CHAPTER

The Experimenter 13

The Explorer and Knowledge Flow environments help you determine how well machine learning schemes perform on given datasets. But serious investigative work involves substantial experiments—typically running several learning schemes on different datasets, often with various parameter settings—and these interfaces are not really suitable for this. The Experimenter enables you to set up large-scale experiments, start them running, leave them and come back when they have finished, and then analyze the performance statistics that have been collected. They automate the experimental process. The statistics can be stored in ARFF format, and can themselves be the subject of further data mining. You invoke this interface by selecting *Experimenter* from the choices in the right panel in Figure 11.3(a).

Whereas the Knowledge Flow transcends limitations of space by allowing machine learning runs that do not load in the whole dataset at once, the Experimenter transcends limitations of time. It contains facilities for advanced users to distribute the computing load across multiple machines using Java RMI. You can set up big experiments and just leave them to run.

13.1 GETTING STARTED

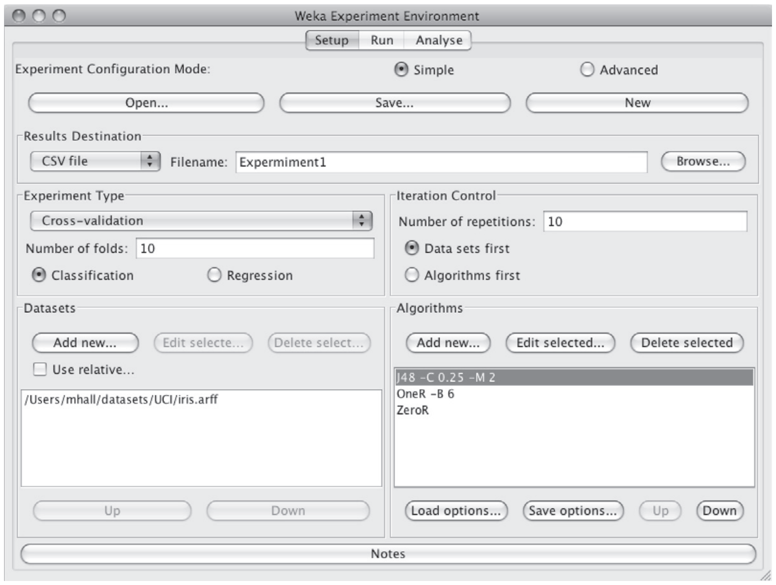
As an example, we will compare the *J48* decision tree method with the baseline methods *OneR* and *ZeroR* on the iris dataset. The Experimenter has three panels: *Setup*, *Run*, and *Analyze*. Figure 13.1(a) shows the first: You select the others from the tabs at the top. Here, the experiment has already been set up. To do this, first click *New* (toward the right at the top) to start a new experiment (the other two buttons in that row save an experiment and open a previously saved one). Then, on the line below, select the destination for the results—in this case the file *Experiment1*—and choose *CSV file*. Underneath, select the datasets—we have only one, the iris data. To the right of the datasets, select the algorithms to be tested—we have three. Click *Add new* to get a standard Weka object editor from which you can choose and configure a classifier. Repeat this operation to add the three classifiers. Now the experiment is ready.

The other settings in Figure 13.1(a) are all default values. If you want to reconfigure a classifier that is already in the list, you can use the *Edit selected* button. You can also save the options for a particular classifier in XML format for later reuse. You can right-click on an entry to copy the configuration to the clipboard, and add or enter a configuration from the clipboard.

Running an Experiment

To run the experiment, click the *Run* tab, which brings up a panel that contains a *Start* button (and little else); click it. A brief report is displayed when the operation is finished. The file *Experiment1.csv* contains the results. The first two lines are shown in Figure 13.1(b): They are in CSV format and can be read directly into a spreadsheet, the first part of which appears in Figure 13.1(c). Each row represents one fold of a tenfold cross-validation (see the *Fold* column). The cross-validation is run 10 times (the *Run* column) for each classifier (the *Scheme* column). Thus, the file contains 100 rows for each classifier, which makes 300 rows in all (plus the header row). Each row contains plenty of information, including the options supplied to the machine learning scheme; the number of training and test instances; the number (and percentage) of correct, incorrect, and unclassified instances; the mean absolute error and the root mean-squared error; and much more.

There is a great deal of information in the spreadsheet, but it is hard to digest. In particular, it is not easy to answer the question posed previously: How does *J48*


Weka Experiment Environment

Setup | Run | Analyse

Experiment Configuration Mode: [✓ Simple] Advanced

Results Destination: CSV file — Filename: Expermiment1

Experiment Type: Cross-validation
Number of folds: 10
[✓ Classification] Regression

Iteration Control:
Number of repetitions: 10
[✓ Data sets first] Algorithms first

Datasets:
Use relative...
/Users/mhall/datasets/UCI/iris.arff

Algorithms:
J48 -C 0.25 -M 2
OneR -B 6
ZeroR

(a)

FIGURE 13.1

An experiment: (a) setting it up, (b) results file, and (c) spreadsheet with results.

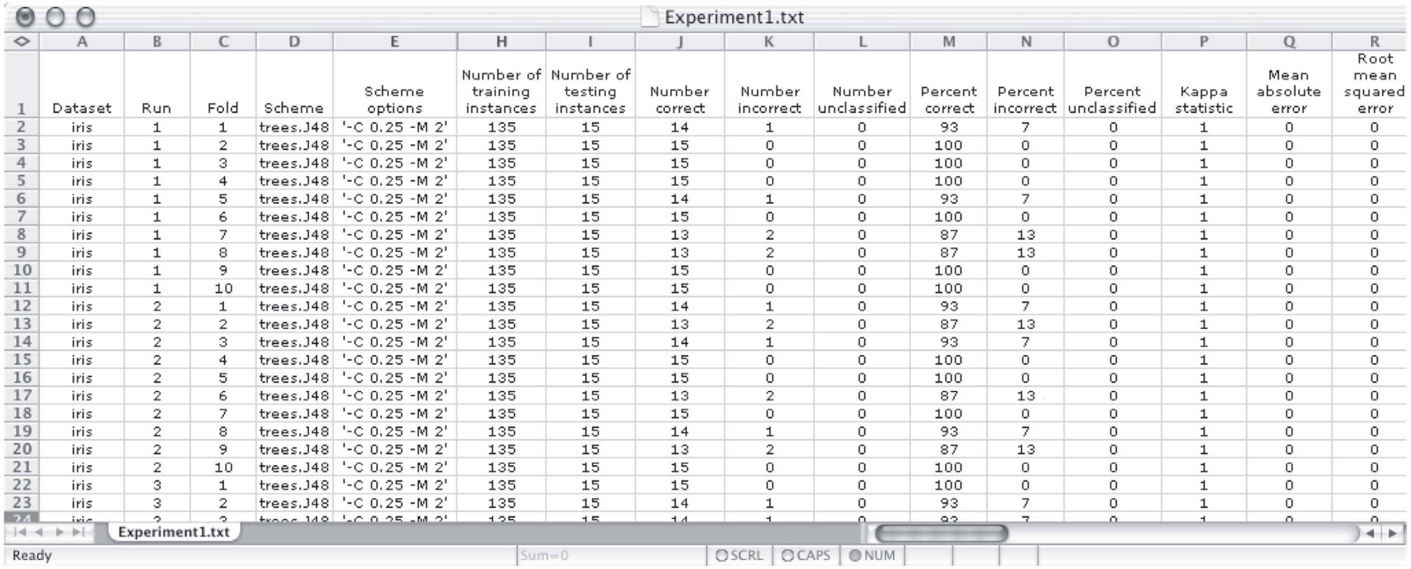
```
Dataset,Run,Fold,Scheme,Scheme_options,Scheme_version_ID,Date_time,Number_of_training_instances,Number_
of_testing_instances,Number_correct,Number_incorrect,Number_unclassified,Percent_correct,Percent_incorr
ect,Percent_unclassified,Kappa_statistic,Mean_absolute_error,Root_mean_squared_error,Relative_absolute_
error,Root_relative_squared_error,SF_prior_entropy,SF_scheme_entropy,SF_entropy_gain,SF_mean_prior_entr
opy,SF_mean_scheme_entropy,SF_mean_entropy_gain,KB_information,KB_mean_information,KB_relative_informat
ion,True_positive_rate,Num_true_positives,False_positive_rate,Num_false_positives,True_negative_rate,Nu
m_true_negatives,False_negative_rate,Num_false_negatives,IR_precision,IR_recall,F_measure,Area_under_RO
C,Weighted_avg_true_positive_rate,Weighted_avg_false_positive_rate,Weighted_avg_true_negative_rate,Weig
hted_avg_false_negative_rate,Weighted_avg_IR_precision,Weighted_avg_IR_recall,Weighted_avg_F_measure,We
ighted_avg_area_under_ROC,Elapsed_Time_training,Elapsed_Time_testing,UserCPU_Time_training,UserCPU_Time
_testing,Serialized_Model_Size,Serialized_Train_Set_Size,Serialized_Test_Set_Size,Summary,measureTreeSi
ze,measureNumLeaves,measureNumRules

iris,1,1,weka.classifiers.trees.J48,'-C 0.25 -M 2',-
217733168393644444,2.01003030223E7,135.0,15.0,14.0,1.0,0.0,93.33333333333333,6.666666666666667,0.0,0.9,
0.0450160137965016,0.1693176548766098,10.128603104212857,35.917698581356284,23.77443751081735,2.6327150
99281766,21.141722411535582,1.5849625007211567,0.17551433995211774,1.4094481607690388,21.61565359986799
4,1.4410435733245328,1363.79589990507,1.0,5.0,0.0,0.0,1.0,10.0,0.0,0.0,1.0,1.0,1.0,1.0,0.93333333333333
33,0.03333333333333333,0.9666666666666667,0.06666666666666667,0.9444444444444445,0.9333333333333333,0.9
326599326599326,1.0,0.059,0.0050,0.024114,0.001788,4449.0,11071.0,2791.0,'Number of leaves: 4\nSize of
the tree: 7\n',7.0,4.0,4.0
```

(b)

FIGURE 13.1, cont'd

Continued



Experiment1.txt

	A	B	C	D	E	H	I	J	K	L	M	N	O	P	Q	R
1	Dataset	Run	Fold	Scheme	Scheme options	Number of training instances	Number of testing instances	Number correct	Number incorrect	Number unclassified	Percent correct	Percent incorrect	Percent unclassified	Kappa statistic	Mean absolute error	Root mean squared error
2	iris	1	1	trees.J48	'-C 0.25 -M 2'	135	15	14	1	0	93	7	0	1	0	0
3	iris	1	2	trees.J48	'-C 0.25 -M 2'	135	15	15	0	0	100	0	0	1	0	0
4	iris	1	3	trees.J48	'-C 0.25 -M 2'	135	15	15	0	0	100	0	0	1	0	0
5	iris	1	4	trees.J48	'-C 0.25 -M 2'	135	15	15	0	0	100	0	0	1	0	0
6	iris	1	5	trees.J48	'-C 0.25 -M 2'	135	15	14	1	0	93	7	0	1	0	0
7	iris	1	6	trees.J48	'-C 0.25 -M 2'	135	15	15	0	0	100	0	0	1	0	0
8	iris	1	7	trees.J48	'-C 0.25 -M 2'	135	15	13	2	0	87	13	0	1	0	0
9	iris	1	8	trees.J48	'-C 0.25 -M 2'	135	15	13	2	0	87	13	0	1	0	0
10	iris	1	9	trees.J48	'-C 0.25 -M 2'	135	15	15	0	0	100	0	0	1	0	0
11	iris	1	10	trees.J48	'-C 0.25 -M 2'	135	15	15	0	0	100	0	0	1	0	0
12	iris	2	1	trees.J48	'-C 0.25 -M 2'	135	15	14	1	0	93	7	0	1	0	0
13	iris	2	2	trees.J48	'-C 0.25 -M 2'	135	15	13	2	0	87	13	0	1	0	0
14	iris	2	3	trees.J48	'-C 0.25 -M 2'	135	15	14	1	0	93	7	0	1	0	0
15	iris	2	4	trees.J48	'-C 0.25 -M 2'	135	15	15	0	0	100	0	0	1	0	0
16	iris	2	5	trees.J48	'-C 0.25 -M 2'	135	15	15	0	0	100	0	0	1	0	0
17	iris	2	6	trees.J48	'-C 0.25 -M 2'	135	15	13	2	0	87	13	0	1	0	0
18	iris	2	7	trees.J48	'-C 0.25 -M 2'	135	15	15	0	0	100	0	0	1	0	0
19	iris	2	8	trees.J48	'-C 0.25 -M 2'	135	15	14	1	0	93	7	0	1	0	0
20	iris	2	9	trees.J48	'-C 0.25 -M 2'	135	15	13	2	0	87	13	0	1	0	0
21	iris	2	10	trees.J48	'-C 0.25 -M 2'	135	15	15	0	0	100	0	0	1	0	0
22	iris	3	1	trees.J48	'-C 0.25 -M 2'	135	15	15	0	0	100	0	0	1	0	0
23	iris	3	2	trees.J48	'-C 0.25 -M 2'	135	15	14	1	0	93	7	0	1	0	0

(c)

FIGURE 13.1, cont'd

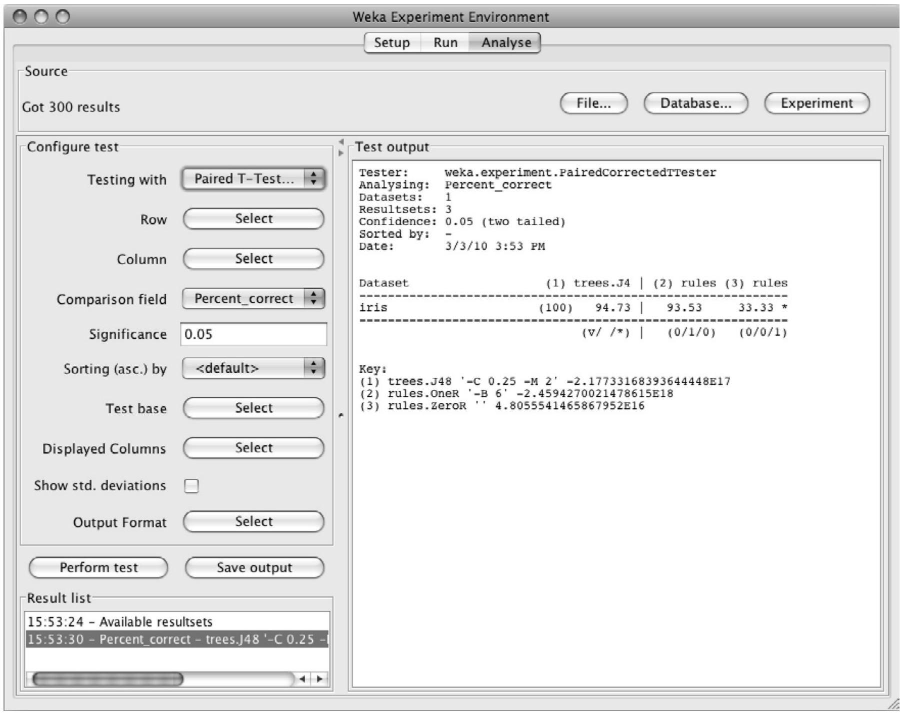
compare with the baseline methods *OneR* and *ZeroR* on this dataset? For that we need the *Analyze* panel.

Analyzing the Results

The reason that we generated the output in CSV format was to show the spreadsheet in Figure 13.1(c). The Experimenter normally produces its output in ARFF format. You can also leave the file name blank, in which case the Experimenter stores the results in a temporary file.

The *Analyze* panel is shown in Figure 13.2. To analyze the experiment that has just been performed, click the *Experiment* button at the top right; otherwise, supply a file that contains the results of another experiment. Then click *Perform test* (near the bottom left). The result of a statistical significance test of the performance of the first learning scheme (*J48*) versus the other two (*OneR* and *ZeroR*) will be displayed in the large panel on the right.

We are comparing the percent correct statistic: This is selected by default as the comparison field shown toward the left in Figure 13.2. The three methods are



Got 300 results

Testing with: Paired T-Test...
Comparison field: Percent_correct
Significance: 0.05
Sorting (asc.) by: <default>
Show std. deviations: ☐

```
15:53:24 - Available resultsets
15:53:30 - Percent_correct - trees.J48 '-C 0.25 -
```

```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Percent_correct
Datasets:   1
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       3/3/10 3:53 PM


Dataset                   (1) trees.J4 | (2) rules (3) rules
------------------------------------------------------------
iris                     (100)   94.73 |   93.53     33.33 *
------------------------------------------------------------
                               (v/ /*) |   (0/1/0)   (0/0/1)


Key:
(1) trees.J48 '-C 0.25 -M 2' -2.17733168393644448E17
(2) rules.OneR '-B 6' -2.4594270021478615E18
(3) rules.ZeroR '' 4.8055541465867952E16
```

FIGURE 13.2

Statistical test results for the experiment of Figure 13.1.

displayed horizontally, numbered *(1)*, *(2)*, and *(3)*, as the heading of a little table. The labels for the columns are repeated at the bottom—*trees.J48*, *rules.OneR*, and *rules.ZeroR*—in case there is insufficient space for them in the heading. The inscrutable integers beside the scheme names identify which version of the scheme is being used. They are present by default to avoid confusion among results generated using different versions of the algorithms. The value in brackets at the beginning of the *iris* row *(100)* is the number of experimental runs: 10 times tenfold cross-validation.

The percentage correct for the three schemes is shown in Figure 13.2: 94.73% for method 1, 93.53% for method 2, and 33.33% for method 3. The symbol placed beside a result indicates that it is statistically better (*v*) or worse (*) than the baseline scheme—in this case *J48*—at the specified significance level (0.05, or 5%). The corrected resampled *t*-test from Section 5.5 (page 159) is used here. As shown, method 3 is significantly worse than method 1 because its success rate is followed by an asterisk. At the bottom of columns 2 and 3 are counts (*x/y/z*) of the number of times the scheme was better than (*x*), the same as (*y*), or worse than (*z*) the baseline scheme on the datasets used in the experiment. In this case there is only one dataset; method 2 was equivalent to method 1 (the baseline) once, and method 3 was worse than it once. (The annotation (*v/ /**) is placed at the bottom of column 1 to help you remember the meanings of the three counts (*x/y/z*).

13.2 SIMPLE SETUP

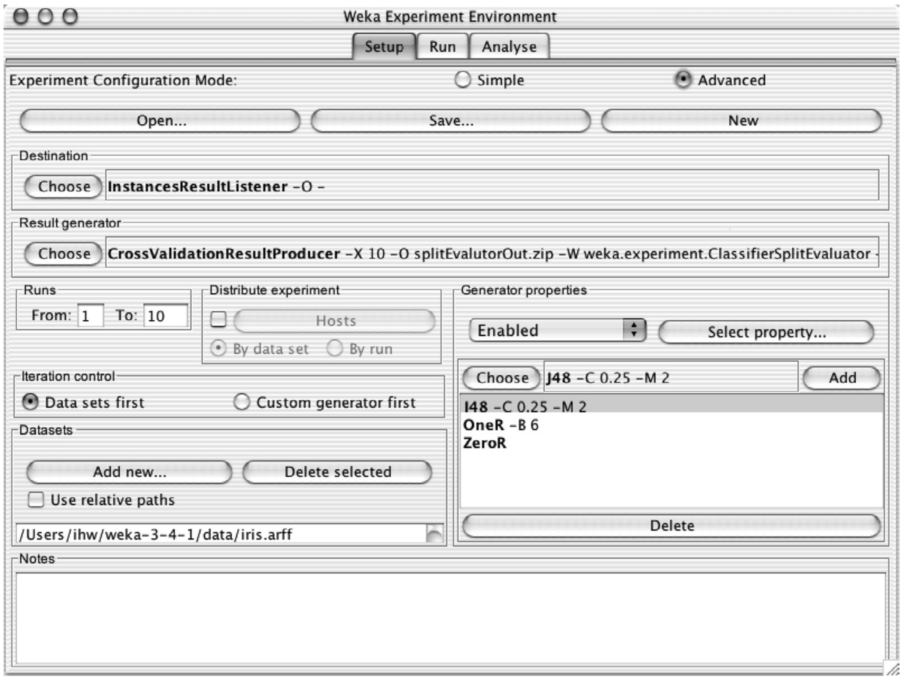
In the *Setup* panel of Figure 13.1(a) we left most options at their default values. The experiment type is a tenfold cross-validation repeated 10 times. You can alter the number of folds in the box at center left and the number of repetitions in the box at center right. The experiment type is classification; you can specify regression instead. You can choose several datasets, in which case each algorithm is applied to each dataset, and change the order of iteration using the *Data sets first* and *Algorithm first* buttons. The alternative to cross-validation is the holdout method. There are two variants depending on whether the order of the dataset is preserved or the data is randomized. You can specify the percentage split (the default is two-thirds training set and one-third test set).

Experimental setups can be saved and reopened. You can make notes about the setup by pressing the *Notes* button, which brings up an editor window. Serious Weka users soon find the need to open up an experiment and rerun it with some modifications, perhaps with a new dataset or a new learning algorithm. It would be nice to avoid having to recalculate all the results that have already been obtained! If the results have been placed in a database rather than an ARFF or CSV file, this is exactly what happens. You can choose *JDBC database* in the results destination selector and connect to any database that has a JDBC driver. You need to specify the database's URL and enter a username and password. To make this work with your database you may need to modify the *weka/experiment/DatabaseUtils.props* file in the Weka distribution. If you alter an experiment that uses a database, Weka will reuse previously computed results whenever they are available. This greatly simplifies the kind of iterative experimentation that typically characterizes data mining research.

13.3 ADVANCED SETUP

The Experimenter has an advanced mode. Click near the top of the panel shown in Figure 13.1(a) to obtain the more formidable version of the panel shown in Figure 13.3. This enlarges the options available for controlling the experiment, including, for example, the ability to generate learning curves. However, the advanced mode is hard to use, and the simple version suffices for most purposes. For example, in the advanced mode you can set up an iteration to test an algorithm with a succession of different parameter values, but the same effect can be achieved in simple mode by putting the algorithm into the list several times with different parameter values.

One thing you can do in advanced mode but not in simple mode is run experiments using clustering algorithms. Here, experiments are limited to those clusterers



Experiment Configuration Mode: Simple [✓ Advanced]

Destination: InstancesResultListener –O –

Result generator: CrossValidationResultProducer –X 10 –O splitEvalutorOut.zip –W weka.experiment.ClassifierSplitEvaluator -

Runs From: 1 To: 10

Distribute experiment: [✓ By data set] By run

Iteration control: [✓ Data sets first] Custom generator first

Generator properties: Enabled

J48 –C 0.25 –M 2

J48 –C 0.25 –M 2
OneR –B 6
ZeroR

Datasets: Use relative paths

/Users/ihw/weka-3-4-1/data/iris.arff

FIGURE 13.3

Setting up an experiment in advanced mode.

that can compute probability or density estimates, and the main evaluation measure for comparison purposes is the log-likelihood. To set this up quickly, first click the *Result generator* to bring up an object editor for the *CrossValidationResultProducer*. Then click the *Choose* button for the split evaluator and select *DensityBasedClustererSplitEvaluator* from the list. At this point the panel on the lower right that contained the list of classifiers goes blank and the *Generator properties* dropdown box displays *Disabled*. Reenable this, and a new window appears with a list of properties (Figure 13.4(a)). Expand the *splitEvaluator* entry, select *clusterer* (as shown in the figure), and click the *Select* button. Now the active list will reappear in the bottom right panel, along with the ability to add clustering schemes, just as we did with classifiers.

Figure 13.4(b) shows a setup with two clustering schemes configured: *EM* and *MakeDensityBasedClusterer* wrapped around *SimpleKMeans*. After running this experiment, these two can be compared in the *Analyze* panel. The comparison field is not set up with a meaningful default, so choose *Log_likelihood* from the dropdown box before pressing the *Perform test* button. Figure 13.4(c) shows the results for these clustering algorithms.

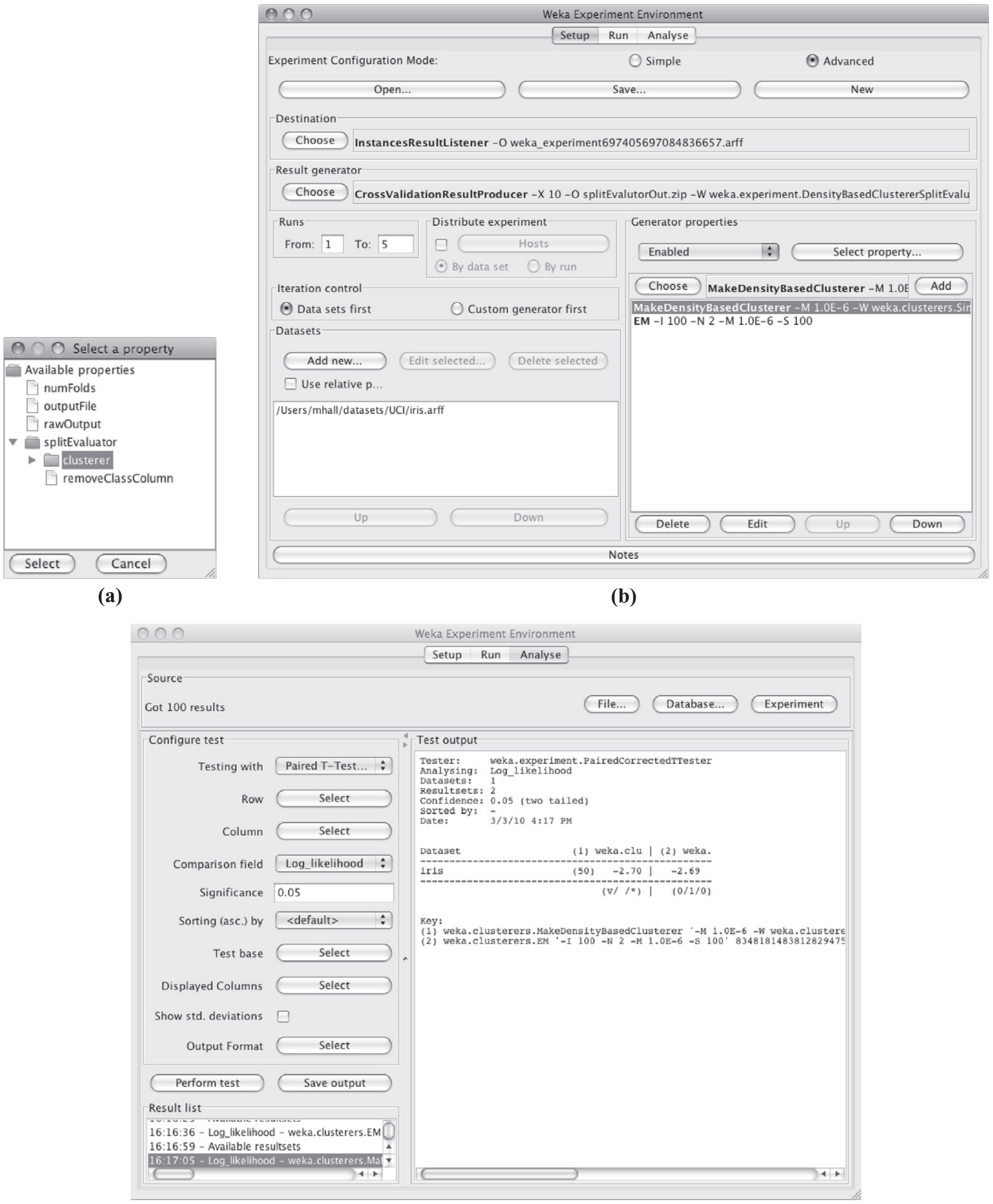
Another thing you may need the advanced mode for is to set up distributed experiments, which we describe in Section 13.5 (page 515).

13.4 THE ANALYZE PANEL

Our walkthrough used the *Analyze* panel to perform a statistical significance test of one learning scheme (*J48*) versus two others (*OneR* and *ZeroR*). The test was on the error rate—the *Comparison* field in Figure 13.2. Other statistics can be selected from the dropdown menu instead: percentage incorrect, percentage unclassified, root mean-squared error, the remaining error measures from Table 5.8, and various entropy figures. Moreover, you can see the standard deviation of the attribute being evaluated by ticking the *Show std deviations* checkbox.

Use the *Test base* menu to change the baseline scheme from *J48* to one of the other learning schemes. For example, selecting *OneR* causes the others to be compared with this scheme. In fact, that would show that there is a statistically significant difference between *OneR* and *ZeroR* but not between *OneR* and *J48*. Apart from the learning schemes, there are two other choices in the *Select base* menu: *Summary* and *Ranking*. The former compares each learning scheme with every other scheme and prints a matrix with cells that contain the number of datasets on which one is significantly better than the other. The latter ranks the schemes according to the total number of datasets that represent wins (>) and losses (<) and prints a league table. The first column in the output gives the difference between the number of wins and the number of losses.

The *Row* and *Column* fields determine the dimensions of the comparison matrix. Clicking *Select* brings up a list of all the features that have been measured in the experiment—in other words, the column labels of the spreadsheet in Figure 13.1(c).



Experiment Configuration Mode: Simple [✓ Advanced]

Destination: InstancesResultListener -O weka_experiment697405697084836657.arff

Result generator: CrossValidationResultProducer -X 10 -O splitEvalutorOut.zip -W weka.experiment.DensityBasedClustererSplitEvalu

Runs: From: 1 To: 5

Distribute experiment

Generator properties: Enabled

Iteration control: [✓ Data sets first] Custom generator first

Datasets: Use relative p...

/Users/mhall/datasets/UCI/iris.arff

MakeDensityBasedClusterer -M 1.0E

MakeDensityBasedClusterer -M 1.0E-6 -W weka.clusterers.Sin
EM -I 100 -N 2 -M 1.0E-6 -S 100

Source: Got 100 results

Configure test

Testing with	Paired T-Test...
Row	Select
Column	Select
Comparison field	Log_likelihood
Significance	0.05
Sorting (asc.) by	<default>
Test base	Select
Displayed Columns	Select
Show std. deviations	☐
Output Format	Select

Result list

```
16:16:36 - Log_likelihood - weka.clusterers.EM
16:16:59 - Available resultsets
16:17:05 - Log_likelihood - weka.clusterers.Ma
```

Test output

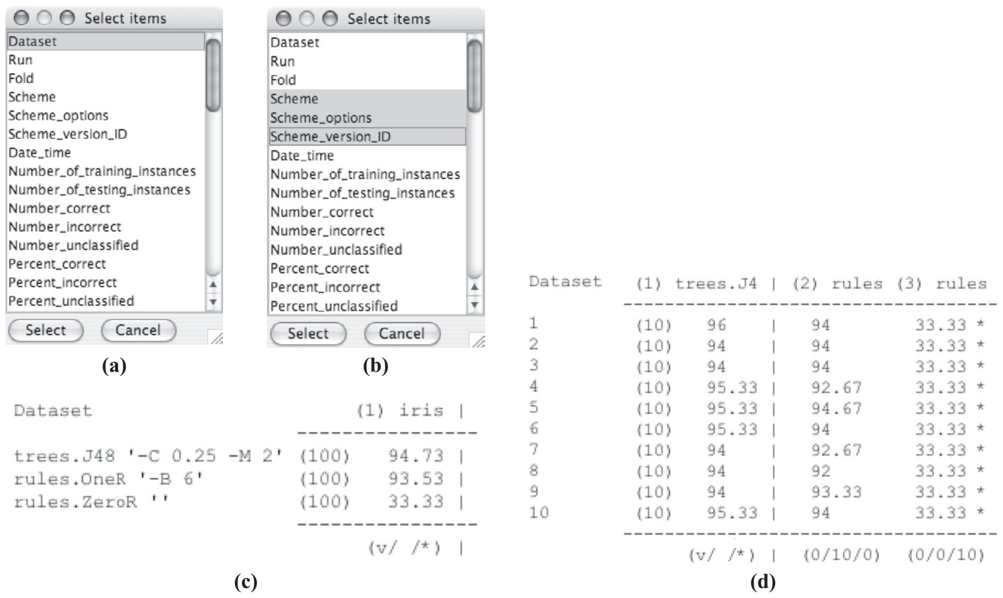
```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Log_likelihood
Datasets:   1
Resultsets: 2
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       3/3/10 4:17 PM


Dataset                   (1) weka.clu | (2) weka.
---------------------------------------------------
iris                      (50)   -2.70 |   -2.69
---------------------------------------------------
                               (v/ /*) |   (0/1/0)


Key:
(1) weka.clusterers.MakeDensityBasedClusterer '-M 1.0E-6 -W weka.clustere
(2) weka.clusterers.EM '-I 100 -N 2 -M 1.0E-6 -S 100' 834818148381282947
```

FIGURE 13.4

An experiment in clustering: (a) generator properties, (b) two clustering schemes, and (c) result panel.



```
Dataset                        (1) iris |
                           -----------------
trees.J48 '-C 0.25 -M 2'  (100)   94.73 |
rules.OneR '-B 6'         (100)   93.53 |
rules.ZeroR ''            (100)   33.33 |
                           -----------------
                                (v/ /*) |
```

```
Dataset    (1) trees.J4 | (2) rules (3) rules
         ---------------------------------------
1          (10)   96    |  94          33.33 *
2          (10)   94    |  94          33.33 *
3          (10)   94    |  94          33.33 *
4          (10)   95.33 |  92.67       33.33 *
5          (10)   95.33 |  94.67       33.33 *
6          (10)   95.33 |  94          33.33 *
7          (10)   94    |  92.67       33.33 *
8          (10)   94    |  92          33.33 *
9          (10)   94    |  93.33       33.33 *
10         (10)   95.33 |  94          33.33 *
         ---------------------------------------
              (v/ /*) |  (0/10/0)    (0/0/10)
```

FIGURE 13.5

Rows and columns of Figure 13.2: (a) row field, (b) column field, (c) result of swapping the row and column selections, and (d) substituting *Run* for *Dataset* as the rows.

You can select which to use as the rows and columns of the matrix. (The selection does not appear in the *Select* box because more than one parameter can be chosen simultaneously.) Figure 13.5 shows which items are selected for the rows and columns of Figure 13.2. The two lists show the experimental parameters (the columns of the spreadsheet). *Dataset* is selected for the rows (and there is only one in this case, the iris dataset), and *Scheme, Scheme options,* and *Scheme_version_ID* are selected for the column (the usual convention of shift-clicking selects multiple entries). All three can be seen in Figure 13.2—in fact, they are more easily legible in the key at the bottom.

If the row and column selections were swapped and the *Perform test* button pressed again, the matrix would be transposed, giving the result in Figure 13.5(c). There are now three rows, one for each algorithm, and one column, for the single dataset. If instead the row of *Dataset* were replaced by *Run* and the test were performed again, the result would be as in Figure 13.5(d). *Run* refers to the runs of the cross-validation, of which there are 10, so there are now 10 rows. The number in parentheses after each row label (100 in Figure 13.5(c) and 10 in Figure 13.5(d)) is the number of results corresponding to that row—in other words, the number of measurements that participate in the averages displayed by the cells in that row.

There is a button that allows you to select a subset of columns to display (the baseline column is always included) and another that allows you to select the output format: plaintext (default), output for the LaTeX typesetting system, CSV format, HTML, data and script suitable for input to the GNUPlot graph plotting software, and just the significance symbols in plaintext format. It is also possible to show averages and abbreviate filter class names in the output.

There is an option for choosing whether to use the paired corrected *t*-test or the standard *t*-test for computing significance. The way that the rows are sorted in the results table can be changed by choosing the *Sorting (asc.) by* option from the dropdown box. The default is to use natural ordering, presenting the rows in the order in which the user entered the dataset names in the *Setup* panel. Alternatively, the rows can be sorted according to any of the measures that are available in the *Comparison field*.

13.5 DISTRIBUTING PROCESSING OVER SEVERAL MACHINES

A remarkable feature of the Experimenter is that it can split up an experiment and distribute it across several processors. This is for advanced Weka users and is only available from the advanced version of the *Setup* panel. Some users avoid working with this panel by setting up the experiment on the simple version and switching to the advanced version to distribute it, because the experiment's structure is preserved when you switch. However, distributing an experiment *is* an advanced feature and is often difficult. For example, file and directory permissions can be tricky to set up.

Distributing an experiment works best when the results are all sent to a central database by selecting *JDBC database* as the results destination in Figure 13.1(a). It uses the RMI facility and works with any database that has a JDBC driver. It has been tested on several freely available databases. Alternatively, you could instruct each host to save its results to a different ARFF file and merge the files afterwards.

To distribute an experiment, each host must (1) have Java installed, (2) have access to whatever datasets you are using, and (3) be running the *weka.experiment.RemoteEngine* experiment server. If results are sent to a central database, the appropriate JDBC drivers must be installed on each host. Getting all this right is the difficult part of running distributed experiments.

To initiate a remote engine experiment server on a host machine, first copy *remoteExperimentServer.jar* from the Weka distribution to a directory on the host. Unpack it with

```
jar –xvf remoteExperimentServer.jar
```

It expands to three files: *remoteEngine.jar*, an executable *jar* file that contains the experiment server; *remote.policy*; and *remote.policy.example*.

The *remote.policy* file grants the remote engine permission to perform certain operations, such as connecting to ports or accessing a directory. It needs to be edited to specify correct paths in some of the permissions; this is self-explanatory when you examine the file. By default, it specifies that code can be downloaded on HTTP port 80 from anywhere on the Web, but the remote engines can also load code from a file URL instead. To arrange this, either uncomment the example in *remote.policy* or tailor *remote.policy.example* to suit your needs. The latter file contains a complete example for a fictitious user (*johndoe*) under a Linux operating system. The remote engines also need to be able to access the datasets used in an experiment (see the first entry in *remote.policy*). The paths to the datasets are specified in the Experimenter (i.e., the client), and the same paths must be applicable in the context of the remote engines. To facilitate this, it may be necessary to specify relative path names by selecting the *Use relative paths* tick box shown in the *Setup* panel of the Experimenter.

To start the remote engine server, type

```
java -classpath remoteEngine.jar:<path_to_any_jdbc_drivers>
-Djava.security.policy=remote.policy weka.experiment.RemoteEngine
```

from the directory containing *remoteEngine.jar*. If everything is going well, you will see this message (or something like it):

```
user@ml:remote_engine>Host name : ml.cs.waikato.ac.nz
Attempting to start RMI registry on port 1099 …
RemoteEngine bound in RMI registry
```

This indicates that the remote engine has started the RMI registry on port 1099 and is running successfully. You can run more than one remote engine on a given machine, and it makes sense to do so if the machine in question has multiple processors or a multicore processor. To do so, start each remote engine as before, but instead of the default port (1099), specify a different one using a command-line option (*–p*) to the remote engine. Repeat the process for all hosts.

Now start the Experimenter by typing

```
java -Djava.rmi.server.codebase=<URL_for_weka_code> weka.gui.
experiment.Experimenter
```

The URL specifies where the remote engines can find the code to be executed. If it denotes a directory (i.e., one that contains the Weka directory) rather than a *jar* file, it must end with a path separator (e.g., /).

The Experimenter's advanced *Setup* panel shown earlier in Figure 13.3 contains a small pane at center left; it determines whether an experiment will be distributed or not. This is normally inactive. To distribute the experiment click the checkbox, which will activate the *Hosts* button; a window will pop up asking for the machines over which to distribute the experiment. Host names should be fully qualified (e.g., *ml.cs.waikato.ac.nz*).

If a host is running more than one remote engine, enter its name into the window multiple times, along with the port number if it is not the default. For example:

```
ml.cs.waikato.ac.nz
ml.cs.waikato.ac.nz:5050
```

This tells the Experimenter that the host *ml.cs.waikato.ac.nz* is running two remote engines, one at the default port of 1099 and a second at port 5050.

Having entered the hosts, configure the rest of the experiment in the usual way (better still, configure it before switching to the advanced setup mode). When the experiment is started using the *Run* panel, the progress of the subexperiments on the various hosts is displayed, along with any error messages.

Distributing an experiment involves splitting it into subexperiments that RMI sends to the hosts for execution. By default, experiments are partitioned by dataset, in which case there can be no more hosts than there are datasets. Then each subexperiment is self-contained: It applies all schemes to a single dataset. An experiment with only a few datasets can be partitioned by run instead. For example, a 10 times tenfold cross-validation would be split into 10 subexperiments, 1 per run.