LAB ASSIGNMENT - 8

8(a)

In this lab, we will use nearest-neighbour classification and decision tree learning. For most of it, a real-world forensic glass classification dataset is used. We begin by taking a preliminary look at the dataset. Then we examine the effect of selecting different attributes for nearest-neighbour classification. Next, we study class noise and its impact on predictive performance for the nearest-neighbour method. Following that we vary the training set size, both for nearest-neighbour classification and for decision tree learning.

Some aspects of the classification task:

• How is the accuracy of a classifier measured?

• To make a good classifier, are all the attributes necessary?

• What is class noise, and how would you measure its effect on learning?

The Glass Dataset The glass dataset glass.arff from the U.S. Forensic Science Service contains data on six types of glass. Glass is described by its refractive index and the chemical elements that it contains; the aim is to classify different types of glass based on these features. This dataset is taken from the UCI datasets, which have been collected by the University of California at Irvine and are freely available on the Web. They are often used as a benchmark for comparing data mining algorithms. Find the dataset **glass.arff** and load it into the Explorer interface.

Answer the following questions.

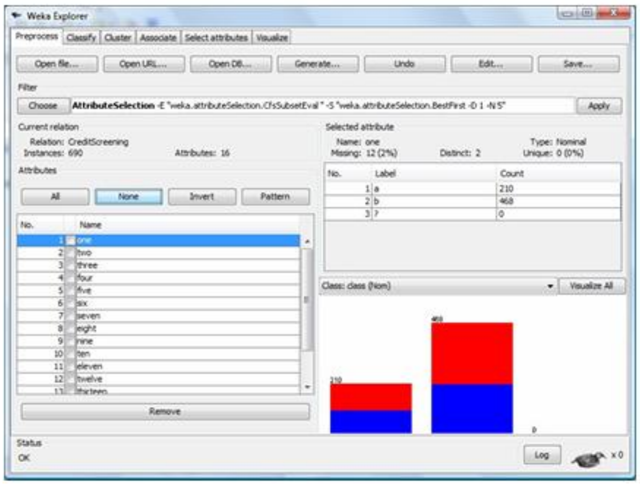
1. **How many attributes are there in the dataset? What are their names? What is the class attribute?**

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 neighbouring instances to use when classifying.

2. **What is the accuracy of IBk (given in the Classifier Output box)? Run IBk again, but increase the number of neighbouring instances to k = 5 by entering this value in the KNN field. Continue to use cross-validation as the evaluation method.**
3. **What is the accuracy of IBk with five neighbouring instances (k = 5)?**

Attribute Selection

Now we investigate which subset of attributes produces the best cross-validated classification accuracy for the decision tree and IBk algorithm on the glass dataset. Weka contains automated attribute selection facilities. 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:



Filter

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

Current relation

Relation: CreditScreening
Instances: 690   Attributes: 16

Selected attribute

Name: one   Type: Nominal
Missing: 12 (2%)   Distinct: 2   Unique: 0 (0%)

No.	Label	Count
1	a	210
2	b	468
3	?	0

Attributes

No.	Name
1	one
2	two
3	three
4	four
5	five
6	six
7	seven
8	eight
9	nine
10	ten
11	eleven
12	twelve
13	thirteen

Class: class (Nom)

Status
OK

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. We will use the default values shown. Close the above window and click on apply. The chosen attributes will be shown in the window.

Next, we invoke J48 and Ibk to perform a data mining. Note the output in each case.

4. **Can we conclude that mining the data with a subset of the attribute set provides a model as accurate as the model developed using all of the original attributes.**

Attribute selection can also be done manually by applying the backward elimination procedure. To do this, first consider dropping each attribute individually from the full dataset, and run a cross-validation for each reduced version. Once you have determined the best eight-attribute dataset, repeat the procedure with this reduced dataset to find the best seven-attribute dataset, and so on. (You may use "Select Attributes" option with Ranker method.)

5. **Record in a Table, the best attribute set and the greatest accuracy obtained in each iteration. The best accuracy obtained in this process is quite a bit higher than the accuracy obtained on the full dataset.**

Accuracy Obtained Using *IBk*, for Different Attribute Subsets

Subset Size (No. of Attributes)	Attributes in "Best" Subset	Classification Accuracy
9		
8		
7		
6		
5		
4		
3		
2		
1		
0		

Class Noise and Nearest-Neighbour Learning

Nearest-neighbour learning, like other techniques, is sensitive to noise in the training data. In this experiment, we inject varying amounts of class noise into the data and observe the effect on classification performance.

You can flip a certain percentage of class labels in the data to a randomly chosen other value using an unsupervised attribute filter called **AddNoise**, in weka. **filters.unsupervised.attribute**. However, for this experiment it is important that the test data remains unaffected by class noise. Filtering the training data without filtering the test data is a common requirement, and is achieved using a **metalearner** called **FilteredClassifier**, in weka.classifiers.meta. This metalearner should be configured to use **IBk as the classifier** and **AddNoise as the filter**. FilteredClassifier applies the filter to the data before running the learning algorithm. This is done in two batches: first the training data and then the test data. The AddNoise filter only adds noise to the first batch of data it encounters, which means that the test data passes through unchanged.

6. **Record in a Table, the cross-validated accuracy estimate of IBk for 6 different percentages of class noise and neighbourhood sizes k = 1, k = 3, k = 5 (determined by the value of k in the k-nearest-neighbour classifier).**

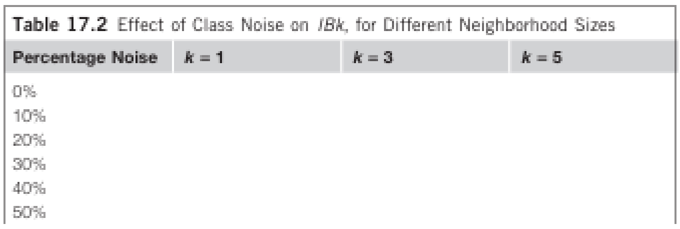


Table 17.2 Effect of Class Noise on *IBk*, for Different Neighborhood Sizes

Percentage Noise	*k* = 1	*k* = 3	*k* = 5
0%			
10%			
20%			
30%			
40%			
50%			

7. **What is the effect of increasing the amount of class noise?**
8. **What is the effect of altering the value of k?**

Varying the Amount of Training Data

Here, we examine the effect of gradually increasing the amount of training data. Again, we use the glass data, but this time with both **IBk and the C4.5 decision tree learners**, implemented in Weka as J48. Use **FilteredClassifier** again, this time in conjunction with **weka.filters.unsupervised.instance.Resample**, which extracts a certain specified percentage of a given dataset and returns the reduced dataset. (This filter performs sampling with replacement, rather than sampling without replacement). Again, this is done only for the first batch to which the filter is applied, so the test data passes unmodified through the FilteredClassifier before it reaches the classifier.

9. **Record in a Table, the data for learning curves for both the one-nearest-neighbour classifier (i.e., IBk with k = 1) and J48.**

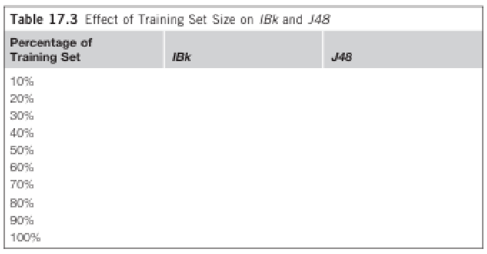


Table 17.3 Effect of Training Set Size on *IBk* and *J48*

Percentage of Training Set	*IBk*	*J48*
10%		
20%		
30%		
40%		
50%		
60%		
70%		
80%		
90%		
100%		

10. **What is the effect of increasing the amount of training data?**
11. **Is this effect more pronounced for IBk or J48?**

8(b)

In this exercise we will build a **decision tree** and also apply **Naïve Bayes** and **k nearest neighbour** methods to the same data set. In the course of doing this, we will find out how to discover what each of the learning procedures provided by Weka actually does and how their parameters may be modified.

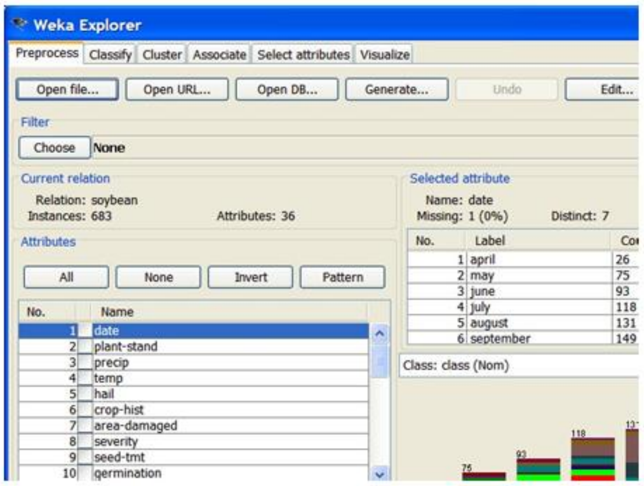
The Soybean Diagnosis Data Set

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:



Filter: None

Relation: soybean
Instances: 683
Attributes: 36

No.	Name
1	date
2	plant-stand
3	precip
4	temp
5	hail
6	crop-hist
7	area-damaged
8	severity
9	seed-tmt
10	germination

Name: date
Missing: 1 (0%)
Distinct: 7

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

Class: class (Nom)

Next, click the 'Classify' tab and **select J48** as the classifier and 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.

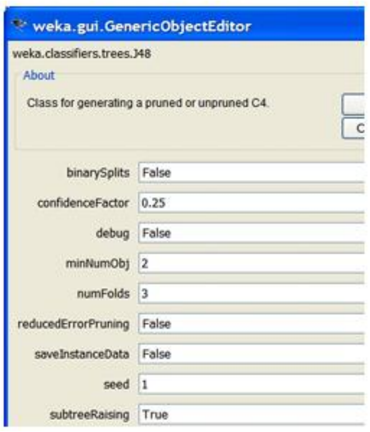
Fortunately, the tree is more intelligible in the separate results window you have just created. Here it is presented in indented text format just after the heading ‘J48 pruned tree’. At the end of the tree is the information that it contains **93 nodes** of which **61 are leaves**. The rest of the output 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:



weka.gui.GenericObjectEditor

weka.classifiers.trees.J48

About

Class for generating a pruned or unpruned C4.

```
          binarySplits  False
      confidenceFactor  0.25
                 debug  False
             minNumObj  2
              numFolds  3
 reducedErrorPruning  False
      saveInstanceData  False
                  seed  1
        subtreeRaising  True
```

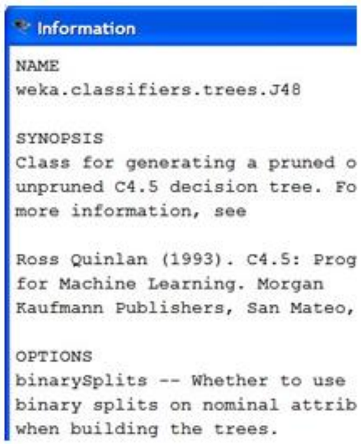
Change ‘unpruned’ in this window to ‘True’, click OK, and then run the program again. The output for this new run replaces the output from your earlier run in the scrollable panel of the Explorer window. However, your earlier results are still there in the separate window you created. You can make another 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. 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 learnt 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?

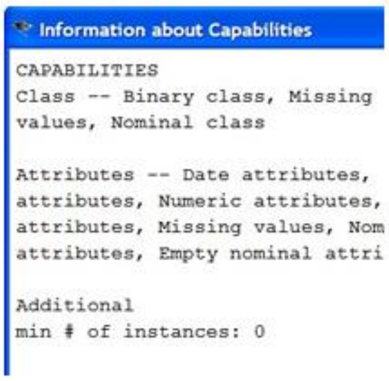
The GenericObjectEditor window that you called up by left clicking on ‘J48’ contains two buttons. Clicking on these will provide more information about the procedure that J48 implements and its parameters. Clicking on the ‘More’ button provides this scrollable window:


```
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.

Running Naïve Bayes on the Soybean data set.

Now, 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.

Now, 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.

<u>Try investigating the effect of repeating the run with different values for k.</u>

The Ionosphere Data Set

This dataset 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**.

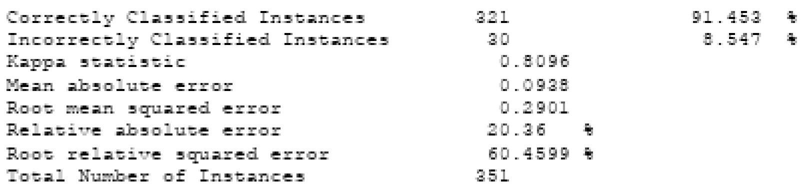
Building a Decision Tree

Open Weka Explorer and in the “preprocess” window then use the browser to select the ionosphere data set.

Click the ’Classify’ tab and select J48 as the classifier an 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.

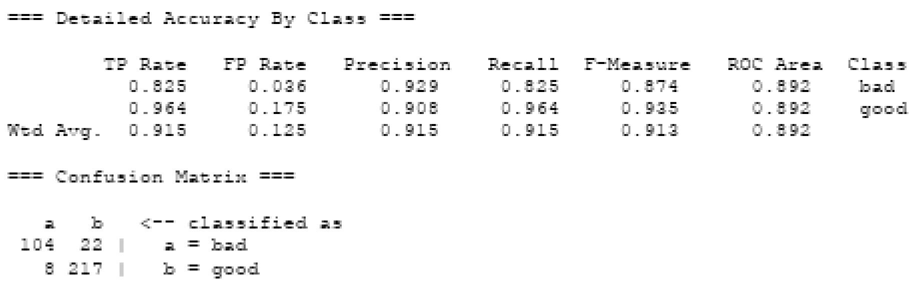
First there is a section that looks like this:



```
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. 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.**

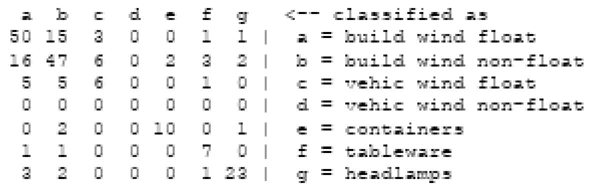
This difference is reflected in the Precision and Recall measures for the two classes. The **Recall** for ‘good’ is high **(0.964)** reflecting the fact that there were only 8 false negatives: ‘good’ examples classified as ‘bad’. The **Precision** for ‘good’ is lower **(0.908)** reflecting the fact that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