

Weka Exercises

Today we learned the basics of running Weka and used it to build a decision tree for a very small data set. In this exercise you will build a decision tree for a more realistic data set. You will also apply Naïve Bayes and k nearest neighbour methods to the same data set. In the course of doing this you will find out how to discover what each of the learning procedures provided by Weka actually does and how their parameters may be modified.

The Soybean Diagnosis Data Set

The data set for this exercise is another of those supplied when you downloaded the Weka system. It should be found in Weka-3-9/data/soybean.arff.

Before running Weka, it is worth having a brief look at the data file using a text editor. Lines beginning with % are comments and you will see that a great deal of background information is supplied in this form. This includes details of the data itself and references to previous work using the data. There are 683 examples, each of which has 35 attributes plus the class attribute. The task is to assign examples to one of 19 disease classes.

Building a Decision Tree

Open Weka Explorer and in the “preprocess” window then use the browser to select the soybean data set. The Explorer window should now look like this:

The screenshot shows the Weka Explorer interface with the 'Preprocess' window active. The 'Current relation' is 'soybean' with 683 instances and 36 attributes. The 'Selected attribute' is 'date', which has 7 distinct values. The 'Attributes' list shows 'date' selected. The 'Class' is 'class (Nom)'. A bar chart at the bottom right shows the distribution of the 'date' attribute.

No.	Label	Count
1	april	26
2	may	75
3	june	93
4	july	118
5	august	131
6	september	149

Next, as in Exercise 1, click the 'Classify' tab and select J48 as the classifier. However, this time leave the default of 10 way cross validation unchanged as the experimental procedure. As it is the final attribute, 'class' will have been selected as the classification attribute by default.

Now click the start button to run J48 on the data set. As usual, the results appear in the scrollable panel on the right. However, it is useful to have them in a separate window, particularly if you want to compare two or more sets of results. To put them in a separate window, right click on the appropriate item in the window on panel on the lower left headed 'Result list (right-click for options)' and then select 'View in separate window'. A copy of the results will appear in a separate window which can be enlarged to fill the screen.

Another option provided on this pop up menu is 'Visualise tree'. You can try this with the current set of results but you will see that the visual output is just a mess because the tree has so many nodes.

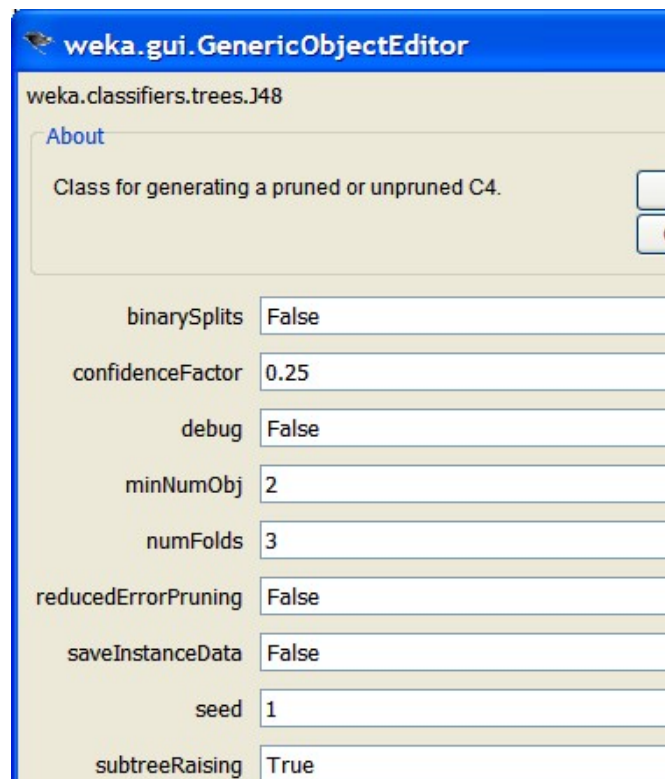
Fortunately the tree is more intelligible in the separate results window you have just created. Here it is presented in indented text format just after the heading 'J48 pruned tree'. At the end of the tree is the information that it contains 93 nodes of which 61 are leaves. The rest of the output provides information about how well the program performed. For now, all we need to note is that it classified 91.51% of examples correctly.

Do not close the separate window displaying the results of this run.

Building an unpruned tree

J48 built a pruned decision tree in the run you have just done because that is the default option. However, it is easy to get it to produce an unpruned tree.

Left click on 'J48' in the 'Classify' window and a window will appear that looks like this:



Change 'unpruned' in this window to 'True', click OK, and then run the program again.

The output for this new run replaces the output from your earlier run in the scrollable panel of the Explorer window. However, your earlier results are still there in the separate window you created. You can make another separate window for these new results.

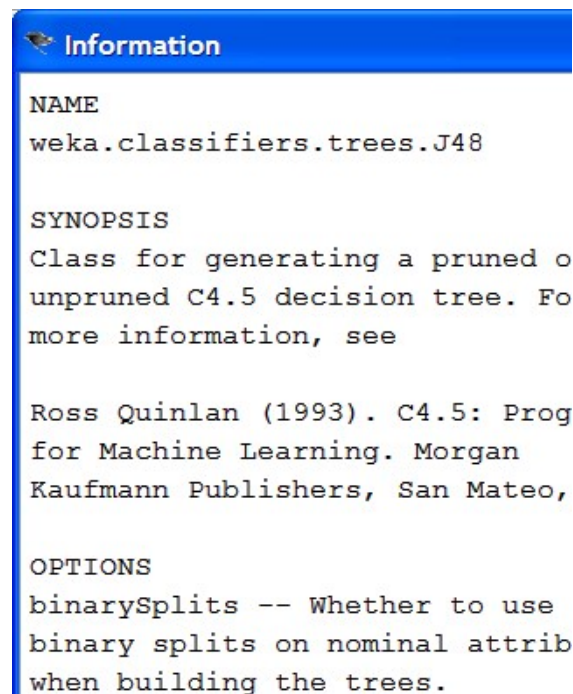
In this particular case, you will see that the resulting unpruned tree classified 91.36% of examples correctly. This is very slightly worse than the pruned tree although I doubt that the difference is statistically significant. However, the unpruned tree is nearly twice as big with 175 nodes of which 121 are leaves.

What do the various learning procedures available in Weka actually do?

So far you have been told that J48 implements a decision tree building procedure. How could you have found this out for yourself and how could you find out what all the procedures available actually do?

The GenericObjectEditor window that you called up by left clicking on 'J48' (see above) contains two buttons. Clicking on these will provide more information about the procedure that J48 implements and its parameters.

Clicking on the 'More' button provides this scrollable window:



```
Information
NAME
weka.classifiers.trees.J48

SYNOPSIS
Class for generating a pruned o
unpruned C4.5 decision tree. Fo
more information, see

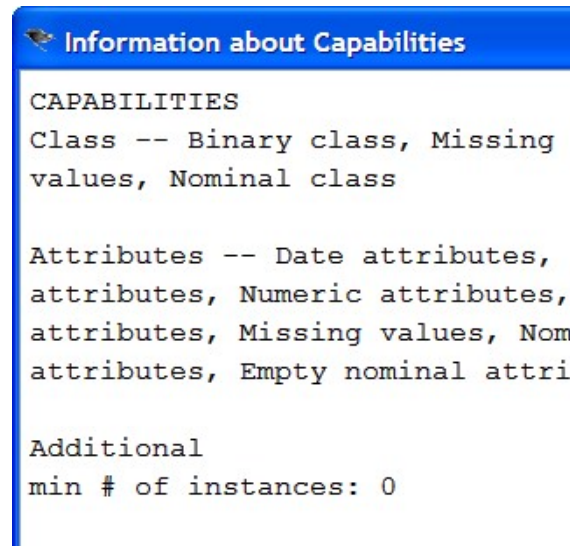
Ross Quinlan (1993). C4.5: Prog
for Machine Learning. Morgan
Kaufmann Publishers, San Mateo,

OPTIONS
binarySplits -- Whether to use
binary splits on nominal attrib
when building the trees.
```

This tells you that J48 is an implementation of Quinlan's C4.5, provides a reference, and then explains the various parameters that the user can set.

The information you are given about the procedures available in Weka varies widely (it depends upon whoever implemented them) but this is typical.

The 'Capabilities' button produces this window:



```
Information about Capabilities
CAPABILITIES
Class -- Binary class, Missing
values, Nominal class

Attributes -- Date attributes,
attributes, Numeric attributes,
attributes, Missing values, Nom
attributes, Empty nominal attri

Additional
min # of instances: 0
```

Essentially this is information about the limitations of the procedure. In this case there are very few. The class can be either binary or nominal and the procedure will even cope with missing class values. A wide range of attribute types can be handled and missing values are acceptable. Somewhat oddly, the program will apparently operate with as few as zero examples!

Running Naïve Bayes on the Soybean data set.

For the next part of this exercise, we will apply a Naïve Bayes classifier to the soybean data set.

Click on 'Choose' in the 'Classify' window of Weka Explorer then select 'NaiveBayes' from the set of classifiers available in the 'bayes' submenu. (There are two other Naïve Bayes programs available; they will produce identical results to NaiveBayes as they differ only in how they handle numeric attributes).

Run this program by clicking the 'Start' button. The results will show that 92.97% of examples are correctly classified by this program. This is slightly better than either of the decision trees but possibly not a statistically significant improvement.

Running k Nearest Neighbour on the Soybean data set.

For the final part of this exercise, we will apply a k nearest neighbour classifier to the soybean data set.

The standard k nearest neighbour method will be found in the 'lazy' submenu of the list presented when you click 'Choose' in Explorer's Classify window. It is called 'IBk'. Select this and then click on IBk so you can modify the parameters. The default value of k is 1. Set it to 3 and then click Start to run the programs.

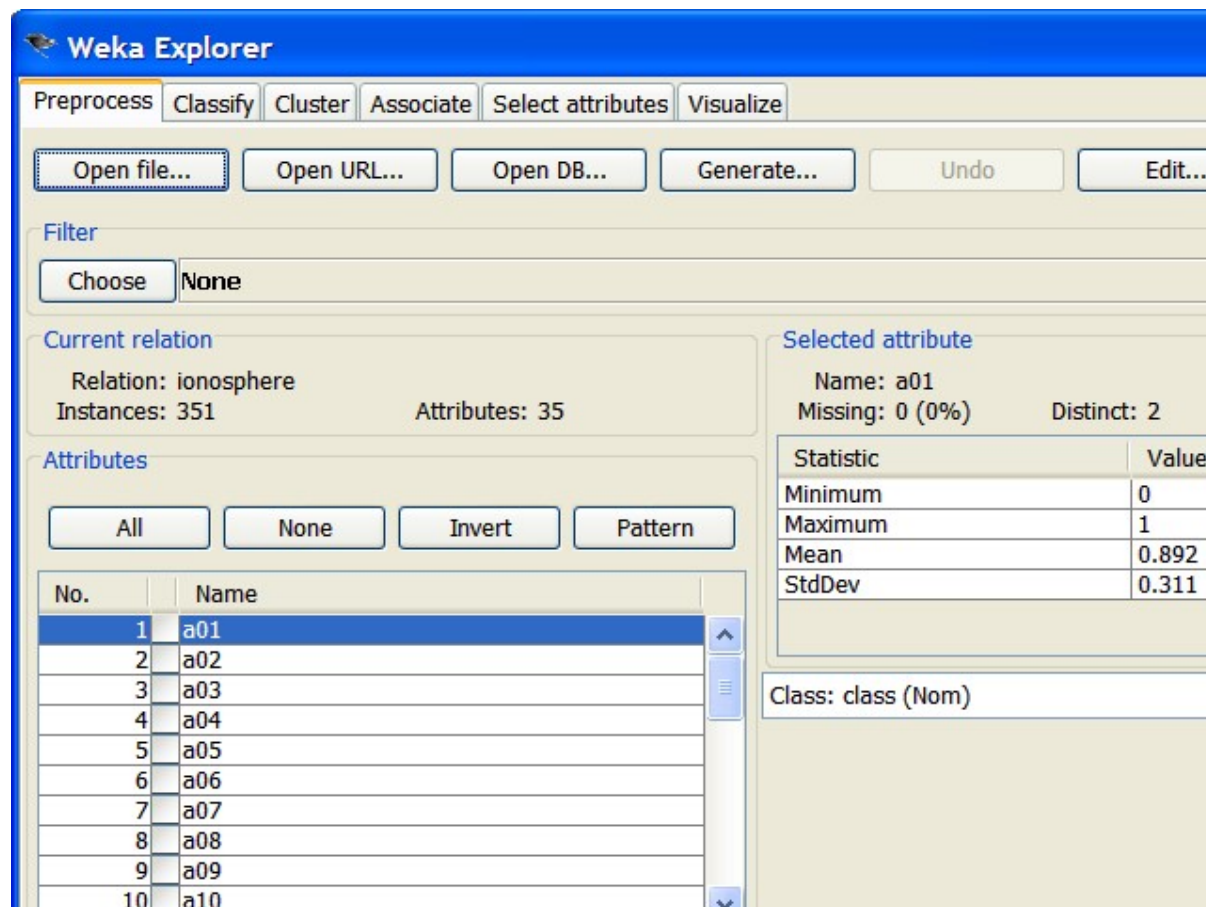
You will see that 91.36% of examples are correctly classified; the same result as with the unpruned decision tree procedure. Try investigating the effect of repeating the run with different values for k.

The Ionosphere Data Set

The first data set for this exercise is another of those supplied when you downloaded the Weka system. It should be found in Weka-3.6/data/ionosphere.arff. It contains 351 examples, each of which has 34 numeric attributes plus the binary class attribute.

Building a Decision Tree

Open Weka Explorer and in the “preprocess” window then use the browser to select the ionosphere data set. The Explorer window should now look like this:



Click the 'Classify' tab and select J48 as the classifier and click the 'Start' button leaving the default of 10 way cross validation unchanged. As usual, the results appear in the scrollable panel on the right. Put them in a separate window, by right clicking on the appropriate item in the window on panel on the lower left headed 'Result list (right-click for options)'. A copy of the results will appear in a separate window which can be enlarged to fill the screen.

The first portion of the output contains the decision tree in indented text format, followed by the count of the number of leaves and nodes. The rest of the output provides a lot of data about the results.

First there is a section that looks like this:

```
Correctly Classified Instances      321          91.453 %
Incorrectly Classified Instances    30           8.547 %
Kappa statistic                    0.8096
Mean absolute error                 0.0938
Root mean squared error             0.2901
Relative absolute error             20.36 %
Root relative squared error         60.4599 %
Total Number of Instances          351
```

Much of this is self-explanatory. The first two lines provide the accuracy and error rate. Then comes the kappa statistic. This is followed by 4 lines that are not particularly useful in a classification task because they are measures used to assess performance when the task is numeric prediction.

The final section of the output provides information about how they performance varies according to the class predicted.

```
=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.825	0.036	0.929	0.825	0.874	0.892	bad
	0.964	0.175	0.908	0.964	0.935	0.892	good
Wtd Avg.	0.915	0.125	0.915	0.915	0.913	0.892	

```
=== Confusion Matrix ===
```

```
  a   b  <-- classified as
104 22 |   a = bad
  8 217 |   b = good
```

The confusion matrix shows that it was rather better at classifying examples of class ‘good’ than class ‘bad’. Only 8 errors were made in 225 examples of class b whereas 22 errors were made in only 126 examples of class a. (N.B. The Weka output is slightly confusing for this data set because one of the classes just happens to be called class ‘b’ – I have relabelled the classes ‘good’ and ‘bad’ to remove this confusion)

This difference is reflected in the Precision and Recall measures for the two classes. The Recall for ‘good’ is high (0.964) reflecting the fact that there were only 8 false negatives: ‘good’ examples classified as ‘bad’. The Precision for ‘good’ is lower (0.908) reflecting the fact that there were 22 false positives: ‘bad’ examples classified as ‘good’.

The Glass Data Set

The ionosphere data set has only two classes so the detailed output is correspondingly simple. The second data set for this exercise is another of those supplied when you downloaded the Weka system. It should be found in Weka-3-9/data/glass.arff. It contains 214 examples, each of which has 10 numeric attributes. These are distributed across 7 classes so a more complex set of results is produced.

Change the Explorer to the ‘PreProcess’ tab and use the browser to select the glass data set. Run J48 to produce a pruned decision tree using the default of 10 way cross-validation.

This is a difficult classification task and the results show an accuracy of only 66.8% and a kappa of 0.55 are achieved. The confusion matrix looks like this:

```
a  b  c  d  e  f  g  <-- classified as
50 15  3  0  0  1  1 | a = build wind float
16 47  6  0  2  3  2 | b = build wind non-float
 5  5  6  0  0  1  0 | c = vehic wind float
 0  0  0  0  0  0  0 | d = vehic wind non-float
 0  2  0  0 10  0  1 | e = containers
 1  1  0  0  0  7  0 | f = tableware
 3  2  0  0  0  1 23 | g = headlamps
```

It is now clear that the poor performance reflects the following difficulties:

1. The system is not very good at distinguishing between two of the classes: ‘build wind float’ and ‘build wind non-float’. This has a major effect because between them, these two classes form 2/3 of the data set.
2. Very poor classification of class ‘vehic wind float’. Only 6 of the 17 examples were classified correctly.

These weaknesses are reflected in the Recall measures for the classes concerned.

Experimenting with WEKA using Attribute Filters

The WEKA toolbox offers several preprocessing algorithms to help determine a best set of attributes and instances for data mining. Here, we present three experiments. The first two experiments illustrate attribute selection. The third concentrates on instance selection.

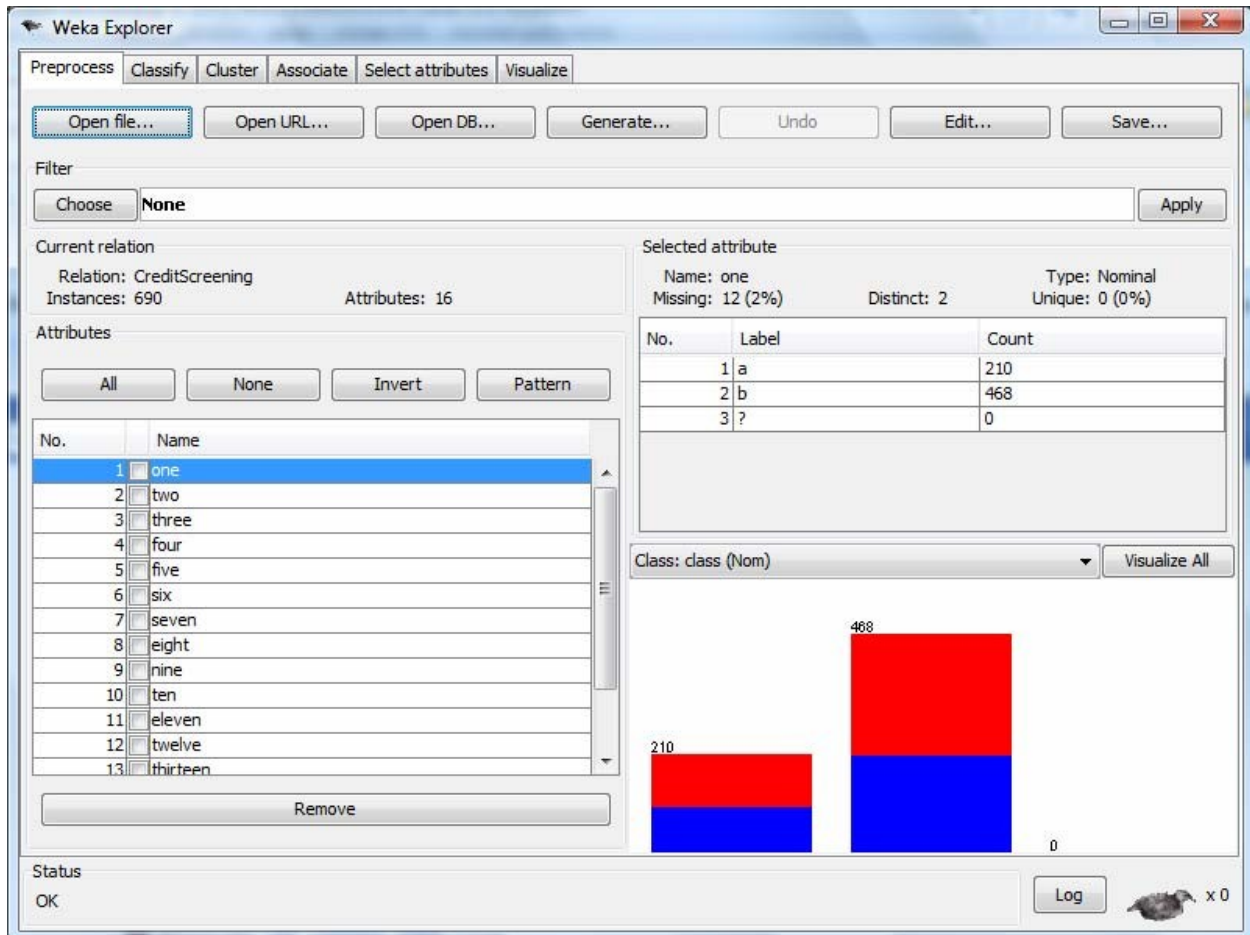
Experiment 1: Attribute Selection - A simple approach

Attempting to build an efficient data model is an exercise in futility when presented with a wealth of irrelevant attributes. It is to our advantage to eliminate most or all irrelevant attributes prior to building a model. We have the option of developing our own attribute selection techniques or using one or more offered by the WEKA toolbox.

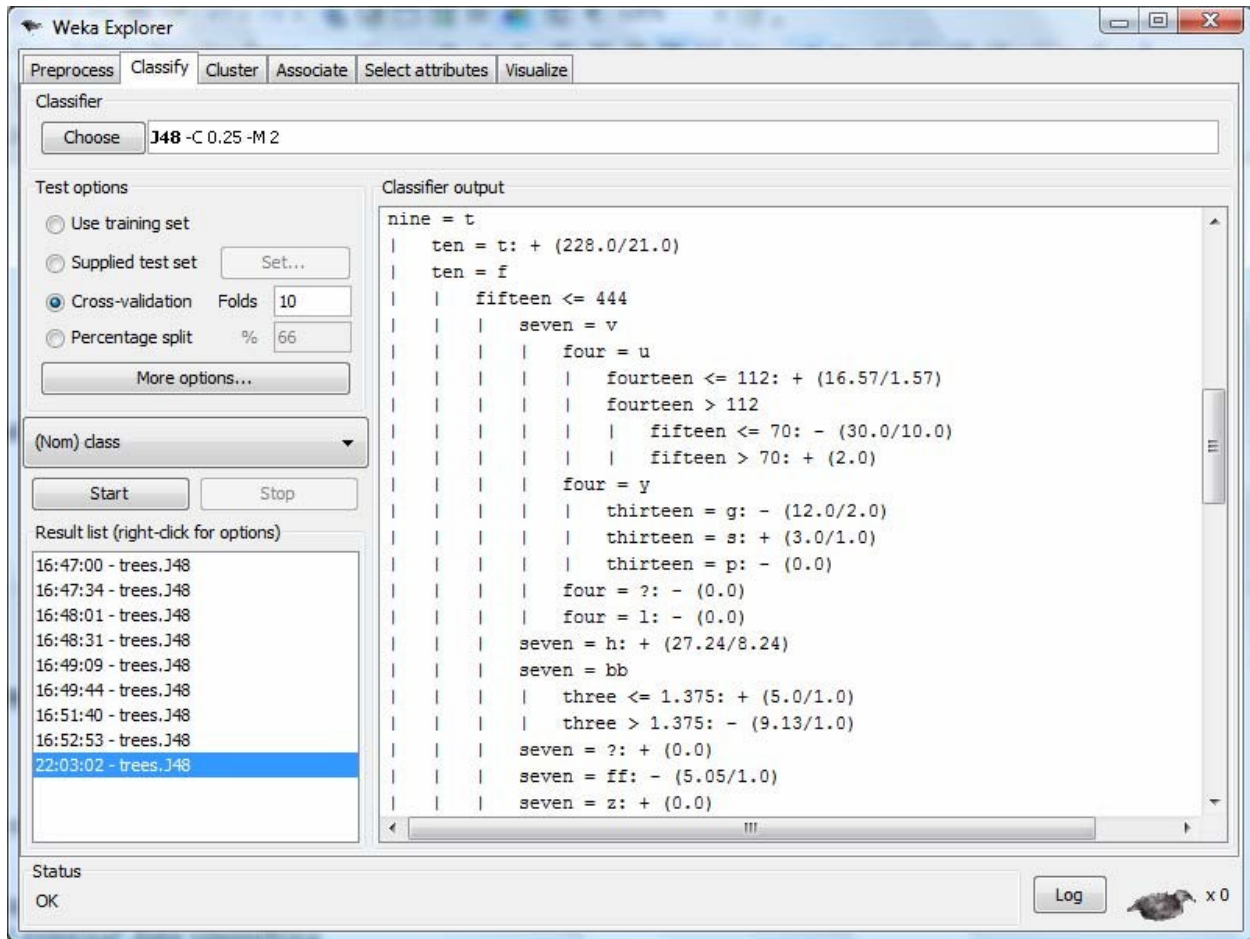
Let’s investigate a simple attribute selection method for supervised learning. Our experiment uses a dataset holding information about individuals who were either accepted or rejected when they applied for a credit card—see the *The Credit Card Screening Dataset* description box. The dataset contains 690 instances, 307 of which represent individuals who were approved to receive a credit card. The remaining 383 individuals had their credit card application rejected. We want to decide on a best set of attributes defining the classes contained in the data. Stated another way, we wish to test the possibility of building an accurate supervised learner model with a subset of attributes taken from the data.

The attributes and values have been mapped to a set of meaningless symbols to protect the confidentiality of the data. However, because the mapping is consistent, we should be able to apply data mining to analyze the dataset.

As a first step, open the WEKA explorer and load the dataset. Your screen will appear similar to the one given below:



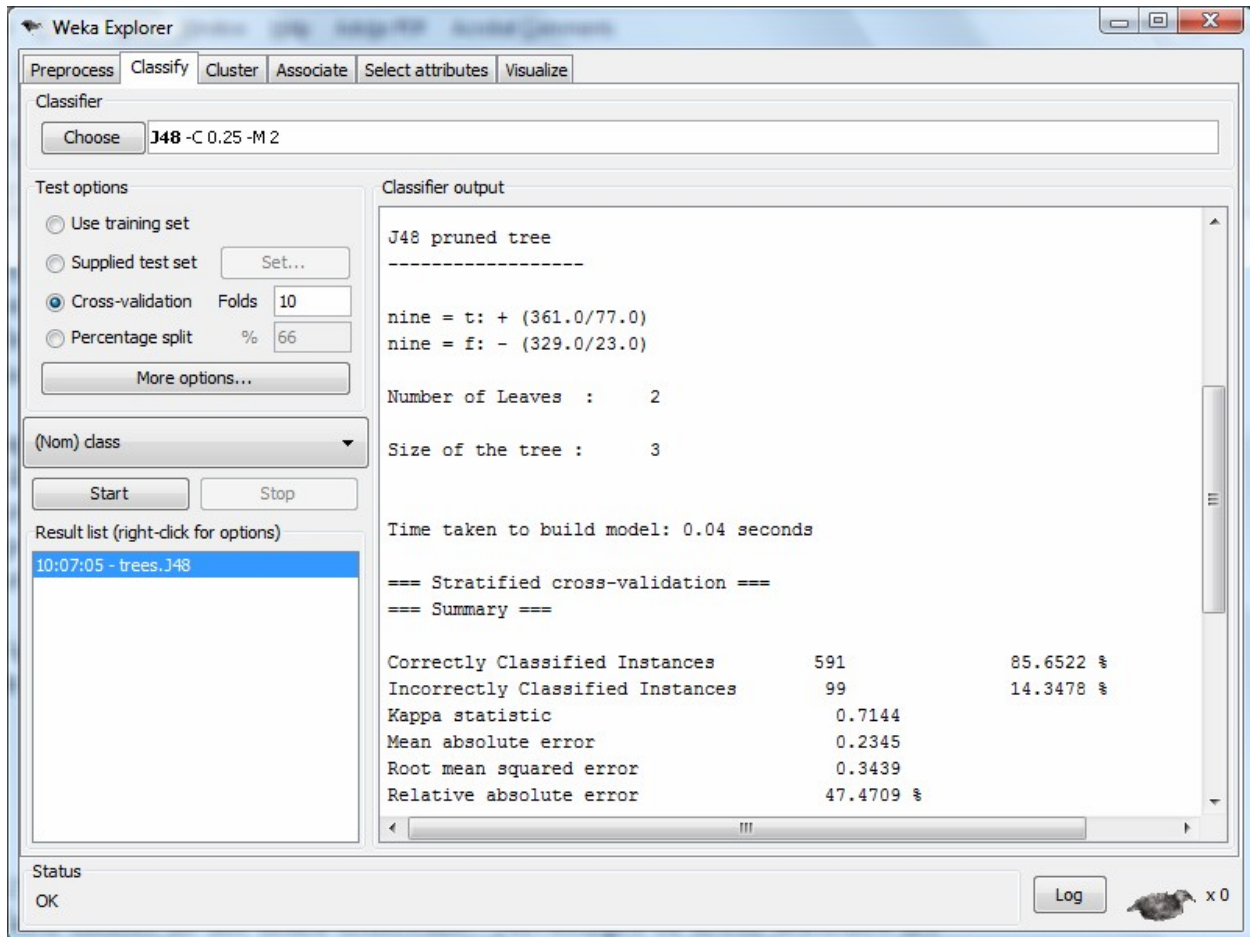
Notice that we are in preprocess mode with *none* displayed in the filter command line. If we scroll the attribute window, we will see the last attribute designated as *class*. To perform our experiment, we first mine the data with J48 using all sixteen input attributes and *class* as the output attribute. The screen containing the resultant decision tree follows:



Notice that attribute *nine* represents the top-level node of the decision tree. Upon scrolling through the tree, we observe a classification accuracy of 84.7826%. Let's see if we can do as well or better with fewer attributes. Using fewer attributes has several advantages two of which are building a simpler model in less time.

One straightforward approach to attribute selection uses a best-first strategy where at each iteration we build a decision tree, then record and remove the top-level attribute. For our example we first remove attribute *nine* and proceed to build a new decision tree. The second iteration shows *eleven* as the top-level node. Continuing, the next iteration shows *ten* as the next top-level node followed by *fifteen*. Let's stop the process here.

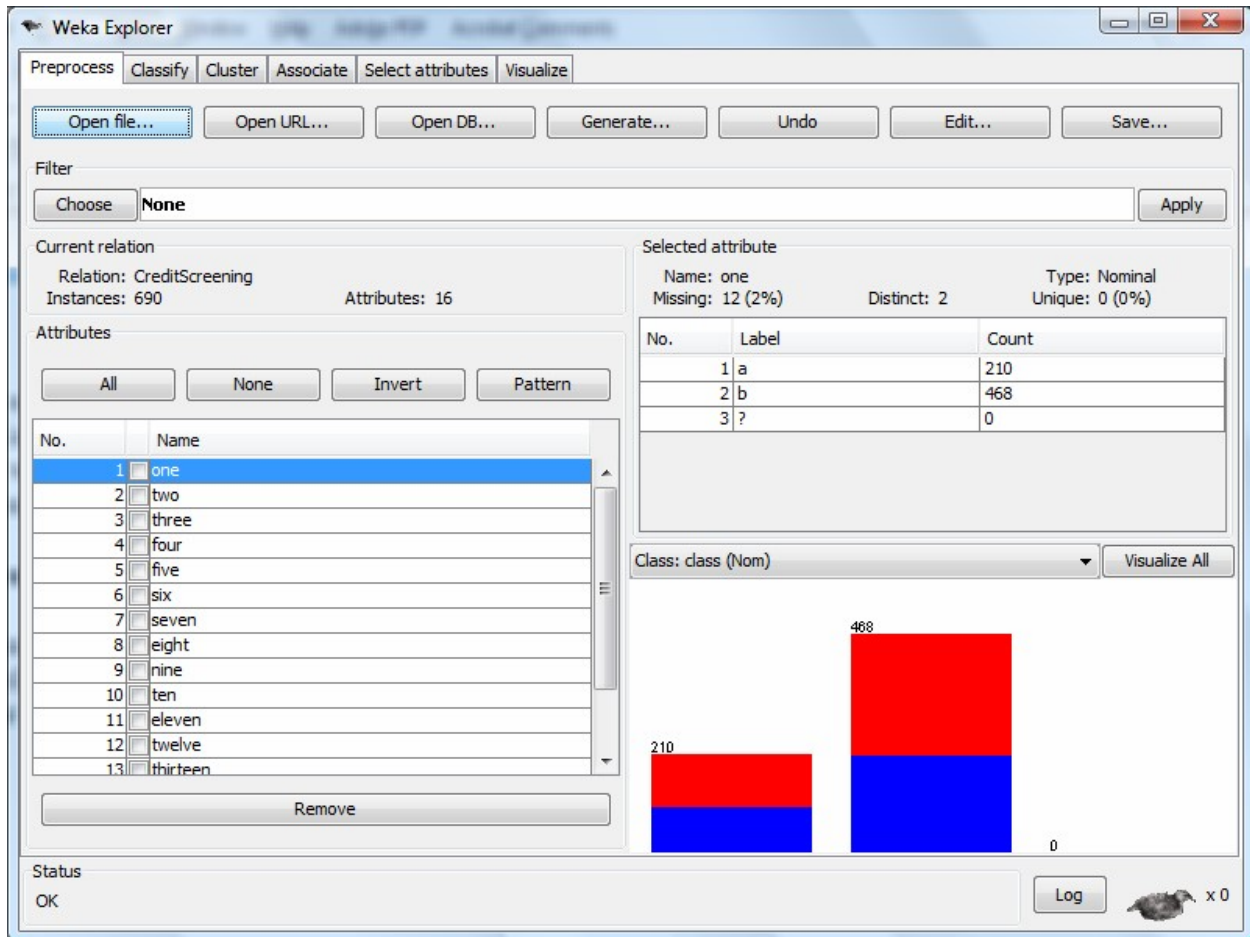
Next, reload the original dataset, remove all attributes except for attributes *nine*, *ten*, *eleven*, *fifteen* and the *class* attribute. Next, we invoke J48 to build a decision tree. Our tree appears as below:



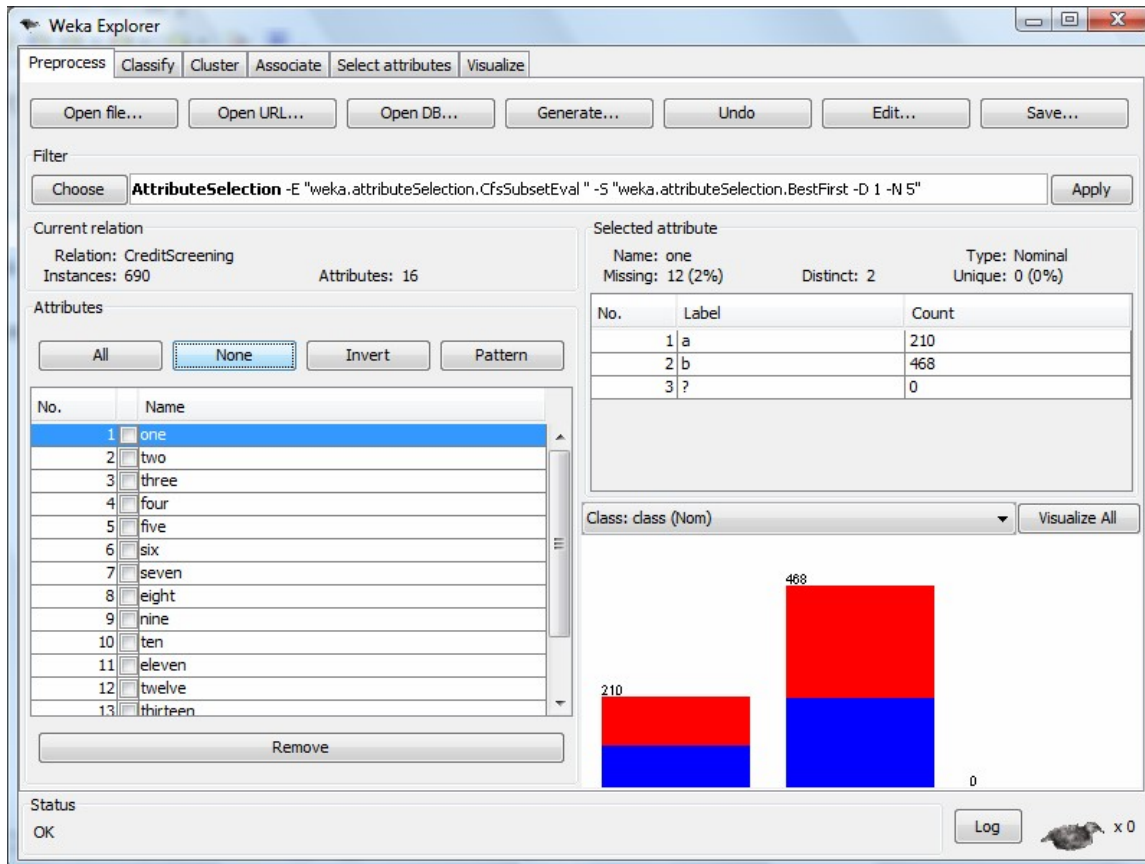
Notice that attribute *nine* is the only attribute making up the tree. The decision tree accuracy is 85.65%. This result is not significantly better than our original result using all attributes. However, we see that we are able to build a model as accurate as our original model using just one attribute. What conclusions might we reach from this result?

Attribute Selection using a WEKA filter.

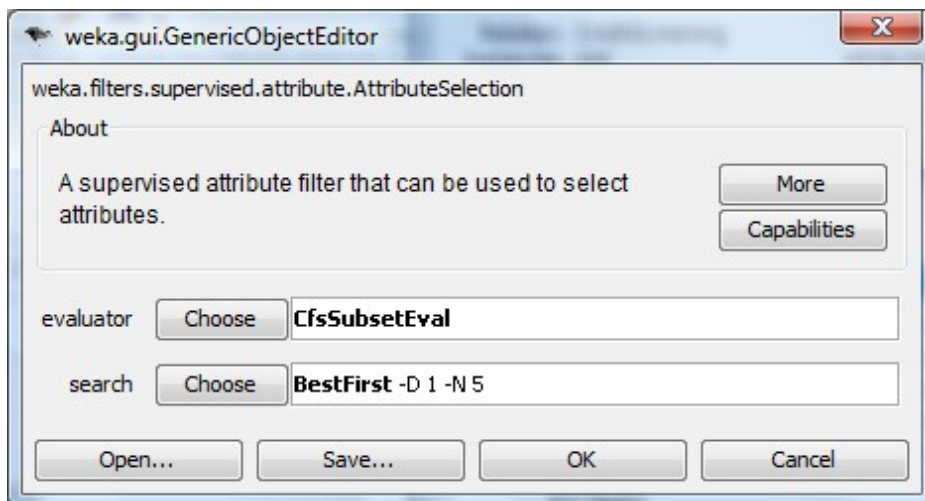
For the second experiment we use the same dataset but this time the attribute selection process takes place with the help of a WEKA preprocessing filter. Once again, load the *Credit Card Screening Dataset*. Our screen appears as follows:



To invoke one of WEKA's attribute filters, we click on *choose* followed by *filters- supervised- attribute-attribute selection*. The resultant screen is given below:



Clearly, the command line showing the parameter list for the attribute selection filter is complex. We can see more filter options by simply clicking on the white space area on the command line. Doing so, we see the following:



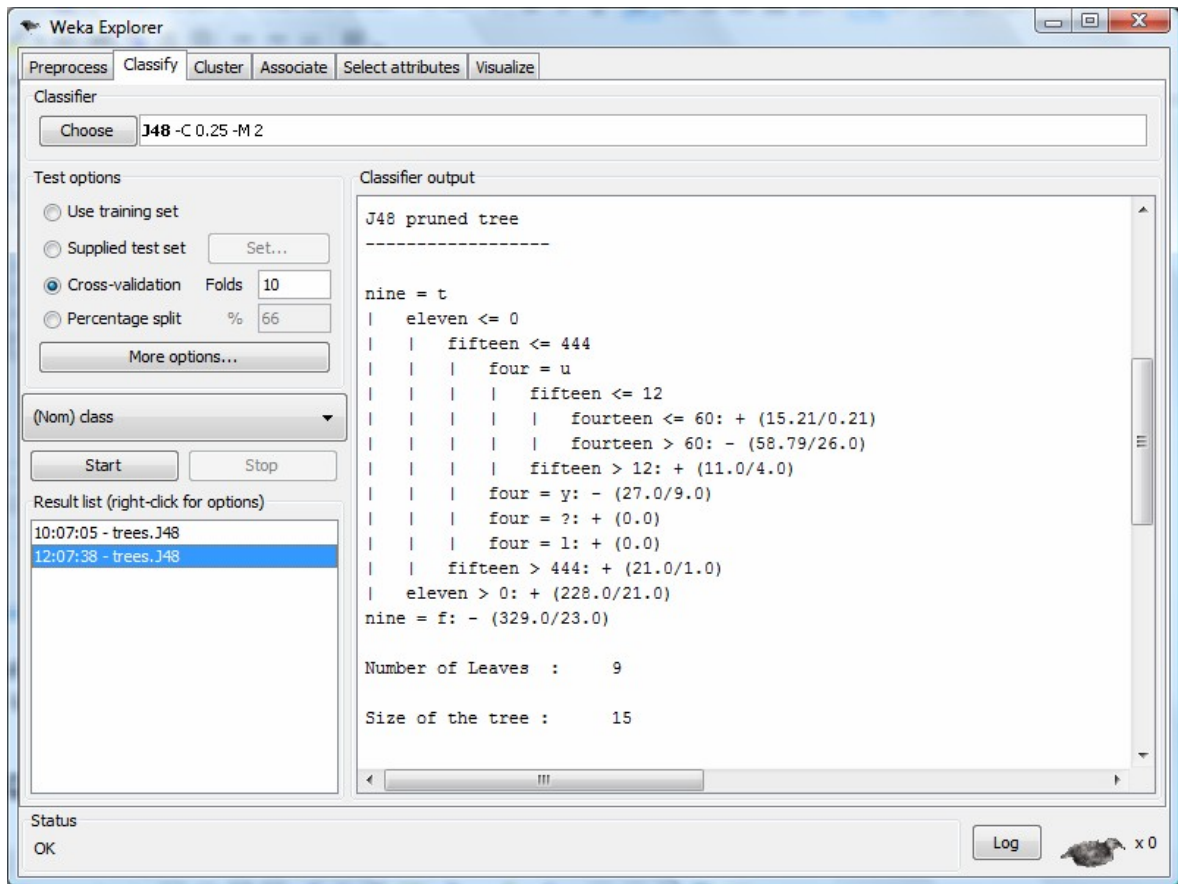
For our experiment, we will use the default values shown. Close the above window and click on apply. The chosen attributes are given in the window below:

The screenshot shows the Weka Explorer interface with the 'AttributeSelection' filter applied. The 'Current relation' is 'CreditsScreening-weka.filters.supervised.attribute.Attribute...' with 690 instances and 8 attributes. The 'Selected attribute' is 'four', which is a nominal attribute with 3 distinct values and 6 missing values (1%). The 'Attributes' list shows 8 attributes: 'four', 'six', 'eight', 'nine', 'eleven', 'fourteen', 'fifteen', and 'class'. The 'Class' is set to 'class (Nom)'. A bar chart visualizes the distribution of the 'four' attribute, showing counts for labels 'u' (519), 'y' (163), and '?' (0).

No.	Label	Count
1	u	519
2	y	163
3	?	0
4	l	2

Notice that nine of the sixteen input attributes have been removed. Also, three of the four attributes initially chosen in the previous experiment are included.

Next, we invoke J48 to perform a data mining session. The outcome is displayed in the screen below:



Attribute *nine* is the top-level node. Four additional attributes make up the remainder of the tree. Scrolling, we see the cross-validation shows an accuracy of 84.9275. We conclude that mining the data with a subset of the attribute set provides a model as accurate as the model developed using all sixteen original attributes.