0,0,0.333333,1,0.5,0.5,0.333333,0.333333,0.666667,0.666667,0.111111,0.333333,0.166667,0.5,0.166667,0.333333,0,0,0,0,1157,8434,5122,100,100,?
iris,5,weka.classifiers.rules.ZeroR,'',48055541465867954,20140716.0732,99,51,17,34,0,33.333333,66.666667,0,0,0.444444,0.471405,100,100,80.833088,80.833088,0,1.584963,1.584963,0,0,0,0,1,17,1,34,0,0,0,0,0.333333,1,0.5,0.5,0.333333,0.333333,0.666667,0.666667,0.111111,0.333333,0.166667,0.5,0.166667,0.333333,0,0,0,0,1157,8434,5122,100,100,?
iris,6,weka.classifiers.rules.ZeroR,'',48055541465867954,20140716.0732,99,51,17,34,0,33.333333,66.666667,0,0,0.444444,0.471405,100,100,80.833088,80.833088,0,1.584963,1.584963,0,0,0,0,1,17,1,34,0,0,0,0,0.333333,1,0.5,0.5,0.333333,0.333333,0.666667,0.666667,0.111111,0.333333,0.166667,0.5,0.166667,0.333333,0,0,0,0,1157,8434,5122,100,100,?
iris,7,weka.classifiers.rules.ZeroR,'',48055541465867954,20140716.0732,99,51,17,34,0,33.333333,66.666667,0,0,0.444444,0.471405,100,100,80.833088,80.833088,0,1.584963,1.584963,0,0,0,0,1,17,1,34,0,0,0,0,0.333333,1,0.5,0.5,0.333333,0.333333,0.666667,0.666667,0.111111,0.333333,0.166667,0.5,0.166667,0.333333,0,0,0,0,1157,8434,5122,100,100,?
iris,8,weka.classifiers.rules.ZeroR,'',48055541465867954,20140716.0732,99,51,17,34,0,33.333333,66.666667,0,0,0.444444,0.471405,100,100,80.833088,80.833088,0,1.584963,1.584963,0,0,0,0,1,17,1,34,0,0,0,0,0.333333,1,0.5,0.5,0.333333,0.333333,0.666667,0.666667,0.111111,0.333333,0.166667,0.5,0.166667,0.333333,0,0,0,0,1157,8434,5122,100,100,?
iris,9,weka.classifiers.rules.ZeroR,'',48055541465867954,20140716.0732,99,51,17,34,0,33.333333,66.666667,0,0,0.444444,0.471405,100,100,80.833088,80.833088,0,1.584963,1.584963,0,0,0,0,1,17,1,34,0,0,0,0,0.333333,1,0.5,0.5,0.333333,0.333333,0.666667,0.666667,0.111111,0.333333,0.166667,0.5,0.166667,0.333333,0,0,0,0,1157,8434,5122,100,100,?
iris,10,weka.classifiers.rules.ZeroR,'',48055541465867954,20140716.0732,99,51,17,34,0,33.333333,66.666667,0,0,0.444444,0.471405,100,100,80.833088,80.833088,0,1.584963,1.584963,0,0,0,0,1,17,1,34,0,0,0,0,0.333333,1,0.5,0.5,0.333333,0.333333,0.666667,0.666667,0.111111,0.333333,0.166667,0.5,0.166667,0.333333,0,0,0,0,1157,8434,5122,100,100,?
```

15. Perform Cross Validation of different algorithms of a DM functionality

Setting up a flow to load an ARFF file (batch mode) and perform a crossvalidation using J48.

• Click on the DataSources tab and choose ArffLoader from the toolbar (the mouse pointer will change to a cross hairs).

• Next place the ArffLoader component on the layout area by clicking somewhere on the layout (a copy of the ArffLoader icon will appear on the layout area).

• Next specify an ARFF file to load by first right clicking the mouse over the ArffLoader icon on the layout. A pop-up menu will appear. Select Configure under Edit in the list from this menu and browse to the location of your ARFF file.

• Next click the Evaluation tab at the top of the window and choose the ClassAssigner (allows you to choose which column to be the class) component from the toolbar. Place this on the layout.

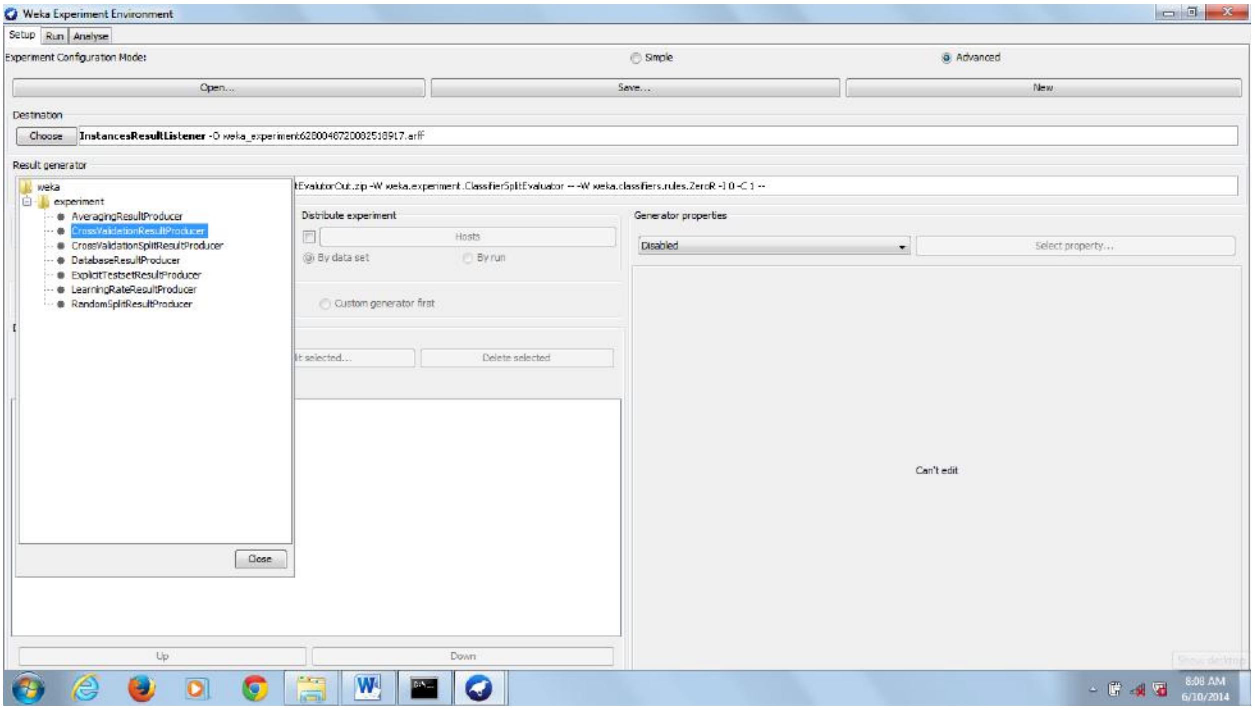
• Now connect the ArffLoader to the ClassAssigner: first right click over the ArffLoader and select the dataSet under Connections in the menu. A rubber band line will appear. Move the mouse over the ClassAssigner component and left click - a red line labeled dataSet will connect the two components.

• Next right click over the ClassAssigner and choose Configure from the menu. This will pop up a window from which you can specify which column is the class in your data (last is the default).

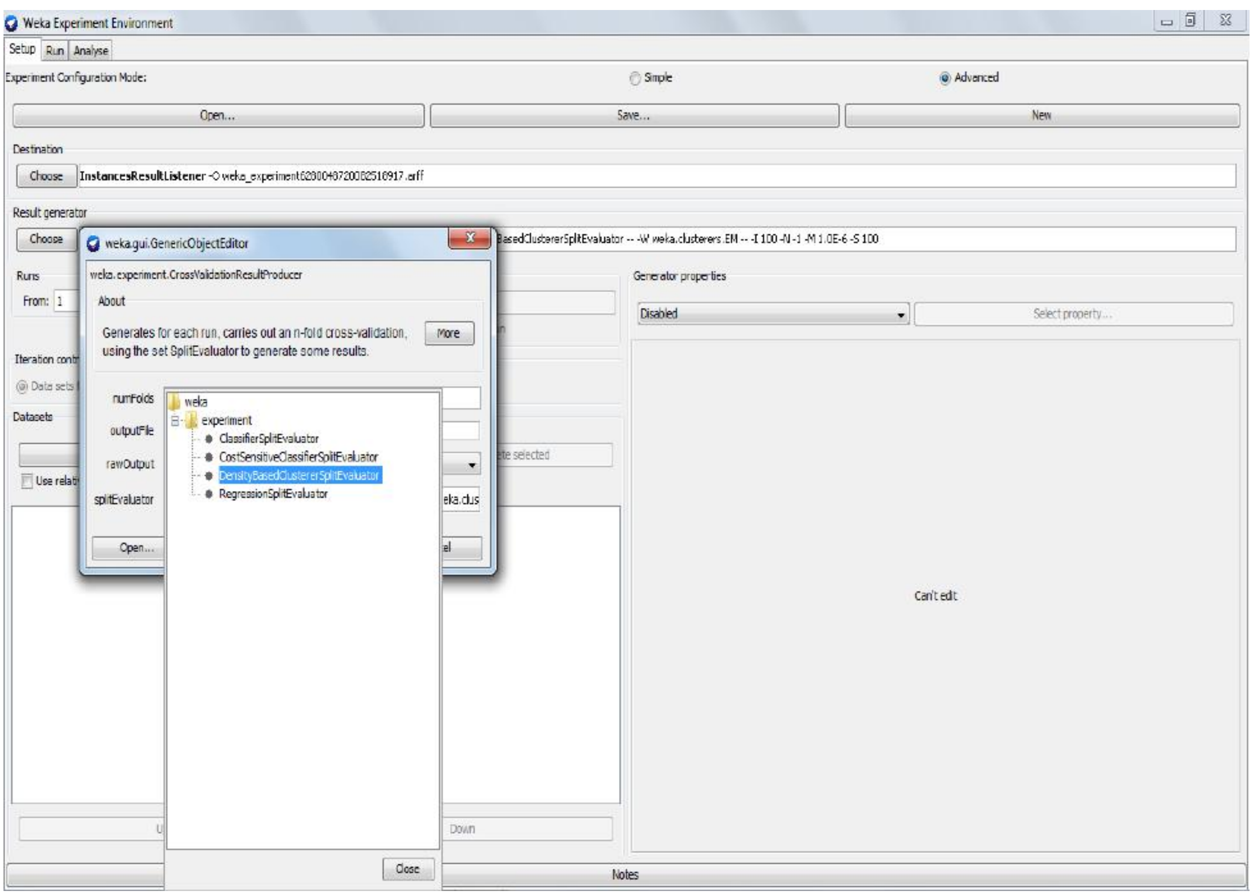
• Next grab a CrossValidationFoldMaker component from the Evaluation toolbar and place it on the layout. Connect the ClassAssigner to the CrossValidationFoldMaker by right clicking over ClassAssigner and selecting dataSet from under Connections in the menu.

• Next click on the Classifiers tab at the top of the window and scroll along the toolbar until you reach the J48 component in the trees section. Place a J48 component on the layout.

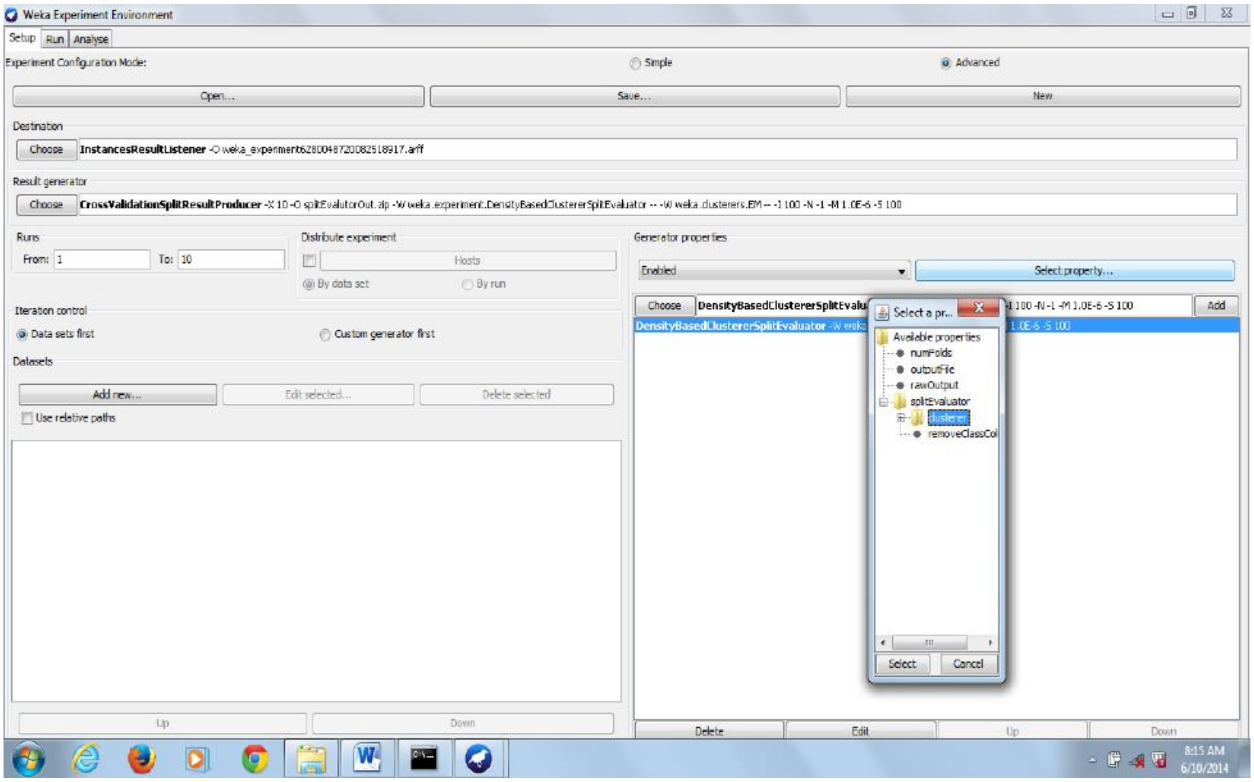
Choose”crossValidationResultProducer” from the result generator panel.



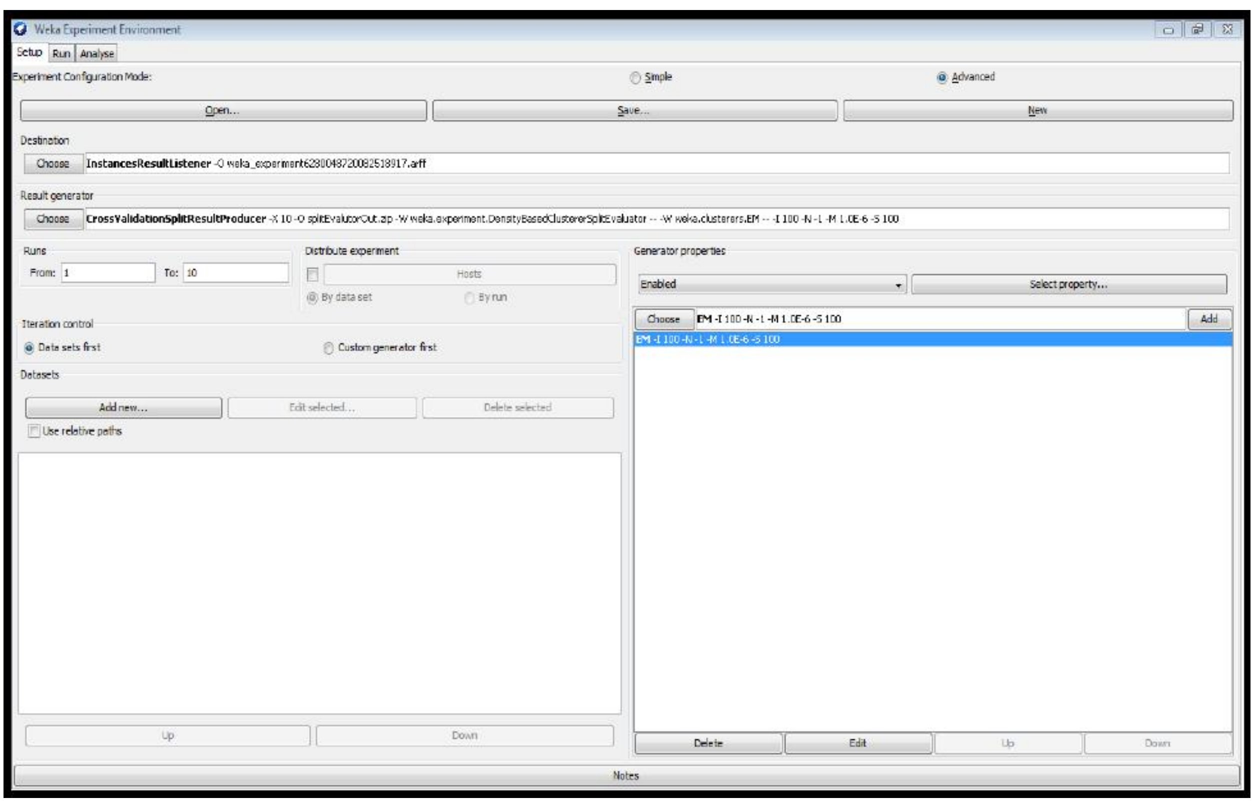

Next, choose “DensityBasedClustererSplitEvaluator” as the split evaluator to use.



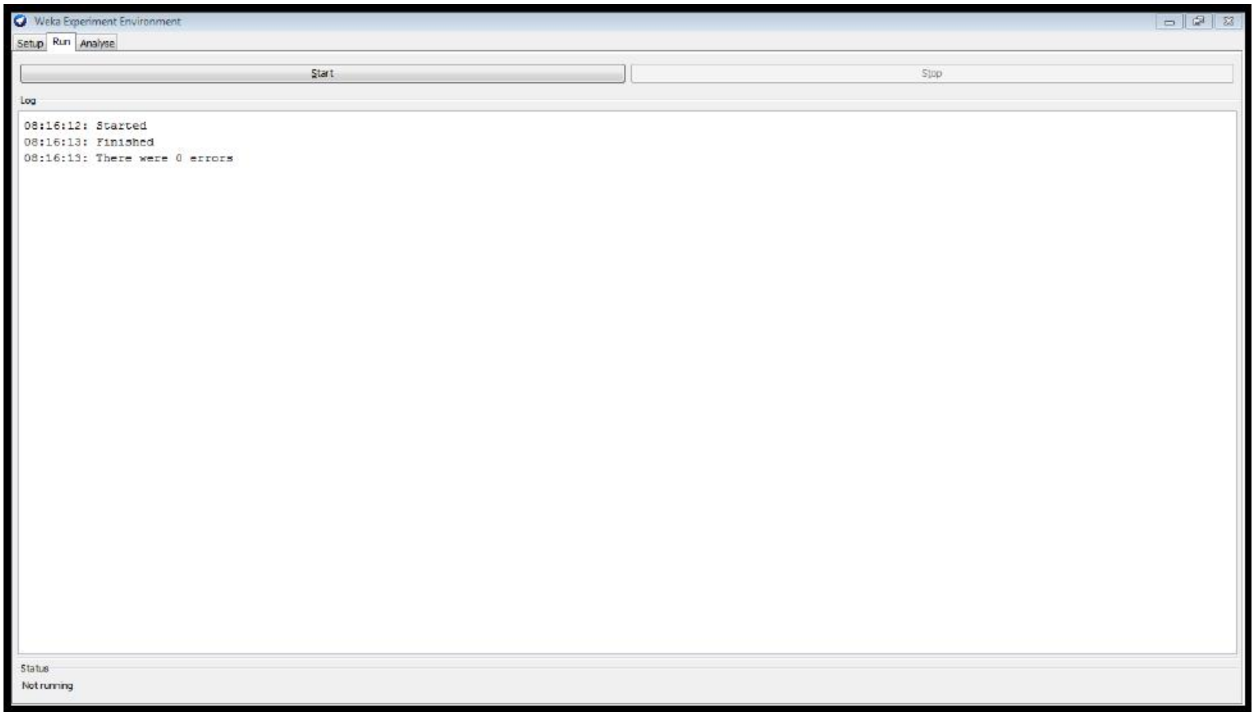
Once DensityBaseClusterSplitEvaluator has selected has been selected, you will notice that the Generator properties have become disabled. Enable them again and expand splitEvaluator. Select the clusterer node.



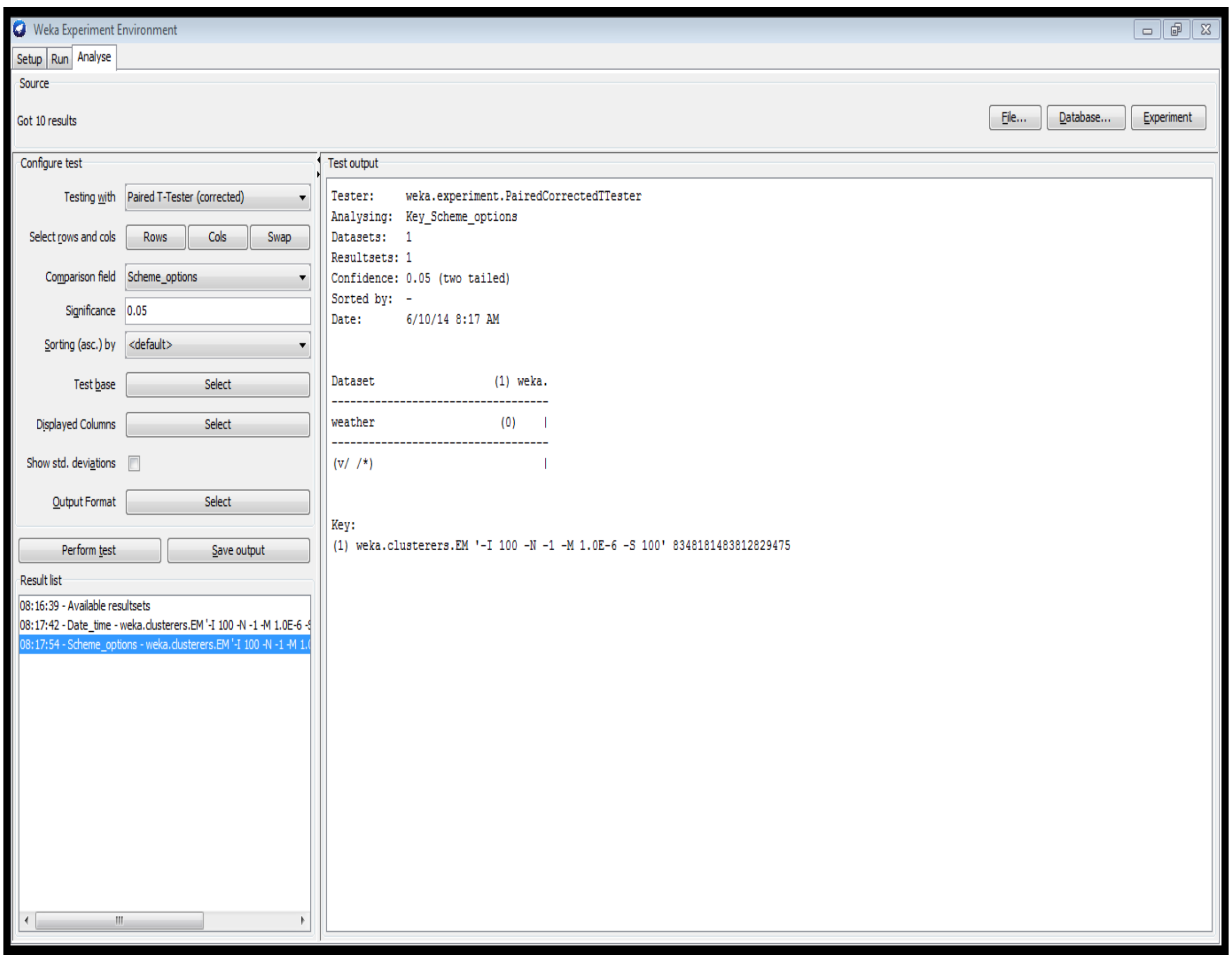
Now you will see that EM becomes the default clustere and gets added to the list of schemes. You can now add/delete other clusterers.



Once and experiment has been run ,




You can analyze results in the analyses panel. In the comparison field you will need to scroll down and select "humidity"



16. To Plot Multiple ROC curves

The KnowledgeFlow can draw multiple ROC curves in the same plot window, something that the Explorer cannot do. In this example we use J48 and RandomForest as classifiers.

• Click on the DataSources tab and choose ArffLoader from the toolbar (the mouse pointer will change to a cross hairs).

• Next place the ArffLoader component on the layout area by clicking somewhere on the layout (a copy of the ArffLoader icon will appear on the layout area).

• Next specify an ARFF file to load by first right clicking the mouse over the ArffLoader icon on the layout. A pop-up menu will appear. Select Configure under Edit in the list from this menu and browse to the location of your ARFF file.

• Next click the Evaluation tab at the top of the window and choose the ClassAssigner (allows you to choose which column to be the class) component from the toolbar. Place this on the layout.

• Now connect the ArffLoader to the ClassAssigner: first right click over the ArffLoader and select the dataSet under Connections in the menu. A rubber band line will appear. Move the mouse over the ClassAssigner component and left click - a red line labeled dataSet will connect the two components.

• Next right click over the ClassAssigner and choose Configure from the menu. This will pop up a window from which you can specify which column is the class in your data (last is the default).

• Next choose the ClassValuePicker (allows you to choose which class label to be evaluated in the ROC) component from the toolbar. Place this on the layout and right click over ClassAssigner and select dataSet from under Connections in the menu and connect it with the ClassValuePicker.

• Next grab a CrossValidationFoldMaker component from the Evaluation toolbar and place it on the layout. Connect the ClassAssigner to the CrossValidationFoldMaker by right clicking over ClassAssigner and selecting dataSet from under Connections in the menu.

• Next click on the Classifiers tab at the top of the window and scroll along the toolbar until you reach the J48 component in the trees section. Place a J48 component on the layout.

• Connect the CrossValidationFoldMaker to J48 TWICE by first choosing trainingSet and then testSet from the pop-up menu for the CrossValidationFoldMaker.

• Repeat these two steps with the RandomForest classifier.

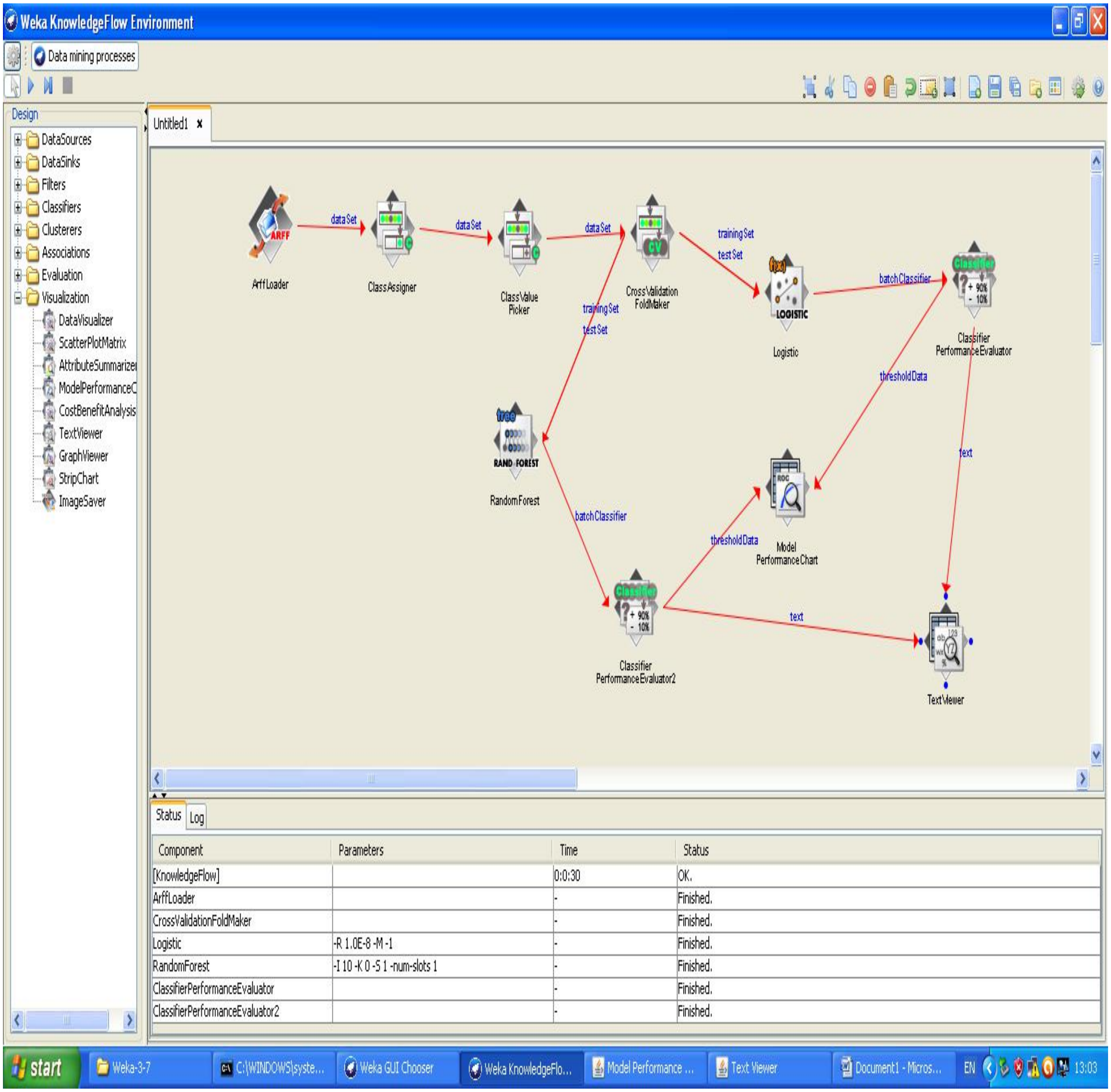
• Next go back to the Evaluation tab and place a ClassifierPerformanceEvaluator component on the layout. Connect J48 to this component by selecting the batchClassifier entry from the pop-up menu for J48. Add another ClassifierPerformanceEvaluator for RandomForest and connect them via batchClassifier as well.

• Next go to the Visualization toolbar and place a ModelPerformanceChart component on the layout. Connect both ClassifierPerformanceEvaluators to the ModelPerformanceChart by selecting the thresholdData entry from the pop-up menu for ClassifierPerformanceEvaluator.

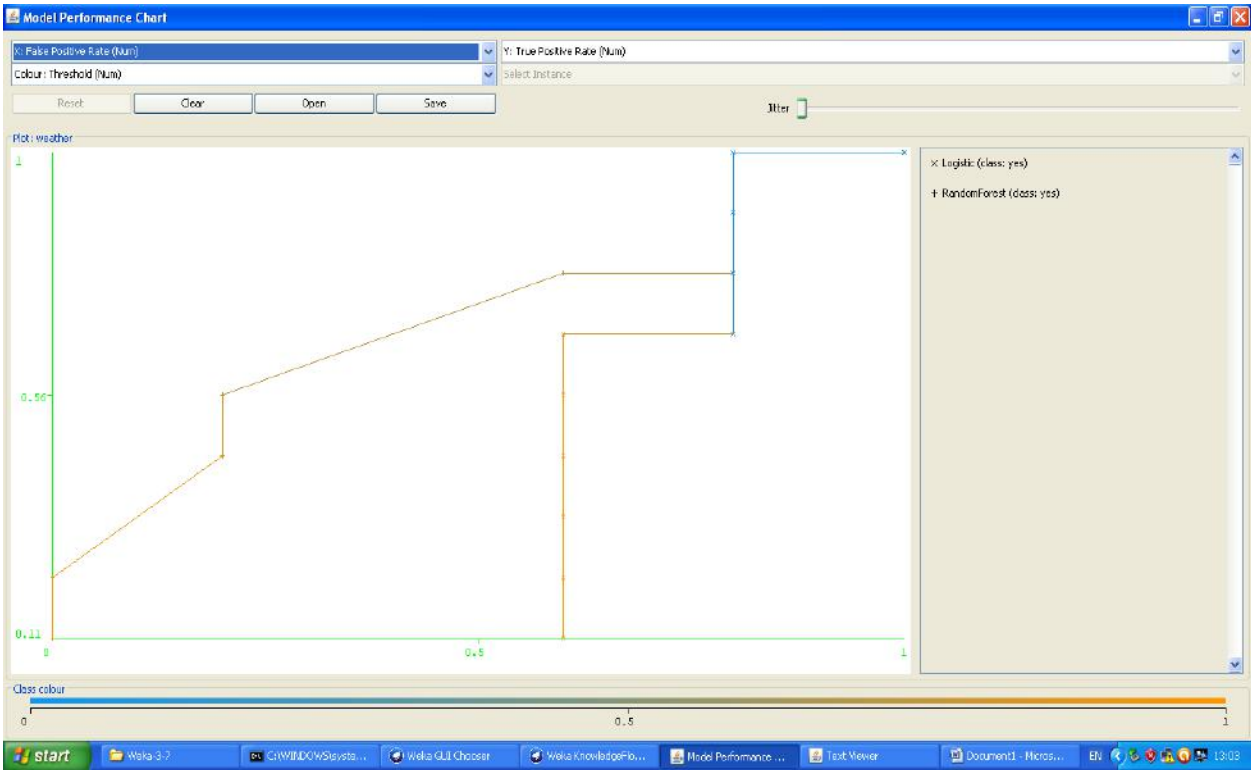
• Now start the flow executing by selecting Start loading from the pop-up menu for ArffLoader. Depending on how big the data set is and how long cross validation takes you will see some animation from some of the icons in the layout. You will also see some progress information in the Status bar and Log at the bottom of the window.

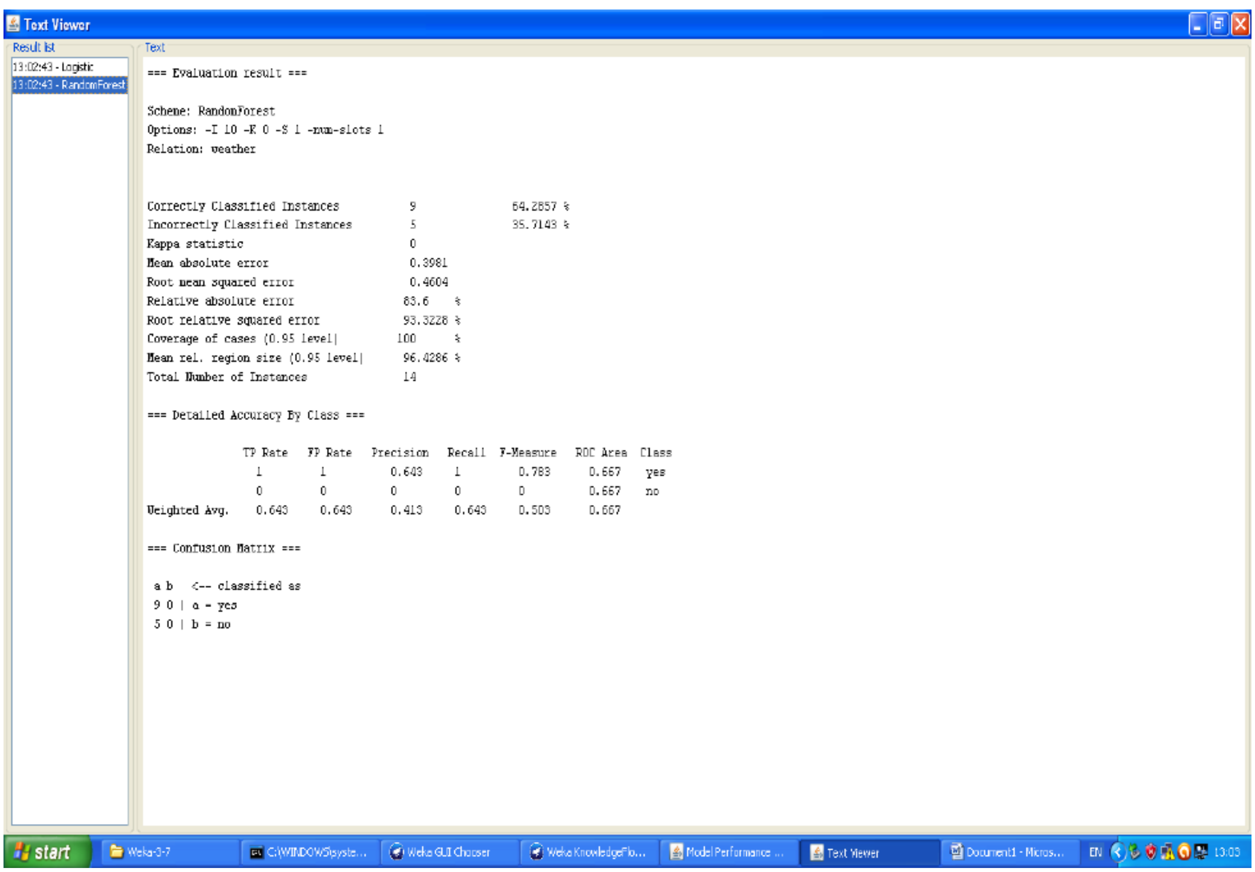
• Select Show plot from the popup-menu of the ModelPerformanceChart under the Actions section.

Go to weka knowledge flow environment.



Here are the two ROC curves generated from the UCI dataset credit-g, evaluated on the class label good:




```
=== Evaluation result ===

Scheme: RandomForest
Options: -I 10 -K 0 -S 1 -num-slots 1
Relation: weather


Correctly Classified Instances           9               64.2857 %
Incorrectly Classified Instances         5               35.7143 %
Kappa statistic                          0
Mean absolute error                      0.3981
Root mean squared error                  0.4604
Relative absolute error                 83.6     %
Root relative squared error             93.3228 %
Coverage of cases (0.95 level)         100      %
Mean rel. region size (0.95 level)      96.4286 %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall   F-Measure   ROC Area   Class
                 1         1         0.643      1         0.783      0.667     yes
                 0         0         0          0         0          0.667     no
Weighted Avg.    0.643     0.643     0.413      0.643     0.503      0.667

=== Confusion Matrix ===

 a b   <-- classified as
 9 0 | a = yes
 5 0 | b = no
```

17. To Process Data Incrementally

Some classifiers, clusterers and filters in Weka can handle data incrementally in a streaming fashion. Here is an example of training and testing naive Bayes incrementally. The results are sent to a TextViewer and predictions are plotted by a StripChart component.

Click on the DataSources tab and choose ArffLoader from the toolbar (the mouse pointer will change to a cross hairs).

• Next place the ArffLoader component on the layout area by clicking some-where on the layout (a copy of the ArffLoader icon will appear on the layout area).

• Next specify an ARFF file to load by first right clicking the mouse over the ArffLoader icon on the layout. A pop-up menu will appear. Select Configure under Edit in the list from this menu and browse to the location of your ARFF file.

• Next click the Evaluation tab at the top of the window and choose the ClassAssigner (allows you to choose which column to be the class) component from the toolbar. Place this on the layout.

• Now connect the ArffLoader to the ClassAssigner: first right click over the ArffLoader and select the dataSet under Connections in the menu. A rubber band line will appear. Move the mouse over the ClassAssigner component and left click - a red line labeled dataSet will connect the two components.

• Next right click over the ClassAssigner and choose Configure from the menu. This will pop up a window from which you can specify which column is the class in your data (last is the default).

• Now grab a NaiveBayesUpdateable component from the bayes section of the Classifiers panel and place it on the layout.

• Next connect the ClassAssigner to NaiveBayesUpdateable using a instance connection.

• Next place an IncrementalClassiferEvaluator from the Evaluation panel onto the layout and connect NaiveBayesUpdateable to it using a incrementalClassifier connection.

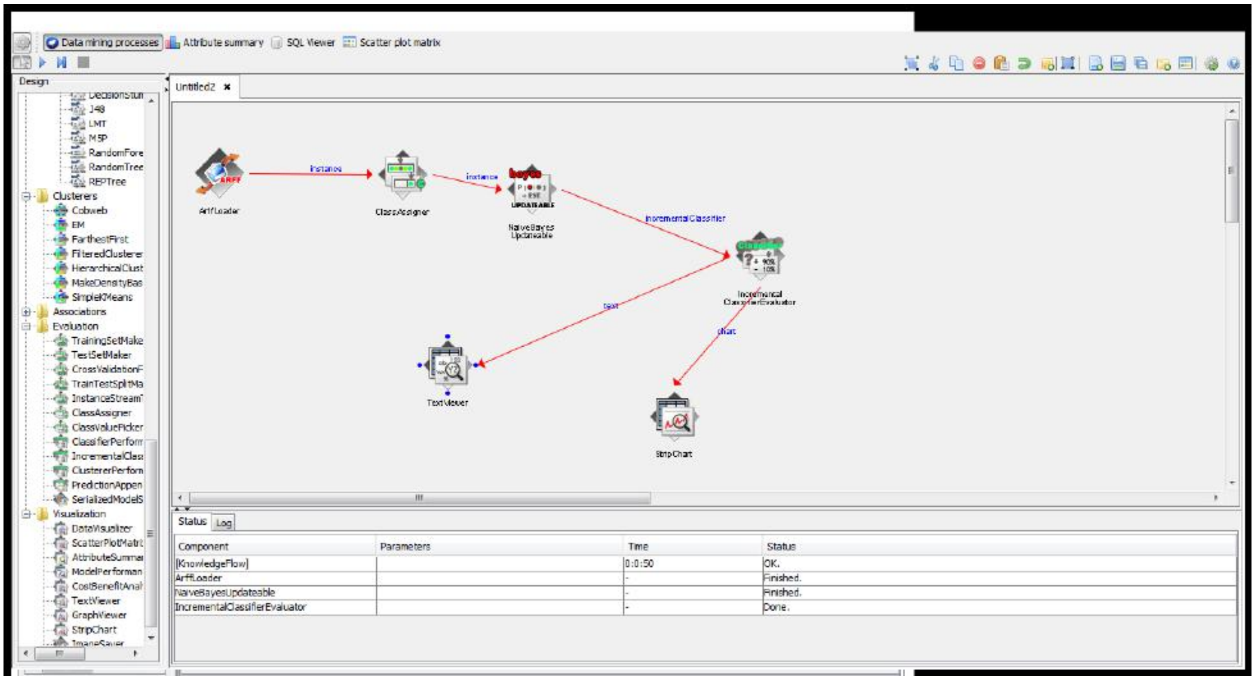
• Next place a TextViewer component from the Visualization panel on the Layout. Connect the IncrementalClassifierEvaluator to it using a text connection.

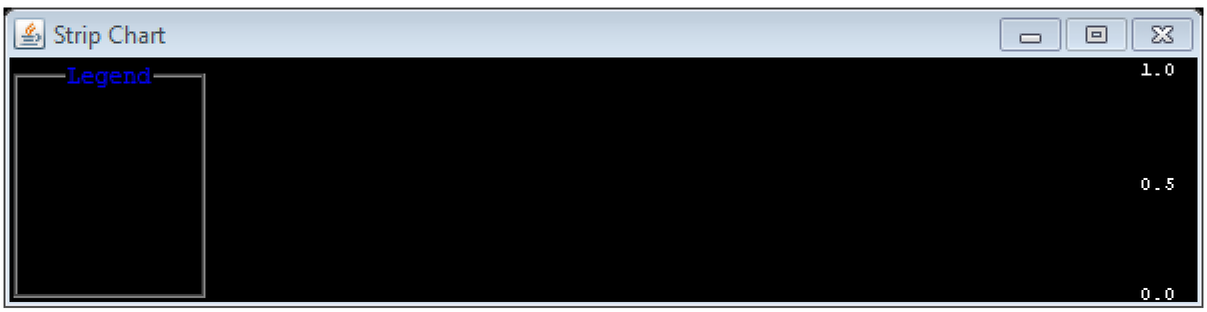
• Next place a StripChart component from the Visualization panel on the layout and connect IncrementalClassifierEvaluator to it using a chart connection.

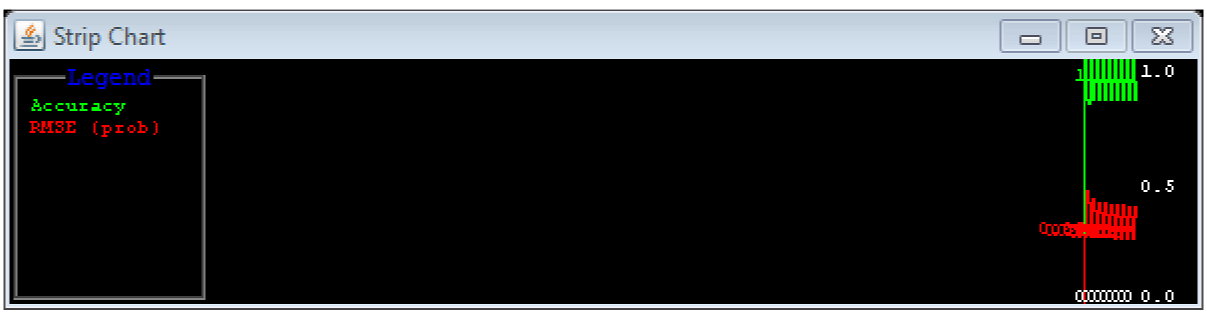
• Display the StripChart’s chart by right-clicking over it and choosing Show chart from the pop-up menu. Note: the StripChart can be configured with options that control how often data points and labels are displayed.

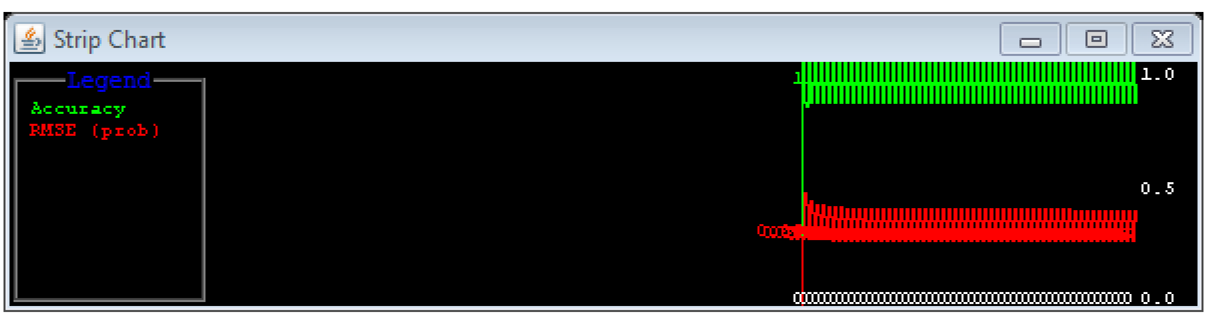
• Finally, start the flow by right-clicking over the ArffLoader and selecting Start loading from the pop-up menu.

Go to knowledgeflow environment.



**18. How to access a database using WEKA**

Go to the Control Panel

Choose Adminstrative Tools

Choose Data Sources (ODBC)

At the User DSN tab, choose Add...

Choose database

Microsoft Access

Note: Make sure your database is not open in another application before following the steps below.

Choose the Microsoft Access driver and click Finish

Give the source a name by typing it into the Data Source Name field

In the Database section, choose Select...

Browse to find your database file, select it and click OK

Click OK to finalize your DSN

You will need to configure a file called DatabaseUtils.props. This file already exists under the path weka/experiment/ in the weka.jar file (which is just a ZIP file) that is part of the Weka download. In this directory you will also find a sample file for ODBC connectivity, called DatabaseUtils.props.odbc, and one specifically for MS Access, called DatabaseUtils.props.msaccess (>3.4.14, >3.5.8, >3.6.0), also using ODBC. You should use one of the sample files as basis for your setup, since they already contain default values specific to ODBC access.

This file needs to be recognized when the Explorer starts. You can achieve this by making sure it is in the working directory or the home directory (if you are unsure what the terms working directory and home directory mean, see the \textit{Notes} section). The easiest is probably the second alternative, as the setup will apply to all the Weka instances on your machine.

Just make sure that the file contains the following lines at least:

jdbcDriver=sun.jdbc.odbc.JdbcOdbcDriver

jdbcURL=jdbc:odbc:dbname

where dbname is the name you gave the user DSN. (This can also be changed once the Explorer is running.)

Start up the Weka Explorer.

Choose Open DB...

The URL should read "jdbc:odbc:dbname" where dbname is the name you gave the user DSN.

Click Connect

Enter a Query, e.g., "select * from tablename" where tablename is the name of the database table you want to read. Or you could put a more complicated SQL query here instead.

Click Execute

When you're satisfied with the returned data, click OK to load the data into the Preprocess panel.

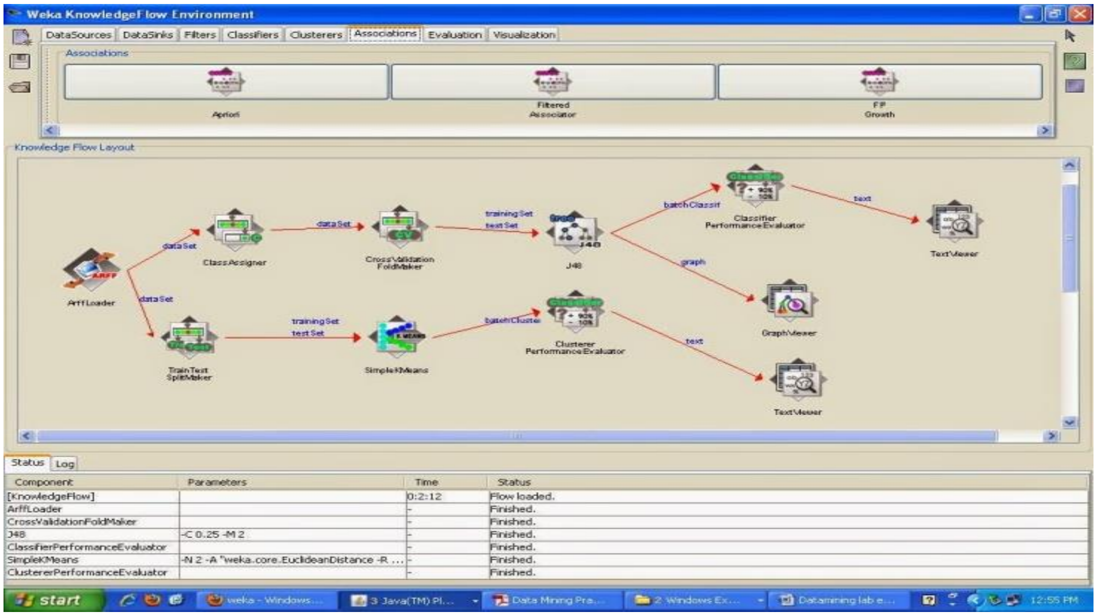



Figure: Knowledge flow directed graph for C4.5 and K-Means.

Exercises on Knowledge FlowComponent of WEKA

I. Use Knowledge flow canvas and develop a directed graph for C4.5 execution\
Goal: Setting up a flow to load an arff file (batch mode) and perform a cross validation using J48 (Weka's C4.5 implementation).

Steps to be done:

1. The Weka GUI Chooser window is used to launch Weka's graphical environments. Select the button labeled "KnowledgeFlow" to start the KnowledgeFlow. Alternatively, you can launch the KnowledgeFlow from a terminal window by typing "java weka.gui.beans.KnowledgeFlow".
2. First start the KnowlegeFlow.
3. Next click on the DataSources tab and choose "ArffLoader" from the toolbar (the mouse pointer will change to a "cross hairs").
4. Next place the ArffLoader component on the layout area by clicking somewhere on the layout (A copy of the ArffLoader icon will appear on the layout area).
5. Next specify an arff file to load by first right clicking the mouse over the ArffLoader icon on the layout. A pop-up menu will appear. Select "Configure" under "Edit" in the list from this menu and browse to the location of your arff file.
6. Next click the "Evaluation" tab at the top of the window and choose the "ClassAssigner" (allows you to choose which column to be the class) component from the toolbar. Place this on the layout.

7. Now connect the ArffLoader to the ClassAssigner: first right click over the ArffLoader and select the "dataSet" under "Connections" in the menu. A "rubber band" line will appear. Move the mouse over the ClassAssigner component and left click - a red line labeled "dataSet" will connect the two components.

8. Next right click over the ClassAssigner and choose "Configure" from the menu. This will pop up a window from which you can specify which column is the class in your data (last is the default).

9. Next grab a "CrossValidationFoldMaker" component from the Evaluation toolbar and place it on the layout. Connect the ClassAssigner to the CrossValidationFoldMaker by right clicking over "ClassAssigner" and selecting "dataSet" from under "Connections" in the menu.

10. Next click on the "Classifiers" tab at the top of the window and scroll along the toolbar until you reach the "J48" component in the "trees" section. Place a J48 component on the layout.

11. Connect the CrossValidationFoldMaker to J48 TWICE by first choosing "trainingSet" and then "testSet" from the pop-up menu for the CrossValidationFoldMaker.

12. Next go back to the "Evaluation" tab and place a "ClassifierPerformanceEvaluator" component on the layout. Connect J48 to this component by selecting the "batchClassifier" entry from the pop-up menu for J48.

13. Next go to the "Visualization" toolbar and place a "TextViewer" component on the layout. Connect the ClassifierPerformanceEvaluator to the TextViewer by selecting the "text" entry from the pop-up menu for ClassifierPerformanceEvaluator.

14. Now start the flow executing by selecting "Start loading" from the pop-up menu for ArffLoader. Depending on how big the data set is and how long cross validation takes you will see some animation from some of the icons in the layout (J48's tree will "grow" in the icon and the ticks will animate on the ClassifierPerformanceEvaluator). You will also see some progress information in the "Status" bar and "Log" at the bottom of the window.

15. When finished you can view the results by choosing show results from the pop-up menu for the TextViewer component.

II. Use Knowledge flow canvas and develop a directed graph for k-means execution

Exercises on Experimenter component of WEKA

1. Use experimenter to compare any two classifiers of your choice on iris dataset.

Exercises from WEKA textbook

1) Weather.nominal.arff

What are the values that the attribute temperature can have?

Load a new dataset. Click the Open file button and select the file iris.arff. . How many instances does this dataset have? How many attributes? What is the range of possible values of the attribute petallength?

2) Weather.nominal.arff

What is the function of the first column in the Viewer window? What is the class value of instance number 8 in the weather data?

Load the iris data and open it in the editor. How many numeric and how many nominal attributes does this dataset have?

3) Load the weather.nominal dataset. Use the filter weka.unsupervised.instance.RemoveWithValues to remove all instances in which the humidity attribute has the value high. To do this, first make the field next to the Choose button show the text RemoveWithValues. Then click on it to get the Generic Object Editor window, and figure out how to change the filter settings appropriately. Undo the change to the dataset that you just performed, and verify that the data has reverted to its original state.

4) Load the iris data using the Preprocess panel. Evaluate C4.5 on this data using (a) the training set and (b) cross-validation. What is the estimated percentage of correct classifications for (a) and (b)? Which estimate is more realistic? Use the Visualize classifier errors function to find the wrongly classified test instances for the cross-validation performed in previous Exercise. What can you say about the location of the errors?

5) Glass.arff

How many attributes are there in the dataset? What are their names? What is the class attribute? Run the classification algorithm IBk (weka.classifiers.lazy.IBk). Use cross-validation to test its performance, leaving the number of folds at the default value of 10. Recall that you can examine the classifier options in the Generic Object Editor window that pops up when you click the text beside the Choose button. The default value of the KNN field is 1: This sets the number of neighboring instances to use when classifying.

6) Glass.arff

What is the accuracy of IBk (given in the Classifier Output box)? Run IBk again, but increase the number of neighboring instances to k = 5 by entering this value in the KNN field. Here and throughout this section, continue to use cross-validation as the evaluation method.

What is the accuracy of IBk with five neighboring instances (k = 5)?

7) Ionosphere.arff

For J48, compare cross-validated accuracy and the size of the trees generated for (1) the raw data, (2) data discretized by the unsupervised discretization method in default mode, and (3) data discretized by the same method with binary attributes.

8) Apply the ranking technique to the labor negotiations data in labor.arff to determine the four most important attributes based on information gain. On the same data, run CfsSubsetEval for correlation-based selection, using the BestFirst search. Then run the wrapper method with J48 as the base learner, again using the BestFirst search. Examine the attribute subsets that are output. Which attributes are selected by both methods? How do they relate to the output generated by ranking using information gain?

9) Run Apriori on the weather data with each of the four rule-ranking metrics, and default settings otherwise. What is the top-ranked rule that is output for each metric?

## Exercises on Nearest Neighbor Learner

• We use a subset of the "Iris Plants Database" dataset (i.e., provided by WEKA, contained in the "iris.aff" file).

• Each plant record (i.e., example) is represented by the 5 attributes.

- SepalLength – the plant's sepal length in cm.
- SepalWidth – the plant's sepal width in cm.
- PetalLength – the plant's petal length in cm.
- PetalWidth – the plant's petal width in cm.
- Class – the classification attribute, with the possible values {Iris-setosa, Iris-versicolor, Iris-virginica}.

| PlantID | SepalLength | SepalWidth | PetalLength | PetalWidth | Class |
|---|---|---|---|---|---|
| 1 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 2 | 7.1 | 3.0 | 5.9 | 2.1 | Iris-virginica |
| 3 | 5.4 | 3.4 | 1.5 | 0.4 | Iris-setosa |
| 4 | 6.4 | 3.2 | 4.5 | 1.5 | Iris-versicolor |
| 5 | 6.3 | 3.3 | 4.7 | 1.6 | Iris-versicolor |
| 6 | 7.3 | 2.9 | 6.3 | 1.8 | Iris-virginica |
| 7 | 4.4 | 2.9 | 1.4 | 0.2 | Iris-setosa |
| 8 | 4.9 | 3.1 | 1.5 | 0.1 | Iris-setosa |
| 9 | 5.8 | 2.8 | 5.1 | 2.4 | Iris-virginica |
| 10 | 5.6 | 2.9 | 3.6 | 1.3 | Iris-versicolor |
| 11 | 6.9 | 3.2 | 5.7 | 2.3 | Iris-virginica |
| 12 | 6.0 | 3.4 | 4.5 | 1.6 | Iris-versicolor |
| 13 | 7.2 | 3.0 | 5.8 | 1.6 | Iris-virginica |
| 14 | 4.8 | 3.4 | 1.9 | 0.2 | Iris-setosa |
| 15 | 6.8 | 2.8 | 4.8 | 1.4 | Iris-versicolor |

## Exercises on Decision tree

• Let's assume that we have collected the following data set of users who decided to buy a computer and others who decided not.

• Each user record (i.e., example) is represented by the 5 attributes.

- Age, with the possible values {Young, Medium, Old}.
- Income, with the possible values {Low, Medium, High}.
- Student, with the possible values {Yes, No}.
- Credit_Rating, with the possible values {Fair, Excellent}.
- Buy_Computer – the classification attribute, with the possible values {Yes, No}.

**UserID Age Income Student Credit_Rating Buy_Computer**

1 Young High No Fair No

2 Young High No Excellent No

3 Medium High No Fair Yes

4 Old Medium No Fair Yes

5 Old Low Yes Fair Yes

6 Old Low Yes Excellent No

7 Medium Low Yes Excellent Yes

8 Young Medium No Fair No

9 Young Low Yes Fair Yes

10 Old Medium Yes Fair Yes

11 Young Medium Yes Excellent Yes

12 Medium Medium No Excellent Yes

13 Medium High Yes Fair Yes

14 Old Medium No Excellent No

15 Medium Medium Yes Fair No

16 Medium Medium Yes Excellent Yes

17 Young Low Yes Excellent Yes

18 Old High No Fair No

19 Old Low No Excellent No

20 Young Medium Yes Excellent Yes

• We want to predict, for each of the following users, if s/he will buy a computer or not.

- User #21. A young student with medium income and fair credit rating.
- User #22. A young non-student with low income and fair credit rating.
- User #23. A medium student with high income and excellent credit rating.
- User #24. An old non-student with high income and excellent credit rating.

**Use the WEKA tool**

• Convert the dataset containing 20 examples (i.e., Users #1-20) into the ARFF format (supported by WEKA), and save it in the "buy_comp.arff" file.

• For each user in the set of Users #21-24, set the values of the Buy_Computer attribute by the predictions computed manually in Part I. Convert the data of these four users into the ARFF format, and save it in the "buy_comp_extra.arff" file.

• Launch the WEKA tool, and then activate the "Explorer" environment.

• Open the "buy_comp" dataset (i.e., saved in the "buy_comp.arff" file).

- For each attribute and for each of its possible values, how many instances in each class have the feature value (i.e., the class distribution of the feature values)?

• Go to the "Classify" tab. Select the **Id3** classifier. Choose "Percentage split" (66% for training) test mode. Run the classifier and observe the results shown in the "Classifier output" window.

- How many instances used for the training? How many for the test?
- Does the test set currently used include the four instances of Users #21-24?
- How many instances are incorrectly classified?
- What is the MAE (mean absolute error) made by the learned DT?

- What can you infer from the information shown in the Confusion Matrix?
- Visualize the errors made by the learned DT. In the plot, how can you differentiate between the correctly and incorrectly classified instances? In the plot, how can you see the detailed information of an incorrectly classified instance?
- How can you save the learned DT to a file?
- How can you visualize the structure of the learned DT?

• Now, in the “Test options” panel select the “Supplied test set” option. Activate the nearby “Set…” button and locate the “buy_comp_extra.arff” file. Run the classifier and observe the results shown in the “Classifier output” window.

- How many instances used for the training? How many for the test?
- Does the test set currently used include the four examples (i.e., Users #21-24)?
- In the “Classifier output” window, where you can find the information that says for which of the four users (i.e., Users #21-24) the learned DT predicts correctly and for which others it predicts incorrectly?
- What is the MAE (mean absolute error) made by the learned DT?

**Exercises on the WEKA tool**

1. Launch the WEKA tool, and activate the **Explorer** environment.
2. Open the "**weather.nominal**" dataset
- How many instances (examples) contained in the dataset?
- How many attributes used to represent the instances?
- Which attribute is the class label?
- What is the data type (e.g., numeric, nominal, etc.) of the attributes in the dataset?
- For each attribute and for each of its possible values, how many instances in each class have the attribute value (i.e., the class distribution of the attribute values)?

3. Go to the **Classify** tab. Select the **ZeroR** classifier. Choose the "Cross-validation" (10 folds) test mode. Run the classifier and observe the results shown in the "Classifier output" window.
- How many instances are incorrectly classified?
- What is the MAE (mean absolute error) made by the classifier?
- What can you infer from the information shown in the Confusion Matrix?
- Visualize the classifier errors. In the plot, how can you differentiate between the correctly and incorrectly classified instances? In the plot, how can you see the detailed information of an incorrectly classified instance?
- How can you save the learned classifier to a file?
- How can you load a learned classifier from a file?

4. Choose the "Percentage split" (66% for training) test mode. Run the **ZeroR** classifier and observe the results shown in the "Classifier output" window.
- How many instances are incorrectly classified? Why this number is smaller than that observed in the previous experiment (i.e., using the cross-validation test mode)?
- What is the MAE made by the classifier?
- Visualize the classifier errors to see the detailed information.

5. Now, select the **Id3** classifier (i.e., you can find this classifier in the weka.classifiers.trees group). Choose the "Cross-validation" (10 folds) test mode. Run the **Id3** classifier and observe the results shown in the "Classifier output" window.
- How many instances are incorrectly classified?
- What is the MAE made by the classifier?
- Visualize the classifier errors.
- Compare these results with those observed for the **ZeroR** classifier in the cross-validation test mode. Which classifier, **ZeroR** or **Id3**, shows a better prediction performance for the current dataset and the cross-validation test mode?

6. Choose the "Percentage split" (66% for training) test mode. Run the **Id3** classifier and observe the results shown in the "Classifier output" window.

- How many instances are incorrectly classified?
- What is the MAE made by the classifier?
- Visualize the classifier errors.
- Compare the results made by the **Id3** classifier for the two considered test modes. In which test mode, does the classifier produces a better result (i.e., a smaller error)?
- Which classifier, **ZeroR** or **Id3**, shows a better prediction performance for the current dataset and the splitting test mode?

**Exercises on the probabilistic models**

• Let's assume we have the following data set that recorded (i.e., in a period of 25 days) whether or not a person played tennis depending on the outlook and wind conditions.

• Each instance (example) is represented by the three attributes.

o Outlook: a value of {Sunny, Overcast, Rain}.

o Wind: a value of {Weak, Strong}.

o PlayTennis: the classification attribute (i.e., Yes- the person plays tennis; No- the person does not play tennis).

**Date Outlook Wind PlayTennis**

1 Sunny Weak No

2 Sunny Strong No

3 Overcast Weak Yes

4 Rain Weak Yes

5 Rain Weak Yes

6 Rain Strong No

7 Overcast Strong Yes

8 Sunny Weak No

9 Sunny Weak Yes

10 Rain Weak Yes

11 Sunny Strong Yes

12 Overcast Strong Yes

13 Overcast Weak Yes

14 Rain Strong No

15 Sunny Strong Yes

16 Overcast Strong No

17 Overcast Weak Yes

18 Rain Weak No

19 Sunny Weak No

20 Rain Strong Yes

21 Sunny Weak Yes

22 Overcast Weak No

23 Rain Weak Yes

24 Sunny Strong Yes

25 Overcast Weak No

• We want to predict if the person will play tennis in the three future days.

o Day 26: (Outlook=Sunny, Wind=Strong) → PlayTennis=?

o Day 27: (Outlook=Overcast, Wind=Weak) → PlayTennis=?

o Day 28: (Outlook=Rain, Wind=Weak) → PlayTennis=?

**Classification / Prediction / Cluster analysis**

The goal of this assignment is to review prediction mining principles and methods, cluster analysis principles and methods, and to apply them to a dataset using Weka data mining tool**.**

**Heart dataset**

The first dataset studied is the ***cleveland*** dataset from ***UCI*** repository. This dataset describes numeric factors of heart disease. It can be downloaded from http://www.cs.waikato.ac.nz/~ml/weka/index_datasets.html and is contained in the ***datasets-numeric.jar*** archive.

**Zoo dataset**

The second dataset studied is the ***zoo*** dataset from ***UCI*** repository. This dataset describes animals with categorical features. It can be downloaded from http://www.cs.waikato.ac.nz/~ml/weka/index_datasets.html and is contained in the ***datasets-UCI.jar*** archive.

**1. Prediction in Weka *(100 points, 5 points per question)***

The goal of this data mining study is to predict the severity of heart disease in the ***cleveland*** dataset (variable ***num***) based on the other attributes. Answer the following questions:

a. What types of variables are in this dataset (numeric / ordinal / categorical)?

b. Load the data in ***Weka Explorer***. Select the ***Classify*** tab. How many different prediction algorithms are available (under ***functions***)?

c. Explain what is ***prediction*** in data mining.

d. Choose ***LinearRegression*** algorithm. Explain what is the principle of this algorithm.

e. Results of this algorithm can be interpreted in the following way. The first part of the output represents the coefficients of the linear equation of the type

$num = w_0 + w_1a_1 + \ldots + w_ka_k$.

The numbers provided in front of each attribute $a_k$ represent the $w_k$. Based on this, interpret the results you get from running ***LinearRegression*** on the dataset. What is the equation of the line found?

f. The second part of the results states the **correlation coefficient**, which measures the statistical correlation between the predicted and actual values (a coefficient of **+1** indicates a perfect

positive relationship, **0** indicates no linear relationship, and **–1** indicates a perfect negative relationship). Only positive correlations make sense in regression, and a coefficient above **0.5** signals a large correlational effect. The remaining figures are the mean absolute error (the average prediction error), the **root mean squared error** (the square root of the mean squared error), which is the most commonly used error measure, the relative absolute error (which compares this error with the one obtained if the prediction had been the mean), the root relative squared error (the square root of the error in comparison with the one obtained if the prediction had been the mean), and the total number of instances considered.
The overall interpretation of these is the following: a prediction is good when the correlation coefficient is as large as possible, and all the errors are as small as possible. These figures are used to compare several prediction results. How do you evaluate the fit of the equation provided in e), meaning how strong is this prediction?

g. It is also notable that an important figure is the square of the correlation coefficient ($\mathbf{R^2}$). In statistical regression analysis, which invented this prediction method, the most used success measures are **R** and $\mathbf{R^2}$. The latter represents the percentage of variation in the target figure accounted for by the model. For example, if we want to predict a sales volume based on three factors such as the advertising budget, the number of plays on the radio per week, and the attractiveness of the band, and if we get a correlation coefficient **R** of 0.8, then we learn from the model that $\mathbf{R^2}$ = 64% of the variability in the outcome (the sales volume) is accounted for by the three factors. How much of the variability of ***num*** can be predicted by the other attributes?

h. Are theses results compatible with the results of assignment #1, which used classification to predict ***num***?

Now compare these figures with the other classifiers provided in functions and fill-in the following table (except the last line):

| Method | Correlation coefficient | Mean absolute error | Root mean squared error | Relative absolute error | Root relative squared error |
|---|---|---|---|---|---|
| *LinearRegression* | | | | | |
| *SMOreg* | | | | | |
| *MultilayerPerceptron* | | | | | |
| *MultilayerPerceptron (optimized)* | | | | | |

a. Which prediction method provides best results with this dataset?

b. Try using the other functions to calculate the same regression. What problem(s) are you facing?

c. Explain what is logistic regression, and how it differs from linear regression.

d. Is in fact logistic regression a prediction method? If not, what kind of data mining method is logistic regression?

e. In the ***MultilayerPerceptron*** function, how many input nodes does this multiplayer perceptron have?

f. In the ***MultilayerPerceptron*** function, how many output nodes does this multiplayer perceptron have?

g. In the ***MultilayerPerceptron*** function, how many hidden layers does this multiplayer perceptron have?

h. After choosing GUI in the panel of ***MultilayerPerceptron*** options, paste here a screenshot of the graphical representation of this neural network.

i. What is its learning rate?

j. By changing the ***MultilayerPerceptron*** parameters, what is the configuration for the best results you get?

k. What best prediction results do you get (fill in the table above)?

**2. Clustering in Weka**

The goal of this data mining study is to find groups of animals in the ***zoo*** dataset, and to check whether these groups correspond to the real animal types in the dataset.

a. What types of variables are in this dataset?

b. How many rows / cases are there?

c. How many animal types are represented in this dataset? List them here.

d. After removing the ***type*** attribute, go to the ***Cluster*** tab. How many clustering algorithms are available in **Weka**?

e. List the clustering algorithms seen in class, and map these to the ones provided in **Weka**.

f. Start using the ***SimpleKMeans*** clusterer choosing 7 clusters. Do the clusters learnt and their centroids seem to match the animal types?

g. Compare results with ***EM*** clusterer (with 7 clusters), ***MakeDensityBasedClusterer***, ***FarthestFirst*** (with 7 clusters), and ***Cobweb***. Which algorithm seems to provide the best clustering match for this dataset?

h. Explain the principles of ***SimpleKMeans***, ***EM***, ***MakeDensityBasedClusterer***, and ***Cobweb*** clustering algorithms.

i. Are results easy to interpret, even with the tree visualizations provided?

j. What would make it easier to evaluate the usefulness of the clusters found?

a. List some animals that are misclassified, meaning classified in a cluster that does not correspond to their actual type, for instance a mammal clustered with fish, or a reptile clustered with amphibian.

b. By modifying the selected parameters, improve the classification, explain which modifications you made, and paste here the resulting dendrogram.

**Case Study 3**

In this assignment, you have to compare the performance of four classification approaches (simply compare the accuracy of the approaches):

- Decision Trees
- Ripper (Rule Learning system (JRip in WEKA)
- SVMs (Not in WEKA? If not use SVMLight or the like)
- Decision Trees with AdaBoost

on three different data sets from UCI, or from other sources of your choice.

**Data Preprocessing with Weka**

**The goal of this case study is to investigate how to preprocess data using Weka data mining tool.**

This assignment will be using **Weka** data mining tool. **Weka** is an open source Java development environment for data mining from the **University of Waikato in New Zealand**. It can be downloaded freely from **http://www.cs.waikato.ac.nz/ml/weka/**, **Weka** is really an asset for learning data mining because it is freely available, students can study how the different data mining models are implemented, and develop customized Java data mining applications. Moreover, data mining results from **Weka** can be published in the most respected journals and conferences, which make it a de facto developing environment of choice for research in data mining, where researchers often need to develop new data mining methods.

**Heart disease datasets**

The dataset studied is the **heart disease** dataset from **UCI repository** (**datasets-UCI.jar**). Two different datasets are provided: **heart-h.arff** (Hungarian data), and **heart-c.arff** (Cleveland data). These datasets describe factors of heart disease. They can be downloaded from: **http://www.cs.waikato.ac.nz/~ml/weka/index_datasets.html.**

The **machine mining project goal** is to better understand the risk factors for heart disease, as represented in the 14$^{th}$ attribute: **num** (<50 means no disease, and values <50-1 to <50-4 represent increasing levels of heart disease).

The **question** on which this machine learning study concentrates is whether it is possible to predict heart disease from the other known data about a patient. The **data mining** task of choice to answer this question will be classification/prediction, and several different algorithms will be used to find which one provides the best predictive power.

**1. Data preparation- integration**

We want to merge the two datasets into one, in a step called data integration. Revise **arff** notation from the tutorial, which is **Weka** data representation language. Answer the following questions:

a. Define what data integration means.

b. Is there an **entity identification** or **schema integration** problem in this dataset? If yes, how to fix it?

c. Is there a **redundancy** problem in this dataset? If yes, how to fix it?

d. Are there **data value conflicts** in this dataset? If yes, how to fix it?

e. Integrate the two datasets into one single dataset, which will be used as a starting point for the next questions, and load it in the **Explorer**. How many instances do you have? How many attributes?

f. Paste a screenshot of the **Explorer** window.

**2. Descriptive data summarization**

Before preprocessing the data, an important step is to get acquainted with the data – also called **data understanding** in **CRISP-DM**.

a. Stay in the **Preprocess** tab for now. Study for example the **age** attribute. What is its **mean**? Its **standard deviation**? Its **min** and **max**?

b. Provide the **five-number summary** of this attribute. Is this figure provided in **Weka**?

c. Specify which attributes are numeric, which are ordinal, and which are categorical/nominal.

d. Interpret the graphic showing in the lower right corner of the **Explorer**. How can you name this graphic? What do the red and blue colors mean (pay attention to the pop-up messages that appear when dragging the mouse over the graphic)? What does this graphic represent?

e. Visualize all the attributes in graphic format. Paste a screenshot.

f. Comment on what you learn from these graphics.

g. Switch to the **Visualize** tab. What is the term used in the textbook to name the series of boxplots represented? By selecting the maximum jitter, and looking at the **num** column – the last one – can you determine which attributes seem to be the most linked to heart disease? Paste the **boxplot** representing the attribute you find the most predictive of heart disease (Y) as a function of **num** (X).

h. Does any pair of different attributes seem correlated?

**3. Data preparation – selection**

The datasets studied have already been processed by selecting a subset of attributes relevant for the data mining project.

a. From the documentation provided in the dataset, how many attributes were originally in these datasets?

b. With **Weka**, attribute selection can be achieved either from the specific **Select attributes** tab, or within **Preprocess** tab. List the different options in **Weka** for selecting attributes, with a short explanation about the corresponding method.

c. In comparison with the methods for attribute selection detailed in the textbook, are any missing? Are any provided in **Weka** not provided in the textbook?

**4. Data preparation - cleaning**

Data cleaning deals with such defaults of real-world data as incompleteness, noise, and inconsistencies. In **Weka**, data cleaning can be accomplished by applying **filters** to the data in the **Preprocess** tab.

a. **Missing values**. List the methods seen in class for dealing with missing values, and which **Weka filters** implement them – if available. Remove the missing values with the method of your choice, explaining which filter you are using and why you make this choice. If a filter is not available for your method of choice, develop a new one that you add to the available filters as a Java class.

b. **Noisy data**. List the methods seen in class for dealing with noisy data, and which **Weka filters** implement them – if available.

c. **Outlier detection**. List the methods seen in class for detecting outliers. How would you detect outliers with **Weka**? Are there any outliers in this dataset, and if yes, list some of them.

d. Save the cleaned dataset into **heart-cleaned.arff**, and paste here a screenshot showing at least the first 10 rows of this dataset – with all the columns.

**5. Data preparation - transformation**

Among the different data transformation techniques, explore those available through the **Weka Filters**. Stay in the **Preprocess** tab for now. Study the following data transformation only:

a. **Attribute construction** – for example adding an attribute representing the sum of two other ones. Which **Weka filter** permits to do this?

b. **Normalize** an attribute. Which **Weka filter** permits to do this? Can this filter perform Min-max normalization? Z-score normalization? Decimal normalization? Provide detailed information about how to perform these in **Weka**.

c. **Normalize** all real attributes in the dataset using the method of your choice – state which one you choose.

d. Save the normalized dataset into **heart-normal.arff**, and paste here a screenshot showing at least the first 10 rows of this dataset – with all the columns.

**6. Data preparation- reduction**

Often, data mining datasets are too large to process directly. Data reduction techniques are used to preprocess the data. Once the data mining project has been successful on these reduced data, the larger dataset can be processed too.

a. Stay in the **Preprocess** tab for now. Beside attribute selection, a reduction method is to select rows from a dataset. This is called sampling. How to perform sampling with **Weka filters**? Can it perform the two main methods: **Simple Random Sample Without Replacement**, and **Simple Random Sample With Replacement**?

The KDD Process in Weka

**Heart disease datasets**

The dataset studied is the ***heart disease*** dataset from ***UCI repository***. Two different datasets are provided: ***heart-h.arff*** (Hungarian data), and ***heart-c.arff*** (Cleveland data). These datasets describe factors of heart disease. Both these data sets are available to you on the assignment page.

The ***data mining project goal*** is to better understand the risk factors for heart disease, as represented in the 14th attribute: ***num*** (<50 means no disease, and values <50-1 to <50-4 represent increasing levels of heart disease).

The ***question*** on which this machine learning study concentrates is whether it is possible to predict heart disease from the other known data about a patient. The ***data mining*** task of choice to answer this question will be classification/prediction, and several different algorithms will be used to find which one provides the best predictive power. However this exercise focuses on the various aspects of the KDD process.

**1. Data preparation- integration**

We want to merge the two datasets into one, in a step called data integration. Revise ***arff*** notation from the tutorial, which is ***Weka*** data representation language. Answer the following questions:

a. Define what data integration means. (in your own words)

b. Is there an ***entity identification*** or ***schema integration*** problem in this dataset? If yes, how to fix it?

c. Is there a ***redundancy*** problem in this dataset? If yes, how to fix it?

d. Are there ***data value conflicts*** in this dataset? If yes, how to fix it?

e. Integrate the two datasets into one single dataset, which will be used as a starting point for the next questions, and load it in the ***Explorer***. How many instances do you have? How many attributes? (You could do this using Excel or spreadsheet programs. First, save your individual files as "csv" files in weka, Open them in a spreadsheet viewing program. Copy the rows from one file to another. Save the merged file (csv). Open it in weka and save it as "csv". Take care of the above questions. Think about rectifying potential problems.

f. Paste a screenshot of the ***Explorer*** window.

**2. Descriptive data summarization**

Before preprocessing the data, an important step is to get acquainted with the data – also called ***data understanding***.

a. Stay in the ***Preprocess*** tab for now. Study for example the ***age*** attribute. What is its ***mean***? What are its ***standard deviation***, ***Min*** and ***max***?

b. Provide the ***five-number summary*** of this attribute. Is this figure provided in ***Weka***? This is min, max, median, lower 25% quartile and upper 25% quartile.

c. Specify which attributes is numeric, which are ordinal, and which are categorical/nominal.

d. Interpret the graphic showing in the lower right corner of the ***Explorer***. How can you name this graphic? What do the red and blue colors mean (pay attention to the pop-up messages that appear when dragging the mouse over the graphic)?

What does this graphic represent?

e. Visualize all the attributes in graphic format. Paste a screenshot.

f. Comment on what you learn from these graphics.

g. Switch to the ***Visualize*** tab. By selecting the maximum jitter, and looking at the ***num*** column – the last one – can you determine which attributes seem to be the most linked to heart disease? Paste the ***boxplot*** representing the attribute you find the most predictive of heart disease (Y) as a function of ***num*** (X).

h. Does any pair of different attributes seem correlated?

**3. Data preparation – selection**

The datasets studied have already been processed by selecting a subset of attributes relevant for the data mining project.

a. From the documentation provided in the dataset, how many attributes were originally in these datasets?

b. With ***Weka***, attribute selection can be achieved either from the specific ***Select attributes*** tab, or within ***Preprocess*** tab. List the different options in ***Weka*** for selecting attributes, with a short explanation about the corresponding method.

**4. Data preparation - cleaning**

Data cleaning deals with such defaults of real-world data as incompleteness, noise, and inconsistencies. In ***Weka***, data cleaning can be accomplished by applying ***filters*** to the data in the ***Preprocess*** tab.

a. **Missing values**. List the methods seen in class for dealing with missing values, and which ***Weka filters*** implement them – if available. Remove the missing values with the method of your choice, explaining which filter you are using and why you make this choice.

b. **Noisy data**. List the methods seen in class for dealing with noisy data, and which ***Weka filters*** implement them – if available.

c. Save the cleaned dataset into ***heart-cleaned.arff***, and paste here a screenshot showing at least the first 10 rows of this dataset – with all the columns.

**5. Data preparation - transformation**

1. Among the different data transformation techniques, explore those available through the ***Weka Filters***. Stay in the ***Preprocess*** tab for now. Study the following data transformation only:

a. ***Attribute construction*** – for example adding an attribute representing the sum of two other ones. Which ***Weka filter*** permits to do this?

b. ***Normalize*** an attribute. Which ***Weka filter*** permits to do this? Can this filter perform Min-max normalization? Z-score normalization? Decimal normalization? Provide detailed information about how to perform these in ***Weka***.

c. ***Normalize*** all real attributes in the dataset using the method of your choice – state which one you choose.

d. Save the normalized dataset into ***heart-normal.arff***, and paste here a screenshot showing at least the first 10 rows of this dataset – with all the columns.

**6. Data preparation- reduction**

Often, data mining datasets are too large to process directly. Data reduction techniques are used to preprocess the data. Once the data mining project has been successful on these reduced data, the larger dataset can be processed too.

a. Stay in the ***Preprocess*** tab for now. Beside attribute selection, a reduction method is to select rows from a dataset. This is called sampling. How to perform sampling with ***Weka filters***? Can it perform the two main methods: ***Simple Random Sample Without Replacement***, and ***Simple Random Sample With Replacement***?

## Association Rules

APRIORI works with categorical values only. Therefore we will use a different dataset called "adult"; This dataset contains census data about 48842 US adults. The aim is to predict whether their income exceeds $50000. The dataset is taken from the Delve website, and originally came from the UCI Machine Learning Repository. More information about it is available in the original UCI Documentation.

Download a copy of adult.arff and load it into Weka.

This dataset is not immediately ready for use with APRIORI. First, reduce its size by taking a random sample. You can do this with the 'ResampleFilter' in the preprocess tab sheet: click on the label under 'Filters', choose 'ResampleFilter' from the drop down menu, set the 'sampleSizePercentage' (to 15 eg), click 'OK' and 'Add', and click 'Apply Filters'. The 'Working relation' is now a subsample of the original adult dataset. Now we have to get rid of the numerical attributes. You can choose to discard them, or to discretise them. We will discretise the first attribute ('age'): choose the 'DiscretizeFilter', set 'attributeIndices' to 'first', bins to a low number, like 4 or 5, and the other options to 'False'. Then add this new filter to the others. We will get rid of the other numerical attributes: choose an 'AttributeFilter', set 'invertSelection' to 'False', and enter the indices of the remaining numeric attributes (3,5,11-13). Apply all the filters together now. Then click on 'Replace' to make the resulting 'Working relation' the new 'Base relation'.

Now go to the 'Associate' tab sheet and click under 'Associator'. Set 'numRules' to 25, and keep the other options on their defaults. Click 'Start' and observe the results. What do you think about these rules? Are they useful?

From the previous, it is obvious that some attributes should not be examined simultaneously because they lead to trivial results. Go back to the 'Preprocess' sheet. If you have replaced the original 'Base relation' by the 'Working relation', you can include and exclude attributes very easily: delete all filters from the 'Filters' window, then remove the check mark next to the attributes you want to get rid of and click 'Apply Filters'. You now have a new 'Working relation'. Try to remove different combinations of the attributes that lead to trivial association rules. Run APRIORI several times and look for interesting rules. You will find that there is often a whole range of rules which are all based on the same simpler rule. Also, you will often get rules that don't include the target class. This is why in most cases you would use APRIORI for dataset exploration rather than for predictive modelling.

**Exercise 2**

Association analysis is concerned with discovering interesting correlations or other relationships between variables in large databases. We are interested into relationships between features themselves, rather than features and class as in the standard classification problem setting. Hence searching for association patterns is no different from classification except that instead of predicting just the class, we try to predict arbitrary attributes or attribute combinations.

1. Fire up Weka software, launch the explorer window and select the \Preprocess" tab. Open the weather.nominal data-set (\weather.nominal.arff", this should be in the ./data/ directory of the Weka install).

2. Often we are in search of discovering association rules showing attribute-value conditions that occur frequently together in a given set of data, such as; buys(X, computer") & buys(X, \scanner") =) buys (X,\printer") [support = 2%, confidence = 60%]. Where confidence and support are measures of rule interestingness. A support of 2% means that 2% of all transactions under analysis show that computer, scanner and printer are purchased together. A confidence of 60% means that 60% of the customers who purchased a computer and a scanner also bought a printer. We are interested into association rules that apply to a reasonably large number of instances and have a reasonably high accuracy on the instances to which they apply.

Weka has three build-in association rule learners. These are, \Apriori", \Predictive Apriori" and \Tertius", however they are not capable of handling numeric data. Therefore in this exericse we use weather data.

(a) Select the \Associate" tab to get into the Association rule mining perspective of Weka. Under \Associator" select and run each of the following \Apriori", \Predictive Apriori" and \Tertius".

Briefly inspect the output produced by each Associator and try to interpret its meaning.

(b) In association rule mining the number of possible association rules can be very large even with tiny datasets, hence it is in our best interest to reduce the count of rules found, to only the most interesting ones. This is usually achieved by setting minimum thresh-

olds on support and confidence values. Still in the \Associate" view, select the \Apriori" algorithm again, click on the textbox next to the \Choose" button and try, in turn, different values for the following parameters \lowerBoundMinSupport" (min threshold for support), \minMetric" (min threshold for confidence). As you change these parameter values what do you notice about the rules that are found by the associator? Note that the parameter \numRules" limits the maximum number of rules that the associator looks for, you can try changing this value.

(c) This time run the Apriori algorithm with the \outputItemSets" parameter set to true. You will notice that the algorithm now also outputs a list of \Generated sets of large itemsets:" at di_erent levels. If you have the module's Data Mining book by Witten & Frank with you, then you can compare and contrast the Apriori associator's output with the association rules on pages 114-116. I also strongly recommend to read through chapter 4.5 in your own time, while playing with the weather data in Weka, this chapter gives a nice & easy introduction to association rules. Notice in particular how the item sets and association rules compare with Weka and tables 4.10-4.11 in the book.

(d) Compare the association rules output from Apriori and Tertius (you can do this by navigating through the already build associator models in the \Result list" on the right side of the screen).

Make sure that the Apriori algorithm shows at least 20 rules. Think about how the association rules generated by the two different methods compare to each other?

Something to always remember with association rules, is that they should not be used for prediction directly, that is without further analysis or domain knowledge, as they do not necessarily indicate causality.

They are however a very helpful starting point for further exploration and for building a better understanding of our data.

As you should certainly know by this point, in order to identify associations between parameters a correlation matrix and scatter plot matrix can be very useful fs.

**Exercise 3: Boolean association rule mining in Weka**

The dataset studied is the ***weather*** dataset from Weka's ***data*** folder

The goal of this data mining study is to find strong association rules in the ***weather.nominal*** dataset. Answer the following questions:

a. What type of variables are in this dataset (numeric / ordinal / categorical) ?

b. Load the data in ***Weka Explorer***. Select the ***Associate*** tab. How many different association rule mining algorithms are available?

c. Choose ***Apriori*** algorithm with the following parameters (which you can select by clicking on the chosen algorithm: support threshold = 15% (lowerBoundMinSupport = 0.15), confidence threshold = 90% (metricType = confidence, minMetric = 0.9), number of rules = 50 (numRules = 50). After starting the algorithm, how many rules do you find? Could you use the regular ***weather*** dataset to get the results? Explain why.

d. Paste a screenshot of the ***Explorer*** window showing at least the first 20 rules.

e. Define the concepts of ***support***, ***confidence***, and ***lift*** for a rule. Write here the first rule discovered. What is its support? Its confidence? Interpret the meaning of these terms and this rule in this particular example.

f. ***Apriori*** algorithm generates association rules from frequent itemsets. How many itemsets of size 4 were found? Which rule(s) have been generated from itemset of size 4 (temperature=mild, windy=false, play=yes, outlook=rainy)? List their numbers in the list of rules.

**Prediction: Linear regression**

Linear Regression can be very useful in association analysis of numerical values, in fact regression analysis is a powerful approach to modeling the relationship between a dependent and independent variables. Simple regression is when we predict from one independent variable and multiple regression is when we predict from more than one independent variables. The model we attempt to _t is a linear one which is, very simply, drawing a line through the data. Of all the lines that can possibly be drawn through the data, we are looking for the one that best fits the data. In fact, we look to find a line that best satisfies

$\gamma = \beta 0 + \beta 1x + \varepsilon$

So a most accurate model is that which yields a best fit line to the data in question, we are looking for minimal sum of squared deviations between actual and fitted values, this is called method of least squares. So now that we have briefly reminded ourselves of the very basics of regression lets directly move onto an example in Weka.

**Exercise 1**

(a) In Weka go back to the \Preprocess" tab. Open the iris data-set (\iris.tar_", this should be in the ./data/ directory of the Weka install).

(b) In the \Attributes" section (bottom left of the screen) select the \class" feature and click \Remove". We need to do this, as simple linear regression cannot deal with non numeric values.

(c) Next select the \Classify" tab to get into the Classification perspective of Weka, and choose \LinearRegression" (under \functions").

(d) Clicking on the textbox next to the \Choose" button brings up the parameter editor window. Click on the \More" button to get information about the parameters. Make sure that \attributeSelectionMethod" is set to \No attribute selection" and "\eliminate-ColinearAttributes" is set to \False".

(e) Finally make sure that you select the parameter "\petalwidth" in the dropdown box just under the "\Test Options". Hit Start to run the regression.

Inspect the results, in particular pay attention to the Linear Regression Model formula returned, and the coefficients and intercept of the straight line equation. As this is a numeric prediction/regression problem, accuracy is measured with Root Mean Squared Error, Mean Absolute Error and the likes. As most of you will have clearly noticed, you can repeat this process for regressing the other features in turn, and compare how well the different features can be predicted.

**Exercise 2**

• Launch the WEKA tool, and then activate the "Explorer" environment.

• Open the "cpu" dataset (i.e., contained in the "cpu.arff" file).

- For each attribute and for each of its possible values, how many instances in each class have the feature value (i.e., the class distribution of the feature values)?

• Go to the "Classify" tab. Select the **SimpleLinearRegression** learner. Choose "Percentage split" (66% for training) test mode. Run the classifier and observe the results shown in the "Classifier output" window.

- Write down the learned regression function.

- What is the MAE (mean absolute error) made by the learned regression function?
- Visualize the errors made by the learned regression function. In the plot, how can you see the detailed information of a predicted instance?

• Now, in the "Test options" panel select the "Cross-validation" option (10 folds). Run the classifier and observe the results shown in the "Classifier output" window.

- Write down the learned regression function.
- What is the MAE (mean absolute error) made by the learned regression function?
- Visualize the errors made by the learned regression function. In the plot, how can you see the detailed information of a predicted instance?

## Interpreting Weka Output

Below is the output from Weka when using the weka.classifiers.trees.J48 classifier with the file $WEKAHOME/data/iris.arff as a training file and no testing file. I.e. using the command:

java weka.classifiers.trees.J48 -t $WEKAHOME/data/iris.arff

In square brackets ([,]) there are comments on how to interpret the output.

```
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
```

[ Above is the decision tree constructed by the J48 classifier. This indicates how the classifier uses the attributes to make a decision. The leaf nodes indicate which class an instance will be assigned to should that node be reached. The numbers in brackets after the leaf nodes indicate the number of instances assigned to that node, followed by how many of those instances are incorrectly classified as a result. With other classifiers some other output will be given that indicates how the decisions are made, e.g. a rule set. Note that the tree has been pruned. An unpruned tree and be produced by using the "-U" option. ]

Time taken to build model: 0.05 seconds

Time taken to test model on training data: 0.01 seconds

=== Error on training data ===

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

[ This gives the error levels when applying the classifier to the training data it was constructed from. For our purposes the most important figures here are the numbers of correctly and incorrectly classified instances. With the exception of the Kappa statistic, the remaining statistics compute various error measures based on the class probabilities assigned by the tree. ]

=== 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
```

[ This shows for each class, how instances from that class received the various classifications. E.g. for class "b", 49 instances were correctly classified but 1 was put into class "c". ]

=== Stratified cross-validation ===

Correctly Classified Instances 144 96 %

Incorrectly Classified Instances 6 4 %

Kappa statistic 0.94

Mean absolute error 0.035

Root mean squared error 0.1586

Relative absolute error 7.8705 %

Root relative squared error 33.6353 %

Total Number of Instances 150

[ This gives the error levels during a 10-fold cross-validation. The "-x" option can be used to specify a different number of folds. The correctly/incorrectly classified instances refers to the case where the instances are used as test data and again are the most important statistics here for our purposes. ]

```
=== Confusion Matrix ===

  a  b  c   <-- classified as

 49  1  0 |  a = Iris-setosa

  0 47  3 |  b = Iris-versicolor

  0  2 48 |  c = Iris-virginica
```

[ This is the confusion matrix for the 10-fold cross-validation, showing what classification the instances from each class received when it was used as testing data. E.g. for class "a" 49 instances were correctly classified and 1 instance was assigned to class "b". ]

## Classification Exercises

**Exercise 1.**

1. Fire up the Weka (Waikato Environment for Knowledge Analysis) soft-
ware, launch the explorer window and select the \Preprocess" tab. Open the iris data-set (\iris.ar_", this should be in the ./data/ directory of the Weka install).
2. Select the \Classify" tab. Under the \Test options" section you have four different testing options. How do each (we cannot use \supplied test set" option as we have no applicable _le) of these options select the training/testing? Which testing mode do you think will perform best? (the ExplorerGuide.pdf", in the ./ directory of the Weka install may help).
3. Under \Classifier" select \MultilayerPerceptron". What type of classifier is this? How does this classifier work? What main parameters can be specified for this classifier?
4. Under \Test options" select \Use training set" and under \More options" check \Output predictions". Now click \Start" to start training the model. You should see a stream of output appear in the window named \Classifier output". What do each of the following sections tell you about the model?

(a) \Predictions on ..."

(b) \Summary"

(c) \Detailed accuracy by class"

(d) \Confusion matrix"

5. Under \Results list" you should see your model, right click on it and select \Visualise classifier errors", points marked with a square are errors i.e. incorrectly classified. How do you think the classifier performed on the test data?

6. Under \Test options" vary the option selected i.e. \cross-validation" or \percentage" and their parameters i.e. \folds" and \%". Then start the training phase again for each model. For each model analyse the classifier output and visualise the classifier errors. How do the different training techniques affect the model? Which technique performed the best? How does this compare to your initial prediction in 4?

7. Repeat the exercise 6 with the \J48" (Decision Tree) and \RBFNetwork" classifiers. How do these compare to each other? How do these compare to the MultilayerPerceptron"?

## Classification

1. The distinct stages of designing a classification model are outlined below:

_ Collect your raw data

_ Clean your data (e.g. outlier removal, missing data removal etc.)

_ Preprocess the data (e.g. normalization, standardization, etc.)

_ Determine the type of problem (i.e. classification or regression)

_ Pick an appropriate classifier (e.g. multilayer perceptron, decision tree, linear regression, etc.)

_ Choose some default parameters for the classifier, the choice of classifier and parameters constitute your model

2. Pick a training/testing strategy (e.g. percentage split, cross-validation etc.)

_ Train the classifer using your training/testing strategy

_ Analyse the performance of your model

_ If your results are unsatisfactory consider altering your model (i.e. changing the classifer, its parameters, and/or your training/testing strategy) and re- training/testing

_ If your results are satisfactory validate your model on an unseen set of cleaned and preprocessed data.

## Clustering

### 1) Clustering using K-Means

Get to the Weka Explorer environment and load the training file using the **Preprocess** mode. Try first with **weather.arff.** Get to the **Cluster** mode (by clicking on the **Cluster** tab) and select a clustering algorithm, for example SimpleKMeans. Then click on **Start** and you get the clustering result in the output window. The actual clustering for this algorithm is shown as one instance for each cluster representing the **cluster centroid**.

```
Scheme:       weka.clusterers.SimpleKMeans -N 2 -S 10
Relation:     weather
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
              play
Test mode:    evaluate on training data
```

```
=== Clustering model (full training set) ===
 kMeans
======
Number of iterations: 4
Within cluster sum of squared errors: 16.156838252701938
Cluster centroids:
Cluster 0
 Mean/Mode:  rainy 75.625  86      FALSE     yes
 Std Devs:   N/A      6.5014  7.5593 N/A     N/A
Cluster 1
 Mean/Mode:  sunny 70.8333 75.8333 TRUE      yes
 Std Devs:   N/A      6.1128 11.143  N/A     N/A

=== Evaluation on training set ===

kMeans
======
Number of iterations: 4
Within cluster sum of squared errors: 32.31367650540387
Cluster centroids:
Cluster 0
 Mean/Mode:  rainy 75.625  86      FALSE yes
 Std Devs:   N/A      6.5014  7.5593 N/A     N/A
Cluster 1
 Mean/Mode:  sunny 70.8333 75.8333 TRUE yes
 Std Devs:   N/A      6.1128 11.143  N/A     N/A
Clustered Instances
0       8 ( 57%)
1       6 ( 43%)
```

---

**Evaluation**

The way Weka evaluates the clusterings depends on the cluster mode you select. Four different cluster modes are available (as buttons in the Cluster mode panel):

1. **Use training set** (default). After generating the clustering Weka classifies the training instances into clusters according to the cluster representation and computes the percentage

of instances falling in each cluster. For example, the above clustering produced by k-means shows 43% (6 instances) in cluster 0 and 57% (8 instances) in cluster 1.

2. In **Supplied test set** or **Percentage split** Weka can evaluate clusterings on separate test data if the cluster representation is probabilistic (e.g. for EM).

3. **Classes to clusters evaluation.** In this mode Weka first ignores the class attribute and generates the clustering. Then during the test phase it assigns classes to the clusters, based on the majority value of the class attribute within each cluster. Then it computes the classification error, based on this assignment and also shows the corresponding confusion matrix. An example of this for k-means is shown below.

---

```
Scheme:       weka.clusterers.SimpleKMeans -N 2 -S 10
Relation:     weather
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
Ignored:
              play
Test mode:    Classes to clusters evaluation on training data

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

kMeans
======
Number of iterations: 4
Within cluster sum of squared errors: 11.156838252701938
Cluster centroids:
Cluster 0
 Mean/Mode:  rainy 75.625  86      FALSE
 Std Devs:   N/A      6.5014  7.5593 N/A
Cluster 1
 Mean/Mode:  sunny 70.8333 75.8333 TRUE
 Std Devs:   N/A      6.1128 11.143  N/A

=== Evaluation on training set ===

kMeans
======
Number of iterations: 4
Within cluster sum of squared errors: 22.31367650540387
Cluster centroids:
Cluster 0
 Mean/Mode:  rainy 75.625  86      FALSE
 Std Devs:   N/A      6.5014  7.5593 N/A
```

Cluster 1
 Mean/Mode:  sunny 70.8333 75.8333 TRUE
 Std Devs:   N/A      6.1128 11.143  N/A
Clustered Instances
0       8 ( 57%)
1       6 ( 43%)

Class attribute: play
Classes to Clusters:
 0 1  <-- assigned to cluster
 5 4 | yes
 3 2 | no
Cluster 0 <-- yes
Cluster 1 <-- no
Incorrectly clustered instances : 7.0  50      %

**EM**

The EM clustering scheme generates probabilistic descriptions of the clusters in terms of **mean** and **standard deviation** for the numeric attributes and value **counts** (incremented by 1 and modified with a small value to avoid zero probabilities) - for the nominal ones. In "Classes to clusters" evaluation mode this algorithm also outputs the log-likelihood, assigns classes to the clusters and prints the confusion matrix and the error rate, as shown in the example below.

Clustered Instances

0       4 ( 29%)
1      10 ( 71%)

Log likelihood: -8.36599

Class attribute: play
Classes to Clusters:
 0 1  <-- assigned to cluster
 2 7 | yes
 2 3 | no
Cluster 0 <-- no
Cluster 1 <-- yes
Incorrectly clustered instances : 5.0  35.7143 %

---

**Cobweb**

Cobweb generates hierarchical clustering, where clusters are described probabilistically. Below is an example clustering of the weather data (weather.arff). The class attribute (play) is ignored (using the **ignore attributes** panel) in order to allow later classes to clusters evaluation. Doing this automatically through the "Classes to clusters" option does not make much sense for hierarchical clustering, because of the large number of clusters. Sometimes we need to evaluate particular clusters or levels in the clustering hierarchy. We shall discuss here an approach to this.

Let us first see how Weka **represents** the Cobweb clusters. Below is a copy of the output window, showing the run time information and the structure of the clustering tree.

---

```
Scheme:       weka.clusterers.Cobweb -A 1.0 -C 0.234
Relation:     weather
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
Ignored:
              play
Test mode:    evaluate on training data


=== Clustering model (full training set) ===
Number of merges: 2
Number of splits: 1
Number of clusters: 6
node 0 [14]
|   node 1 [8]
|   |   leaf 2 [2]
|   node 1 [8]
|   |   leaf 3 [3]
|   node 1 [8]
|   |   leaf 4 [3]
node 0 [14]
|   leaf 5 [6]

=== Evaluation on training set ===
Number of merges: 2
Number of splits: 1
Number of clusters: 6
node 0 [14]
|   node 1 [8]
|   |   leaf 2 [2]
|   node 1 [8]
|   |   leaf 3 [3]
|   node 1 [8]
|   |   leaf 4 [3]
node 0 [14]
|   leaf 5 [6]
Clustered Instances
2       2 ( 14%)
3       3 ( 21%)
```

4 3 ( 21%)
5 6 ( 43%)

---

Here is some comment on the output above:

- **-A** 1.0 **-C** 0.234 in the command line specifies the Cobweb parameters **A**cuity and **C**utoff (see the text, page 215). They can be specified through the pop-up window that appears by clicking on area left to the Choose button.

- **node N** or **leaf N** represents a **subcluster, whose parent cluster is N.**

- The **clustering tree** structure is shown as a horizontal tree, where subclusters are aligned at the same column. For example, cluster 1 (referred to in node 1) has three subclusters 2 (leaf 2), 3 (leaf 3) and 4 (leaf 4).

- The **root** cluster is 0. Each line with **node 0** defines a subcluster of the root.

- The number in square brackets after **node N** represents the number of instances in the parent cluster **N**.

- Clusters with [1] at the end of the line are **instances**.

- For example, in the above structure cluster 1 has 8 instances and its subclusters 2, 3 and 4 have 2, 3 and 3 instances correspondingly.

- To view the clustering tree **right click** on the last line in the **result list** window and then select **Visualize tree**.

To **evaluate** the Cobweb clustering using the **classes to clusters** approach we need to know the class values of the instances, belonging to the clusters. We can get this information from Weka in the following way: After Weka finishes (with the class attribute ignored), **right click** on the last line in the **result list** window. Then choose **Visualize cluster assignments** - you get the **Weka cluster visualize** window. Here you can view the clusters, for example by putting **Instance_number** on X and **Cluster** on Y. Then click on **Save** and choose a file name (*.arff). Weka saves the **cluster assignments** in an ARFF file. Below is shown the file corresponding to the above Cobweb clustering.

---

```
@relation weather_clustered
@attribute Instance_number numeric
@attribute outlook {sunny,overcast,rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {TRUE,FALSE}
@attribute play {yes,no}
@attribute Cluster {cluster0,cluster1,cluster2,cluster3,cluster4,cluster5}

@data
0,sunny,85,85,FALSE,no,cluster3
1,sunny,80,90,TRUE,no,cluster5
```

2,overcast,83,86,FALSE,yes,cluster2
3,rainy,70,96,FALSE,yes,cluster4
4,rainy,68,80,FALSE,yes,cluster4
5,rainy,65,70,TRUE,no,cluster5
6,overcast,64,65,TRUE,yes,cluster5
7,sunny,72,95,FALSE,no,cluster3
8,sunny,69,70,FALSE,yes,cluster3
9,rainy,75,80,FALSE,yes,cluster4
10,sunny,75,70,TRUE,yes,cluster5
11,overcast,72,90,TRUE,yes,cluster5
12,overcast,81,75,FALSE,yes,cluster2
13,rainy,71,91,TRUE,no,cluster5

---

To represent the cluster assignments Weka adds a new attribute **Cluster** and includes its corresponding values at the end of each data line. Note that **all other** attributes are shown, including the ignored ones (play, in this case). Also, **only the leaf clusters are shown**.

Now, to **compute the classes to clusters error** in, say, **cluster 3** we look at the corresponding data rows in the ARFF file and get the distribution of the class variable: {no, no, yes}. This means that the majority class is **no** and the error is **1/3**.

If we want to compute the error **not only for leaf clusters**, we need to look at the clustering structure (the Visualize tree option helps here) and determine how the leaf clusters are combined in other clusters at higher levels of the hierarchy. For example, at the top level we have two clusters - 1 and 5. We can get the class distribution of 5 directly from the data (because 5 is a leaf) - **3 yes's** and **3 no's**. While for cluster 1 we need its subclusters - 2, 3 and 4. Summing up the class values we get **6 yes's** and **2 no's**. Finally, the majority in **cluster 1** is **yes** and in **cluster 5** is **no** (could be yes too) and the error (for the top level partitioning in two clusters) is **5/14**.

Weka provides another approach to see the instances belonging to each cluster. When you visualize the clustering tree, you can click on a node and then see the visualization of the instances falling into the corresponding cluster (i.e. into the leafs of the subtree). This is a very useful feature, however if you ignore an attribute (as we did with "play" in the experiments above) it does not show in the visualization.

## Data Preprocessing Exercises

### Exercise 1) Attribute Relevance Ranking

For each step, open the indicated file in the “Preprocess” window. Then, go to the “Attribute Selection” window and set the “Attribute selection mode to “Use full training set”. For below mentioned case, perform attribute ranking using the following attribute selection methods with default parameters:

a) InfoGainAttributeEval; and

b) GainRatioAttributeEval;

These attribute selection methods should consider only non-class dimensions (for each set, the class attribute is indicated above the “Start” button). Record the output of each run in a text file called “output.txt”. For that, copy the output of the run from the “Attribute selection output” window in the Explorer and paste it at the end of the “output.txt” file.

**a).** Perform attribute ranking on the “contact-lenses.arff” data set using the two attribute ranking methods with default parameters.

**Evaluation**

Once you have performed the experiments, you should spend some time evaluating your results. In particular, try to answer at least the following questions: Why would one need attribute relevance ranking? Do these attribute-ranking methods often agree or disagree? On which data set(s), if any, these methods disagree? Does discretization and its method affect the results of attribute ranking? Do missing values affect the results of attribute ranking? Record these and any other observations in a Word file called “Observations.doc”.

**Exercise 2**

1. Fire up the Weka (Waikato Environment for Knowledge Analysis) software, launch the explorer window and select the \Preprocess" tab.

2. Open the iris data-set (\iris.ar_", this should be in the ./data/ directory of the Weka install). What information do you have about the data set (e.g. number of instances, attributes and classes)? What type of attributes does this data-set contain (nominal or numeric)? What are the classes in this data-set? Which attribute has the greatest standard deviation? What does this tell you about that attribute? (You might also find it useful to open \iris.ar_" in a text editor).

3. Under \Filter" choose the \Standardize" _lter and apply it to all attributes. What does it do? How does it afect the attributes' statistics? Click \Undo" to un-standardize the data and now apply the \Normalize" filter and apply it to all the attributes. What does it do? How does it affect the attributes' statistics? How does it differ from \Standardize"? Click \Undo" again to return the data to its original state.

4. At the bottom right of the window there should be a graph which visualizes the data-set, making sure \Class: class (Nom)" is selected in the drop-down box click \Visualize All". What can you interpret from these graphs? Which attribute(s) discriminate best between the classes in the data-set? How do the \Standardize" and \Normalize" filters affect these graphs?

5. Under \Filter" choose the \AttributeSelection" filter. What does it do? Are the attributes it selects the same as the ones you chose as discriminatory above? How does its behavior change as you alter its parameters?

6. Select the \Visualize" tab. This shows you 2D scatter plots of each attribute against each other attribute (similar to the F1 vs F2 plots from tutorial 1). Make sure the drop-down box at the bottom says \Color: class (Nom)". Pay close attention to the plots between attributes you think discriminate best between classes, and the plots between attributes selected by the \AttributeSelection" filter. Can you verify from these plots whether your thoughts and the \AttributeSelection" filter are correct? Which attributes are correlated?

**<u>Attribute-Relation File Format (ARFF)</u>**

An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the

Department of Computer Science of The University of Waikato for use with the Weka machine learning software.

**Overview**

ARFF files have two distinct sections. The first section is the **Header** information, which is followed the **Data** information.

The **Header** of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types. An example header on the standard IRIS dataset looks like this:

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
%      (a) Creator: R.A. Fisher
%      (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
%      (c) Date: July, 1988
%
@RELATION iris
 @ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}
```

The **Data** of the ARFF file looks like the following:

```
@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
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```

Lines that begin with a % are comments. The **@RELATION**, **@ATTRIBUTE** and **@DATA** declarations are case insensitive.

**The ARFF Header Section**

The ARFF Header section of the file contains the relation declaration and attribute declarations.

**The @relation Declaration**

The relation name is defined as the first line in the ARFF file. The format is:

```
@relation <relation-name>
```

where <relation-name> is a string. The string must be quoted if the name includes spaces.

**The @attribute Declarations**

Attribute declarations take the form of an orderd sequence of **@attribute** statements. Each attribute in the data set has its own **@attribute** statement which uniquely defines the name of that attribute and it's data type. The order the attributes are declared indicates the column position in the data section of the file. For example, if an attribute is the third one declared then Weka expects that all that attributes values will be found in the third comma delimited column. The format for the **@attribute** statement is:

@attribute <attribute-name> <datatype>

where the *<attribute-name>* must start with an alphabetic character. If spaces are to be included in the name then the entire name must be quoted.

The *<datatype>* can be any of the four types currently (version 3.2.1) supported by Weka:

- numeric
- <nominal-specification>
- string
- date [<date-format>]

where <nominal-specification> and <date-format> are defined below. The keywords **numeric**, **string** and **date** are case insensitive.

**Numeric attributes**

Numeric attributes can be real or integer numbers.

**Nominal attributes**

Nominal values are defined by providing an <nominal-specification> listing the possible values: {<nominal-name1>, <nominal-name2>, <nominal-name3>, ...} For example, the class value of the Iris dataset can be defined as follows:

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

Values that contain spaces must be quoted.

**String attributes**

String attributes allow us to create attributes containing arbitrary textual values. This is very useful in text-mining applications, as we can create datasets with string attributes, then write Weka Filters to manipulate strings (like StringToWordVectorFilter). String attributes are declared as follows:

@ATTRIBUTE LCC string

**Date attributes**

Date attribute declarations take the form:

@attribute <name> date [<date-format>]

where <name> is the name for the attribute and <date-format> is an optional string specifying how date values should be parsed and printed (this is the same format used by SimpleDateFormat). The default format string accepts the ISO-8601 combined date and time format: "yyyy-MM-dd'T'HH:mm:ss".

Dates must be specified in the data section as the corresponding string representations of the date/time (see example below).

**ARFF Data Section**

The ARFF Data section of the file contains the data declaration line and the actual instance lines.

**The @data Declaration**

The **@data** declaration is a single line denoting the start of the data segment in the file. The format is:

```
@data
```

**The instance data**

Each instance is represented on a single line, with carriage returns denoting the end of the instance.
Attribute values for each instance are delimited by commas. They must appear in the order that they were declared in the header section (i.e. the data corresponding to the nth **@attribute** declaration is always the nth field of the attribute).
Missing values are represented by a single question mark, as in:

```
@data

4.4,?,1.5,?,Iris-setosa
```

Values of string and nominal attributes are case sensitive, and any that contain space must be quoted, as follows:

```
@relation LCCvsLCSH
 @attribute LCC string
@attribute LCSH string
@data
AG5,   'Encyclopedias and dictionaries.;Twentieth century.'
AS262, 'Science -- Soviet Union -- History.'
AE5,   'Encyclopedias and dictionaries.'
AS281, 'Astronomy, Assyro-Babylonian.;Moon -- Phases.'
AS281, 'Astronomy, Assyro-Babylonian.;Moon -- Tables.'
```

Dates must be specified in the data section using the string representation specified in the attribute declaration. For example:

```
@RELATION Timestamps

@ATTRIBUTE timestamp DATE "yyyy-MM-dd HH:mm:ss"

@DATA

"2001-04-03 12:12:12"

"2001-05-03 12:59:55"
```

**Sparse ARFF files**

Sparse ARFF files are very similar to ARFF files, but data with value 0 are not be explicitly represented.
Sparse ARFF files have the same header (i.e **@relation** and **@attribute** tags) but the data section is different. Instead of representing each value in order, like this:

```
@data
0, X, 0, Y, "class A"
0, 0, W, 0, "class B"
```

the non-zero attributes are explicitly identified by attribute number and their value stated, like this:

```
@data

{1 X, 3 Y, 4 "class A"}

{2 W, 4 "class B"}
```

Each instance is surrounded by curly braces, and the format for each entry is: <index> <space> <value> where index is the attribute index (starting from 0).
Note that the omitted values in a sparse instance are **0**, they are not "missing" values! If a value is unknown, you must explicitly represent it with a question mark (?).
**Warning**: There is a known problem saving SparseInstance objects from datasets that have string attributes. In Weka, string and nominal data values are stored as numbers; these numbers act as indexes into an array of possible attribute values (this is very efficient). However, the first string value is assigned index 0: this means that, internally, this value is stored as a 0. When a SparseInstance is written, string instances with internal value 0 are not output, so their string value is lost (and when the arff file is read again, the default value 0 is the index of a different string value, so the attribute value appears to change). To get around this problem, add a dummy string value at index 0 that is never used whenever you declare string attributes that are likely to be used in SparseInstance objects and saved as Sparse ARFF files.