CHAPTER

The Knowledge Flow Interface 12

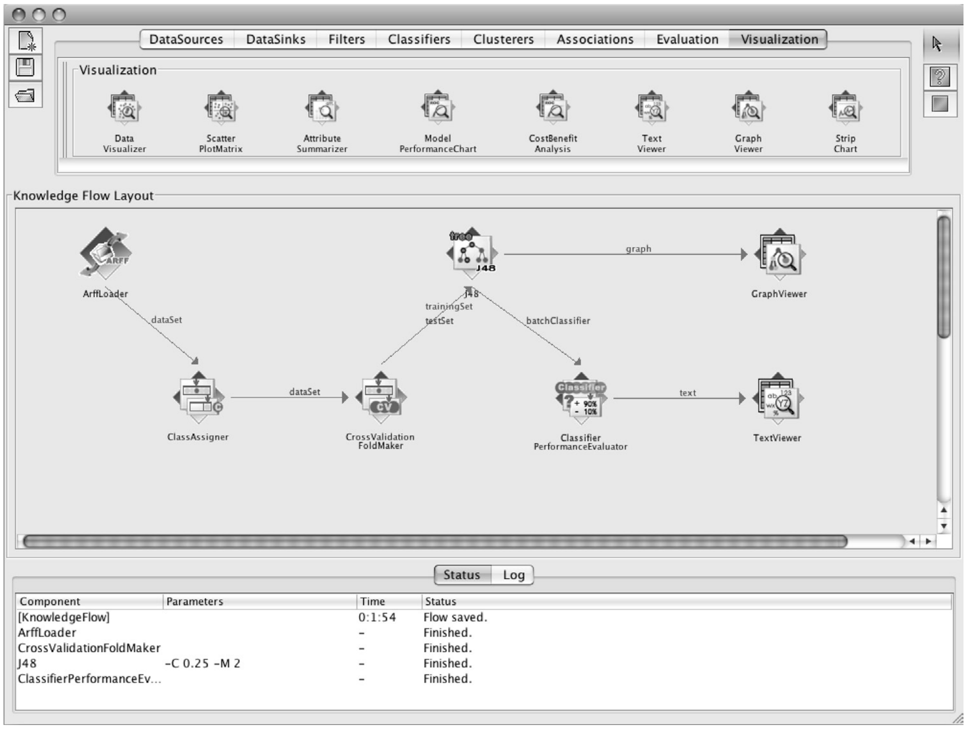
With the Knowledge Flow interface, users select Weka components from a tool bar, place them on a layout canvas, and connect them into a directed graph that processes and analyzes data. It provides an alternative to the Explorer for those who like thinking in terms of how data flows through the system. It also allows the design and execution of configurations for streamed data processing, which the Explorer cannot do. You invoke the Knowledge Flow interface by selecting *KnowledgeFlow* from the choices in the right panel shown in Figure 11.3(a).

12.1 GETTING STARTED

Here is a step-by-step example that loads an ARFF file and performs a cross-validation using *J4.8*. We describe how to build up the final configuration shown in Figure 12.1. First, create a source of data by clicking on the *DataSources* tab (leftmost entry in the bar at the top) and selecting *ARFFLoader* from the toolbar. The mouse cursor changes to crosshairs to signal that you should next place the component. Do this by clicking anywhere on the canvas, whereupon a copy of the ARFF loader icon appears there. To connect it to an ARFF file, right-click it to bring up the pop-up menu shown in Figure 12.2(a). Click *Configure* to get the file browser in Figure 12.2(b), from which you select the desired ARFF file (alternatively, double-clicking on a component's icon is a short-cut for selecting *Configure* from the pop-up menu).

Now we specify which attribute is the class using a *ClassAssigner* object. This is on the *Evaluation* panel, so click the *Evaluation* tab, select the *ClassAssigner*, and place it on the canvas. To connect the data source to the class assigner, right-click the data source icon and select *dataSet* from the menu, as shown in Figure 12.2(a). A rubber-band line appears. Move the mouse over the class assigner component and left-click. A red line labeled *dataSet* appears, joining the two components. Having connected the class assigner, choose the class by right-clicking it, selecting *Configure*, and entering the location of the class attribute.

We will perform cross-validation on the *J48* classifier. In the data flow model, we first connect the *CrossValidationFoldMaker* to create the folds on which the classifier will run, and then pass its output to an object representing *J48*.



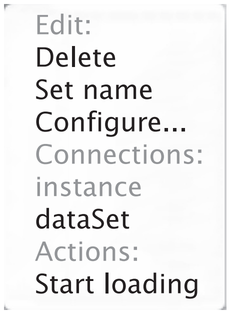
Component	Parameters	Time	Status
[KnowledgeFlow]		0:1:54	Flow saved.
ArffLoader		-	Finished.
CrossValidationFoldMaker		-	Finished.
J48	-C 0.25 -M 2	-	Finished.
ClassifierPerformanceEv...		-	Finished.

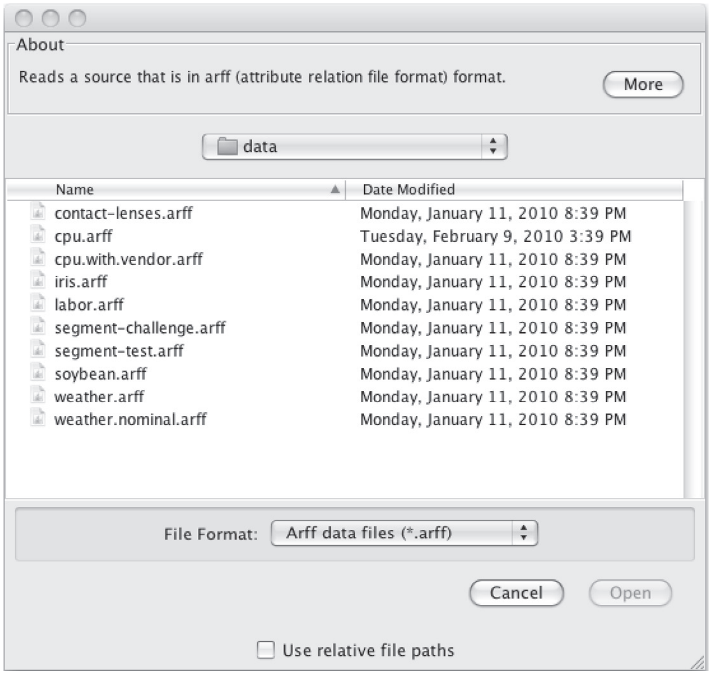
FIGURE 12.1

The Knowledge Flow interface.

CrossValidationFoldMaker is on the *Evaluation* panel. Select it, place it on the canvas, and connect it to the class assigner by right-clicking the latter and selecting *dataSet* from the menu (which is similar to that in Figure 12.2(a)). Next, select *J48* from the *Classifiers* panel and place a *J48* component on the canvas; there are so many different classifiers that you have to scroll along the toolbar to find it. Connect *J48* to the cross-validation fold maker in the usual way, but make the connection *twice* by first choosing *trainingSet* and then *testSet* from the pop-up menu for the cross-validation fold maker. The next step is to select a *ClassifierPerformanceEvaluator* from the *Evaluation* panel and connect *J48* to it by selecting the *batchClassifier* entry from the pop-up menu for *J48*. Finally, from the *Visualization* toolbar place a *TextViewer* component on the canvas. Connect the classifier performance evaluator to it by selecting the *text* entry from the pop-up menu for the performance evaluator.

At this stage the configuration is as shown in Figure 12.1 except that there is as yet no graph viewer. Start the flow of execution by selecting *Start loading* from the pop-up menu for the ARFF loader, shown in Figure 12.2(a). For a small dataset


Edit:
Delete
Set name
Configure...
Connections:
instance
dataSet
Actions:
Start loading


About

Reads a source that is in arff (attribute relation file format) format.

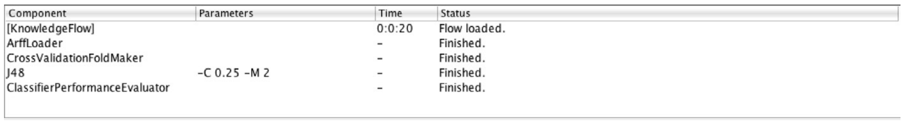
Name	Date Modified
contact-lenses.arff	Monday, January 11, 2010 8:39 PM
cpu.arff	Tuesday, February 9, 2010 3:39 PM
cpu.with.vendor.arff	Monday, January 11, 2010 8:39 PM
iris.arff	Monday, January 11, 2010 8:39 PM
labor.arff	Monday, January 11, 2010 8:39 PM
segment-challenge.arff	Monday, January 11, 2010 8:39 PM
segment-test.arff	Monday, January 11, 2010 8:39 PM
soybean.arff	Monday, January 11, 2010 8:39 PM
weather.arff	Monday, January 11, 2010 8:39 PM
weather.nominal.arff	Monday, January 11, 2010 8:39 PM

File Format: Arff data files (*.arff)

(a) (b)

FIGURE 12.2

Configuring a data source: (a) right-click menu and (b) file browser obtained from the *Configure* menu item.



Component	Parameters	Time	Status
[KnowledgeFlow]		0:0:20	Flow loaded.
ArffLoader		-	Finished.
CrossValidationFoldMaker		-	Finished.
J48	-C 0.25 -M 2	-	Finished.
ClassifierPerformanceEvaluator		-	Finished.

FIGURE 12.3

Status area after executing the configuration shown in Figure 12.1.

things happen quickly. Progress information appears in the status area at the bottom of the interface. The entries in the status area show the progress of each step in the flow, along with their parameter settings (for learning schemes) and elapsed time. Any errors that occur in a processing step are shown in the status area by highlighting the corresponding row in red. Figure 12.3 shows the status area after executing the configuration shown in Figure 12.1. Choosing *Show results* from the text viewer's pop-up menu brings up the results of cross-validation in a separate window, in the same form as for the Explorer.

To complete the example, add a *GraphViewer* and connect it to *J48*'s *graph* output to see a graphical representation of the trees produced for each fold of the cross-validation. Once you have redone the cross-validation with this extra component in place, selecting *Show results* from its pop-up menu produces a list of trees, one for each cross-validation fold. By creating cross-validation folds and passing them to the classifier, the Knowledge Flow model provides a way to hook into the results for each fold. The Explorer cannot do this: It treats cross-validation as an evaluation method that is applied to the output of a classifier.

12.2 COMPONENTS

Most of the Knowledge Flow components will be familiar from the Explorer. The *Classifiers* panel contains all of Weka's classifiers, the *Filters* panel contains the filters, the *Clusterers* panel holds the clusterers, and the *Associations* panel holds the association rule learners. Classifiers in the Knowledge Flow have the option of processing cross-validation training sets in parallel. In fact, since dual-core processors are the norm these days, the default is to process two cross-validation training folds in parallel. The option that controls this is called *Execution slots* and is available from the object editor that appears when *Configure* is selected after right-clicking on a *Classifiers* component.

Possible data sources are ARFF files, XML ARFF files, CSV files exported from spreadsheets, the C4.5 file format, databases, serialized instances, LibSVM and SVMLight data formats, and a special loader (*TextDirectoryLoader*) to load a directory of plaintext files into a single set of instances. There is a data sink that corresponds to each data source, with the exception of the *TextDirectoryLoader*.

The components for visualization and evaluation, which are listed in Table 12.1, have not all been encountered yet. Under *Visualization*, the *DataVisualizer* pops up a panel for visualizing data in a two-dimensional scatter plot, as in Figure 11.6(b), in which you can select the attributes you would like to see. *ScatterPlot-Matrix* pops up a matrix of two-dimensional scatter plots for every pair of attributes, shown in Figure 11.17(a). *AttributeSummarizer* gives a matrix of histograms, one for each attribute, like that in the lower right corner of Figure 11.3(b). *Model PerformanceChart* draws ROC curves and other threshold curves. *CostBenefit-Analysis* allows interactive exploration of the tradeoffs in cost or benefit arising from different cost matrices (see Section 5.7, page 166). *GraphViewer*, used earlier, pops up a panel for visualizing tree-based models, as in Figure 11.6(a). As before, you can zoom, pan, and visualize the instance data at a node (if it has been saved by the learning algorithm).

StripChart is a new visualization component designed for use with incremental learning. In conjunction with the *IncrementalClassifierEvaluator* described next, it displays a learning curve that plots accuracy—both the percentage accuracy and the root mean-squared probability error—against time. It shows a fixed-size time window that scrolls horizontally to reveal the latest results.

Table 12.1 Visualization and Evaluation Components

	Name	Function
Visualization	*DataVisualizer*	Visualize data in a two-dimensional scatter plot
	ScatterPlotMatrix	Matrix of scatter plots
	AttributeSummarizer	Set of histograms, one for each attribute
	ModelPerformanceChart	Draw ROC and other threshold curves
	CostBenefitAnalysis	Visualize cost or benefit tradeoffs
	TextViewer	Visualize data or models as text
	GraphViewer	Visualize tree-based models
	StripChart	Display a scrolling plot of data
Evaluation	*TrainingSetMaker*	Make dataset into a training set
	TestSetMaker	Make dataset into a test set
	CrossValidationFoldMaker	Split dataset into folds
	TrainTestSplitMaker	Split dataset into training and test sets
	InstanceStreamToBatchMaker	Collect instances from a stream and assemble them into a batch dataset
	ClassAssigner	Assign one of the attributes to be the class
	ClassValuePicker	Choose a value for the *positive* class
	ClassifierPerformanceEvaluator	Collect evaluation statistics for batch evaluation
	IncrementalClassifierEvaluator	Collect evaluation statistics for incremental evaluation
	ClustererPerformanceEvaluator	Collect evaluation statistics for clusterers
	PredictionAppender	Append a classifier's predictions to a dataset
	SerializedModelSaver	Save trained models as serialized Java objects

The *Evaluation* panel has the components listed in the lower part of Table 12.1. *TrainingSetMaker* and *TestSetMaker* make a dataset into the corresponding kind of set. *CrossValidationFoldMaker* constructs cross-validation folds from a dataset; *TrainTestSplitMaker* splits it into training and test sets by holding part of the data out for the test set. *InstanceStreamToBatchMaker* collects instances arriving in a stream from an incoming "instance" connection and produces a batch dataset when the last instance has arrived. *ClassAssigner* allows you to decide which attribute is the class. With *ClassValuePicker* you choose a value that is treated as the *positive* class when generating ROC and other threshold curves. *ClassifierPerformanceEvaluator* collects evaluation statistics: It can send the textual evaluation to a text viewer and the threshold curves to a performance chart. *IncrementalClassifierEvaluator* performs the same function for incremental classifiers: It computes running squared errors and so on. There is also *ClustererPerformanceEvaluator*, which is similar to *ClassifierPerformanceEvaluator*. *PredictionAppender* takes a classifier and a dataset and appends the classifier's predictions to the dataset. *SerializedModelSaver* takes a classifier or clusterer and saves the model out to a file as a serialized Java object.

12.3 CONFIGURING AND CONNECTING THE COMPONENTS

You establish the knowledge flow by configuring individual components and connecting them. Figure 12.4 shows typical operations that are available by right-clicking the various component types. These menus shown have up to three sections: *Edit*, *Connections*, and *Actions*. The *Edit* operations delete components and open up their configuration panel. You can give a component a name by choosing *Set name* from the pop-up menu. Classifiers and filters are configured just as in the Explorer. Data sources are configured by opening a file (as we

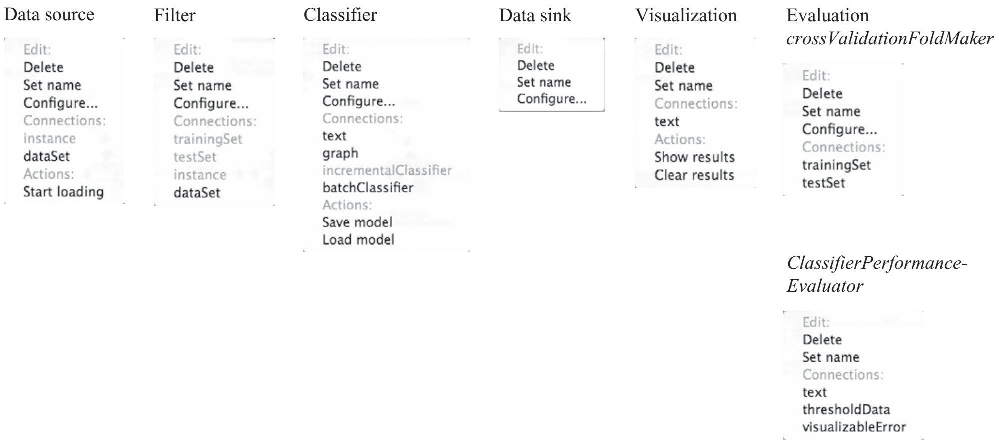


FIGURE 12.4

Operations on the Knowledge Flow components.

saw previously) or by setting a database connection, and evaluation components are configured by setting parameters such as the number of folds for cross-validation. The *Actions* operations are specific to the type of component, such as starting to load data from a data source or opening a window to show the results of visualization. The *Connections* operations are used to connect components together by selecting the type of connection from the source component and then clicking on the target object. Not all of the targets are suitable; applicable ones are highlighted. Items on the *Connections* menu are disabled (grayed out) until the component receives other connections that render them applicable.

There are two kinds of connections from data sources: *dataSet* connections and *instance* connections. The former are for batch operations such as classifiers like *J48*; the latter are for stream operations such as *NaiveBayesUpdateable*. A data source component cannot provide both types of connection: Once one is selected, the other is disabled. When a *dataSet* connection is made to a batch classifier, the classifier needs to know whether it is intended to serve as a training set or a test set. To do this, you first make the data source into a test or training set using the *TestSetMaker* or *TrainingSetMaker* components from the *Evaluation* panel.

On the other hand, an *instance* connection to an incremental classifier is made directly: There is no distinction between training and testing because the instances that flow update the classifier incrementally. In this case a prediction is made for each incoming instance and incorporated into the test results; then the classifier is trained on that instance. If you make an *instance* connection to a batch classifier, it will be used as a test instance because training cannot possibly be incremental whereas testing always can be. Conversely, it is quite possible to test an incremental classifier in batch mode using a *dataSet* connection.

Connections from a filter component are enabled when the component receives input from a data source, whereupon follow-on *dataSet* or *instance* connections can be made. *Instance* connections cannot be made to supervised filters or to unsupervised filters that cannot handle data incrementally (e.g., *Discretize*). To get a test or training set out of a filter, you need to put the appropriate kind in.

The classifier menu has two types of connection. The first type, namely *graph* and *text* connections, provides graphical and textual representations of the classifier's learned state, and it is only activated when it receives a training set input. The other type, *batchClassifier* and *incrementalClassifier*, makes data available to a performance evaluator, and it is activated only when a test set input is present too. Which one is activated depends on the type of the classifier.

Evaluation components are a mixed bag. *TrainingSetMaker* and *TestSetMaker* turn a dataset into a training or test set. *CrossValidationFoldMaker* turns a dataset into *both* a training set and a test set. *ClassifierPerformanceEvaluator* (used in the example in Section 12.1) generates textual and graphical output for visualization components. Other evaluation components operate like filters: They enable follow-on *dataSet*, *instance*, *trainingSet*, or *testSet* connections depending on the input (e.g., *ClassAssigner* assigns a class to a dataset). *InstanceStreamToBatchMaker* takes an incoming stream of instances and assembles them into a batch dataset. This is particularly useful when placed after a reservoir sampling filter—it allows the instances output by reservoir sampling to be used to train batch learning schemes.

Visualization components do not have connections, although some have actions such as *Show results* and *Clear results*.

12.4 INCREMENTAL LEARNING

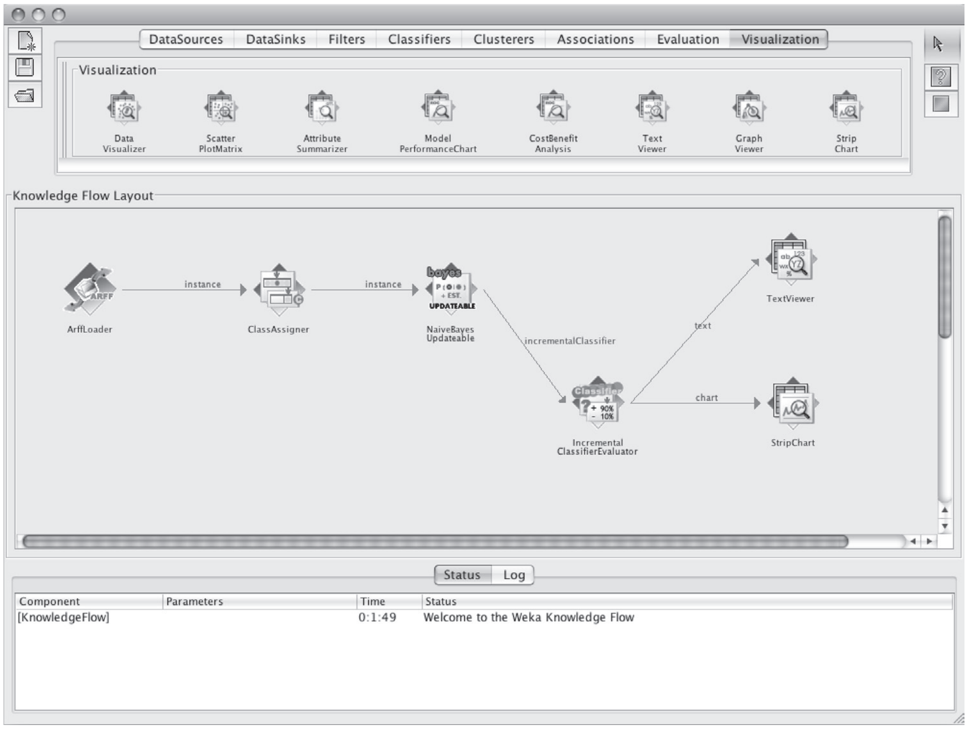
In most respects the Knowledge Flow interface is functionally similar to the Explorer: You can do similar things with both. It does provide some additional flexibility—for example, you can see the tree that *J48* makes for each cross-validation fold. But its real strength is the potential for incremental operation.

Weka has several classifiers that can handle data incrementally: *AODE*, a version of Naïve Bayes (*NaiveBayesUpdateable*), *Winnow*, instance-based learners (*IB1*, *IBk*, *KStar*, *LWL*), *DMNBText*, *NaiveBayesMultinomialUpdateable*, and *NNge*. The meta-learner *RacedIncrementalLogitBoost* operates incrementally (see Section 11.1, page 477). All filters that work instance by instance are incremental: *Add*, *AddExpression*, *AddValues*, *ChangeDateFormat*, *ClassAssigner*, *Copy, FirstOrder*, *MakeIndicator*, *MergeTwoValues*, *NonSparseToSparse*, *NumericToBinary*, *NumericTransform*, *NumericCleaner*, *Obfuscate*, *RandomSubset*, *Remove*, *RemoveType*, *RemoveWithValues*, *Reorder*, *ReservoirSample*, *SparseToNonSparse*, and *SwapValues*.

If all components connected up in the Knowledge Flow interface operate incrementally, so does the resulting learning system. It does not read in the dataset before learning starts, as the Explorer does. Instead, the data source component reads the input instance by instance and passes it through the Knowledge Flow chain.

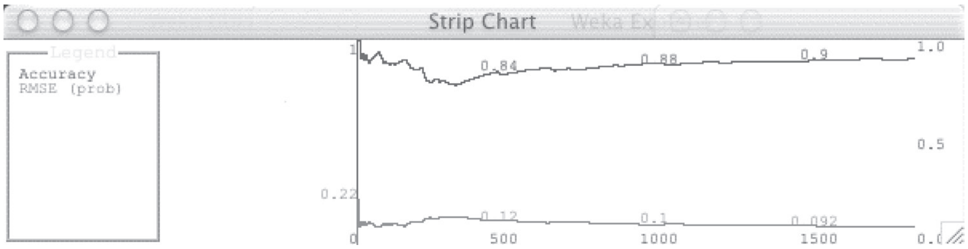
Figure 12.5(a) shows a configuration that works incrementally. An *instance* connection is made from the loader to a class assigner component, which, in turn, is connected to the updatable Naïve Bayes classifier. The classifier's text output is taken to a viewer that gives a textual description of the model. Also, an *incrementalClassifier* connection is made to the corresponding performance evaluator. This produces an output of type *chart*, which is piped to a strip chart visualization component to generate a scrolling data plot.

Figure 12.5(b) shows the strip chart output. It plots both accuracy and the root mean-squared probability error against time. As time passes, the whole plot (including the axes) moves leftward to make room for new data at the right. When the vertical axis representing time 0 can move left no farther, it stops and the time origin starts to increase from 0 to keep pace with the data coming in at the right. Thus, when the chart is full it shows a window of the most recent time units.



Component	Parameters	Time	Status
[KnowledgeFlow]		0:1:49	Welcome to the Weka Knowledge Flow

(a)



(b)

FIGURE 12.5

A Knowledge Flow that operates incrementally: (a) the configuration and (b) strip chart output.

The strip chart can be configured to alter the number of instances shown on the *x*-axis.

This particular Knowledge Flow configuration can process input files of any size, even ones that do not fit into the computer's main memory. However, it all depends on how the classifier operates internally. For example, although they are incremental, many instance-based learners store the entire dataset internally.