

Laboratory Module 2

Working with data in Weka

Purpose:

- Attribute-Relation File Format (ARFF)
- Managing the data flow using WEKA

1 Preparation Before Lab

Attribute-Relation File Format (ARFF)

An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files have two distinct sections. The first section is the **Header** information, which is followed the **Data** information.

The **Header** of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types.

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

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
```

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

The ARFF Header Section

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

The @relation Declaration

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

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

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

The @attribute Declarations

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

The format for the **@attribute** statement is:

```
@attribute <attribute-name> <datatype>
```

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

The <datatype> can be any of the four types currently supported by Weka:

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

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

Numeric attributes

Numeric attributes can be real or integer numbers.

Nominal attributes

Nominal values are defined by providing an <nominal-specification> listing the possible values: {<nominal-name1>, <nominal-name2>, <nominal-name3>, ...}

For example, the class value of the Iris dataset can be defined as follows:

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

Values that contain spaces must be quoted.

String attributes

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

```
@ATTRIBUTE LCC string
```

Date attributes

Date attribute declarations take the form:

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

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

Dates must be specified in the data section as the corresponding string representations of the date/time

ARFF Data Section

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

The @data Declaration

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

```
@data
```

The instance data

Each instance is represented on a single line, with carriage returns denoting the end of the instance.

Attribute values for each instance are delimited by commas. They must appear in the order that they were declared in the header section (i.e. the data corresponding to the nth @attribute declaration is always the nth field of the attribute).

Missing values are represented by a single question mark, as in:

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

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

```
@relation LCCvsLCSH
```

```
@attribute LCC string
```

```
@attribute LCSH string
```

```
@data
```

```
AG5, 'Encyclopedias and dictionaries.;Twentieth century.'
```

```
AS262, 'Science -- Soviet Union -- History.'
```

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

```
@RELATION Timestamps
```

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

```
@DATA
```

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

Sparse ARFF files

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

```
@data
```

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

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

```
@data
{ 1 X, 3 Y, 4 "class A" }
{ 2 W, 4 "class B" }
```

Each instance is surrounded by curly braces, and the format for each entry is: <index> <space> <value> where index is the attribute index (starting from 0).

Note that the omitted values in a sparse instance are 0, they are not "missing" values! If a value is unknown, you must explicitly represent it with a question mark (?).

3. Weka GUI

3.1. The Command Line Interface

- One can use the command line interface of Weka either through a command prompt or through the SimpleCLI mode
- For example to fire up Weka and run J48 on a ARFF file present in the current working directory, the command is:

```
Java weka.classifiers.trees.J48 -t weather.arff
```

- Weka consists of a hierarchical package system. For example here J48 program is part of the trees package which further resides in the classifier package. Finally the weka package contains the classifiers package
- Each time the Java virtual machine executes J48, it creates an instance of this class by allocating memory for building and storing a decision tree classifier
- The -t option was used in the command line to communicate the name of the training file to the learning algorithm

Weka.filters

The weka.filters package is concerned with classes that transforms datasets -- by removing or adding attributes, resampling the dataset, removing examples and so on. This package offers useful support for data preprocessing, which is an important step in machine learning.

All filters offer the options -i for specifying the input dataset, and -o for specifying the output dataset. If any of these parameters is not given, this specifies standard input resp. output for use within pipes. Other parameters are specific to each filter and can be found out via -h, as with any other class. The weka.filters package is organized into supervised and unsupervised filtering, both of which are again subdivided into instance and attribute filtering. We will discuss each of the four subsection separately.

3.1.1. Weka.filters.supervised

Classes below weka.filters.supervised in the class hierarchy are for supervised filtering, i.e. taking advantage of the class information. A class must be assigned via -c, for WEKA default behaviour use -c last.

3.1.1.1. Attribute

Discretize is used to discretize numeric attributes into nominal ones, based on the class information, via Fayyad & Irani's MDL method, or optionally with Kononeko's MDL method. At least some learning schemes or classifiers can only process nominal data, e.g. rules.Prism; in some cases discretization may also reduce learning time.

```
java weka.filters.supervised.attribute.Discretize -i data/iris.arff -o iris-nom.arff -c last
java weka.filters.supervised.attribute.Discretize -i data/cpu.arff -o cpu-classvendor-nom.arff -c first
```

NominalToBinary encodes all nominal attributes into binary (two-valued) attributes, which can be used to transform the dataset into a purely numeric representation, e.g. for visualization via multi-dimensional scaling.

```
java weka.filters.supervised.attribute.NominalToBinary -i data/contact-lenses.arff -o contact-lenses-bin.arff -c last
```

Keep in mind that most classifiers in WEKA utilize transformation filters internally, e.g. Logistic and SMO, so you will usually not have to use these filters explicitly. However, if you plan to run a lot of experiments, pre-applying the filters yourself may improve runtime performance.

3.1.1.2.Instance

Resample creates a stratified subsample of the given dataset. This means that overall class distributions are approximately retained within the sample. A bias towards uniform class distribution can be specified via -B.

```
java weka.filters.supervised.instance.Resample -i data/soybean.arff -o soybean-5%.arff -c last -Z 5
java weka.filters.supervised.instance.Resample -i data/soybean.arff -o soybean-uniform-5%.arff -c last -Z 5 -B 1
```

StratifiedRemoveFolds creates stratified cross-validation folds of the given dataset. This means that per default the class distributions are approximately retained within each fold. The following example splits soybean.arff into stratified training and test datasets, the latter consisting of 25% (=1/4) of the data.

```
java weka.filters.supervised.instance.StratifiedRemoveFolds -i data/soybean.arff -o soybean-train.arff \
-c last -N 4 -F 1 -V
java weka.filters.supervised.instance.StratifiedRemoveFolds -i data/soybean.arff -o soybean-test.arff \
-c last -N 4 -F 1
```

3.1.2.Weka.filters.unsupervised

Classes below weka.filters.unsupervised in the class hierarchy are for unsupervised filtering, e.g. the non-stratified version of Resample. A class should not be assigned here.

3.1.2.1.Attribute

StringToWordVector transforms string attributes into a word vectors, i.e. creating one attribute for each word which either encodes presence or word count (-C) within the string. -W can be used to set an approximate limit on the number of words. When a class is assigned, the limit applies to each class separately. This filter is useful for text mining.

Obfuscate renames the dataset name, all attribute names and nominal attribute values. This is intended for exchanging sensitive datasets without giving away restricted information.

Remove is intended for explicit deletion of attributes from a dataset, e.g. for removing attributes of the iris dataset:

```
java weka.filters.unsupervised.attribute.Remove -R 1-2 -i data/iris.arff -o iris-simplified.arff
java weka.filters.unsupervised.attribute.Remove -V -R 3-last -i data/iris.arff -o iris-simplified.arff
```

3.1.2.2.Instance

Resample creates a non-stratified subsample of the given dataset, i.e. random sampling without regard to the class information. Otherwise it is equivalent to its supervised variant.

```
java weka.filters.unsupervised.instance.Resample -i data/soybean.arff -o soybean-5%.arff -Z 5
```

RemoveFolds creates cross-validation folds of the given dataset. The class distributions are not retained. The following example splits soybean.arff into training and test datasets, the latter consisting of 25% (=1/4) of the data.

```
java weka.filters.unsupervised.instance.RemoveFolds -i data/soybean.arff -o soybean-train.arff -c last -N 4 -F 1 -V
java weka.filters.unsupervised.instance.RemoveFolds -i data/soybean.arff -o soybean-test.arff -c last -N 4 -F 1
```

RemoveWithValues filters instances according to the value of an attribute.

```
java weka.filters.unsupervised.instance.RemoveWithValues -i data/soybean.arff \
-o soybean-without_herbicide_injur
```

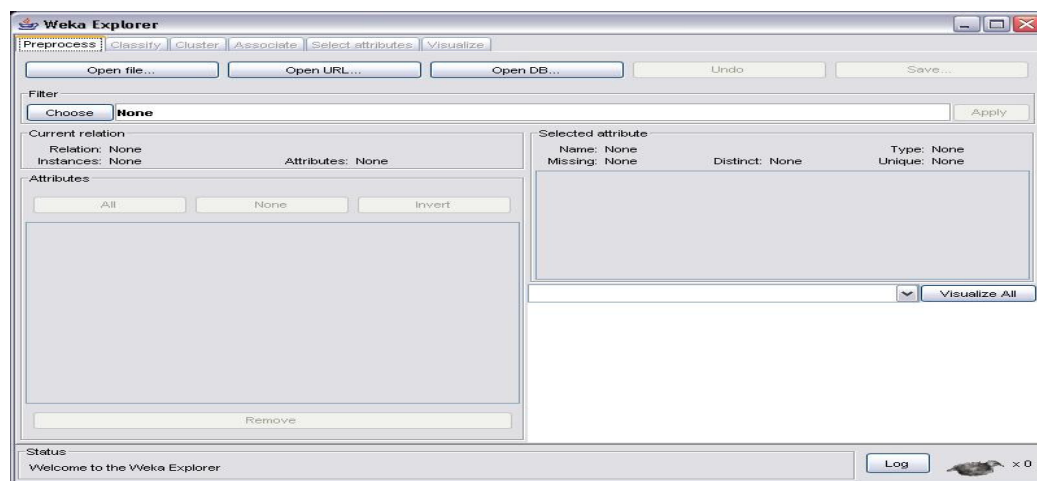
General options

Option	Function
-t <training file>	Specify training file
-T <test file>	Specify test file; if none, a cross-validation is performed on the training data
-c <class index>	Specify index of class attribute
-s <random number stxxi>	Specify random number seed for cross-validation

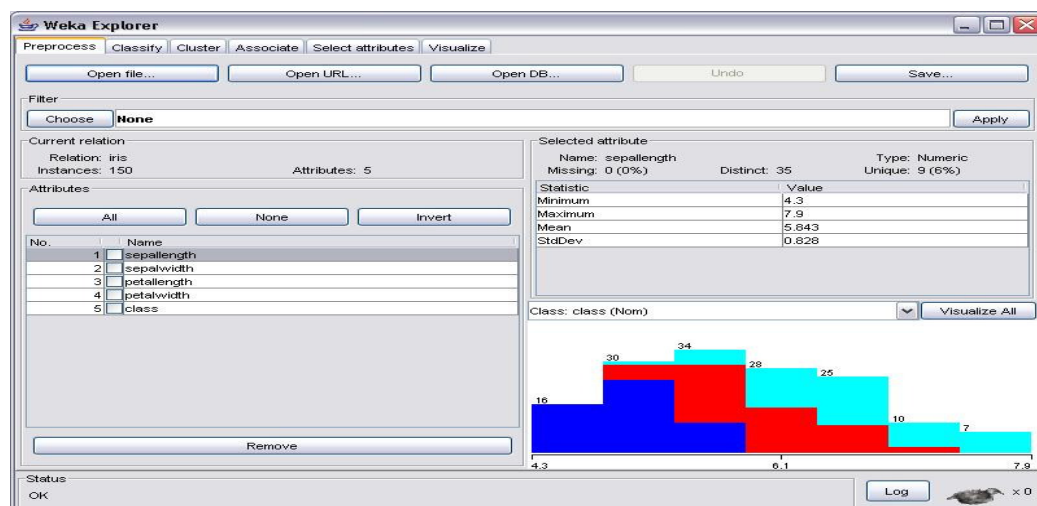
-x <number of folds>	Specify number of folds for cross-validation
-m<cost matrix file>	Specify file containing cost matrix
-d <output file>	Specify output file for model
-l <input file>	Specify input file for model
o	Output statistics only, not the classifier
-i	Output information retrieval statistics for two-class problems
-k	Output information-theoretical statistics
-p <attribute range>	Output predictions for test instances
-v	Output no statistics for training data
r	Output cumulative margin distribution
-z <class name>	Output source representation of classifier
-g	Output graph representation of classifier

3.2. Explorer

Start up Weka. You'll have a choice between the Command Line Interface, the Experimenter, the Explorer and Knowledge flow. Initially, we'll stick with the Explorer. Once you click on that you'll see the main GUI.



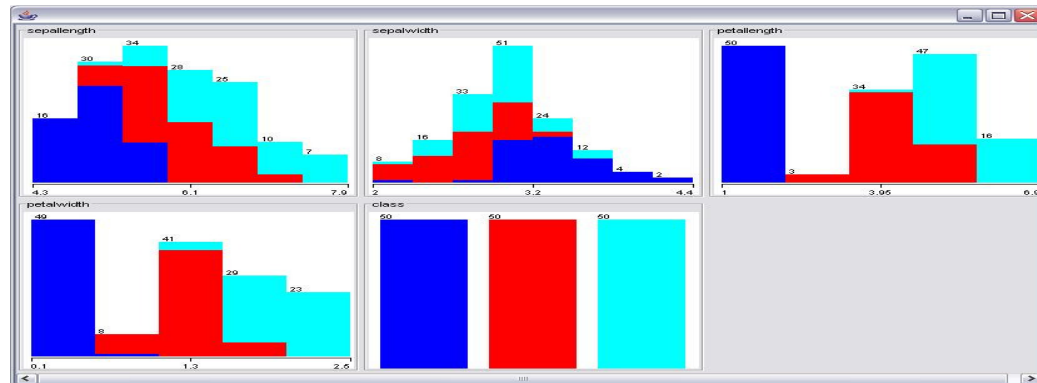
You now have a number of choices, but before you can work with any data, you'll have to load it into Weka. For now, we'll use one of the datasets that are included, but later on you'll have to get any file you'll use into the right format. Open a file from the data subcategory, for example the Iris data to find the following screen.



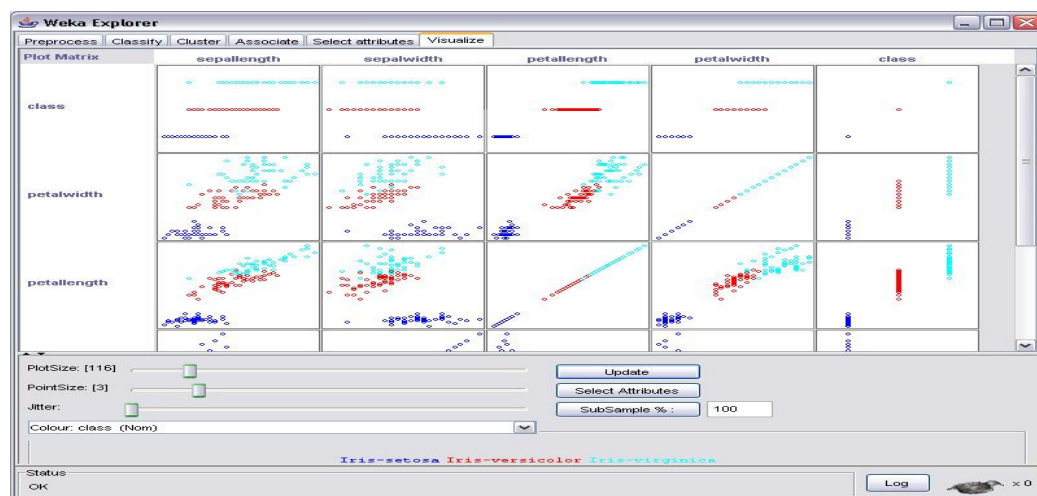
You'll notice that Weka now provides some information about the data, such as for example the number of instances, the number of attributes, and also some statistical information about the attributes one at a time. Figure out how to switch between attributes for which this statistical information is displayed.

Visualization

There are a number of ways in which you can use Weka to visualize your data. The main GUI will show a histogram for the attribute distributions for a single selected attribute at a time, by default this is the class attribute. Note that the individual colors indicate the individual classes (the Iris dataset has 3). If you move the mouse over the histogram, it will show you the ranges and how many samples fall in each range. The button VISUALIZE ALL will let you bring up a screen showing all distributions at once as in the picture below.



There is also a tab called VISUALIZE. Clicking on that will open the scatterplots for all attribute pairs:



From these scatterplots, we can infer a number of interesting things. For example, in the picture above we can see that in some examples the clusters (for now, think of clusters as collections of points that are physically close to each other on the screen) and the different colors correspond to each other such as for example in the plots for class/(any attribute) pairs and the petalwidth/petalwidth attribute pair, whereas for other pairs (sepalwidth/sepalwidth for example) it's much harder to separate the clusters by color.

By default, the colors indicate the different classes, in this case we used red and two shades of blue. Left clicking on any of the highlighted class names towards the bottom of the screenshot allows you to set your own color for the classes. Also, by default, the color is used in conjunction with the class attribute, but it can be useful to color the other attributes as well. For example, changing the color to the fourth attribute by clicking on the arrow next to the bar that currently reads Color: class (Num) and selecting petalwidth enables us to observe even more about the data, for example the fact that for the class/sepalwidth attribute pair, which range of attribute values (indicated by different color) tends to go along with which class.

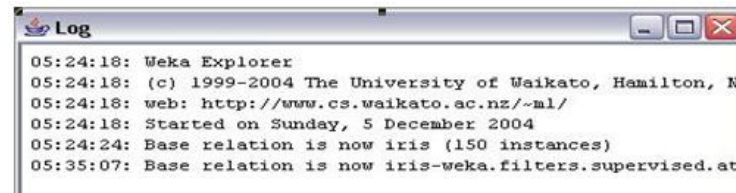
Filters

There are also a number of filters available, which apply different criteria to select either objects (the rows in your data matrix) or attributes (the columns in your data matrix). This allows you to discard parts of your matrix without having to manipulate your original data file. For example, you can look at subsets of attributes, discard the first 20 rows, normalize or discretize attributes and so on. To apply a filter, you first have to select which type of

filter you'd like by clicking on the CHOOSE button right underneath Filter in your main GUI. Double clicking on the FILTER folder that appeared will expand the window to show two folders named supervised and unsupervised, both of which you can expand again. Both unsupervised and supervised filters can be applied to objects and attributes. Once you have chosen a filter, the selected option will show up in the bar next to FILTER, but at this stage, nothing has happened to your data yet. You then have to press apply to actually filter your data. There is also a SAVE button which allows you to save any changes you made to your data. Make sure you don't overwrite your original data file!

The log file

The log file is used to keep track of what you did. Clicking on LOG in your main GUI will bring up another window which will show exactly what you did, in this case it shows that we loaded the Iris data set and applied a filter.

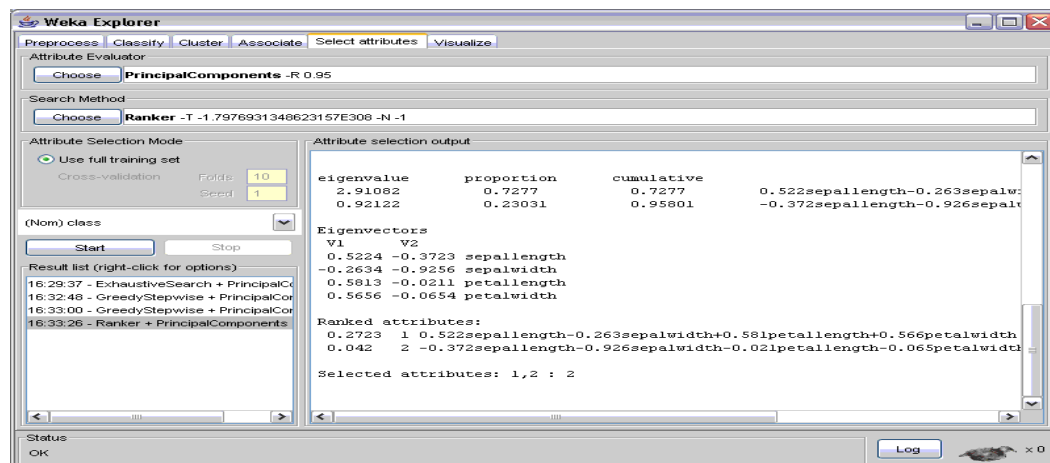


```

05:24:18: Weka Explorer
05:24:18: (c) 1999-2004 The University of Waikato, Hamilton, N
05:24:18: web: http://www.cs.waikato.ac.nz/~ml/
05:24:18: Started on Sunday, 5 December 2004
05:24:24: Base relation is now iris (150 instances)
05:35:07: Base relation is now iris-weka.filters.supervised.at
  
```

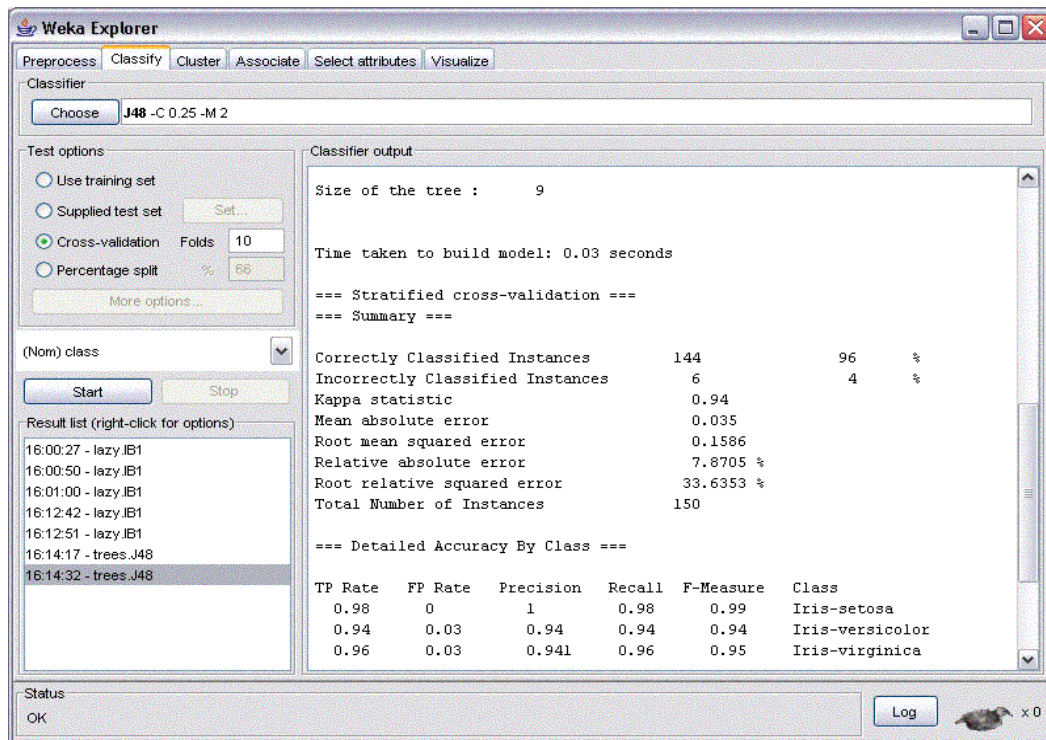
Selecting Attributes

Weka also provides techniques to discard irrelevant attributes or reduce the dimensionality of your dataset. After loading a dataset, click on the select attributes tag to open a GUI which will allow you to choose both the evaluation method (such as Principal Components Analysis for example) and the search method (f. ex. greedy or exhaustive search). Depending on the chosen combination, the actual time spend on selecting attributes can vary substantially and can also be very long, even for small datasets such as the Iris data with only five features (including the class attribute) for each of the 150 samples. The picture below shows the results for a sample application. It is also important to note that not all evaluation/search method combinations are valid, watch out for the error message in the Status bar. There's also a problem using Discretize while in the preprocessing mode, which leads to false results. If you need to use this filter, you can work around this by using the FilteredClassifier option in the classify menu.



Classification

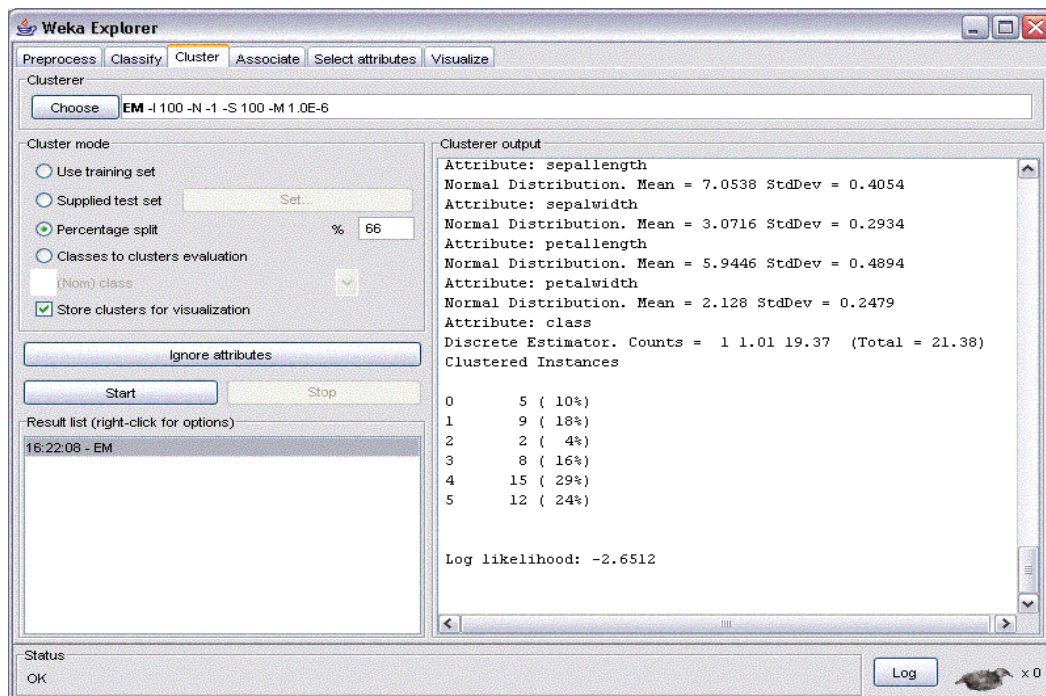
Clicking on the classifier tab after loading a dataset into Weka and selecting the choose tab will bring up a menu with a number of choices for the classifier that is to be applied to the dataset. Note that you have 4 options on how to test the model you're building: Using the test set, a training set (you will need to specify the location of the training set in this case), cross validation and a percentage. The achieved accuracy of your model will vary, depending on the option you select. One pitfall to avoid is to select the training set as a test set, as that will result in an underestimate of the error rate. The resulting model, with a lot of additional information will be displayed after you click on start. What exactly is contained in the output can be determined under options. A sample output for applying the J48 decision tree algorithm to the Iris dataset is shown in the Figure below.



One of the things to watch out for is that the confusion matrix is displayed, as this gives a lot more information than just the prediction accuracy. Other useful things are the options showing up when right clicking the results list on the bottom right. For example, this is where you can load and save the models you built, as well as save the results page. Another fact to keep in mind is that Weka gives hints on how to achieve the same result from the command line: look at what is displayed next to the Choose button and how it changes with the options that you select. This information can also be found towards the top of your results page.

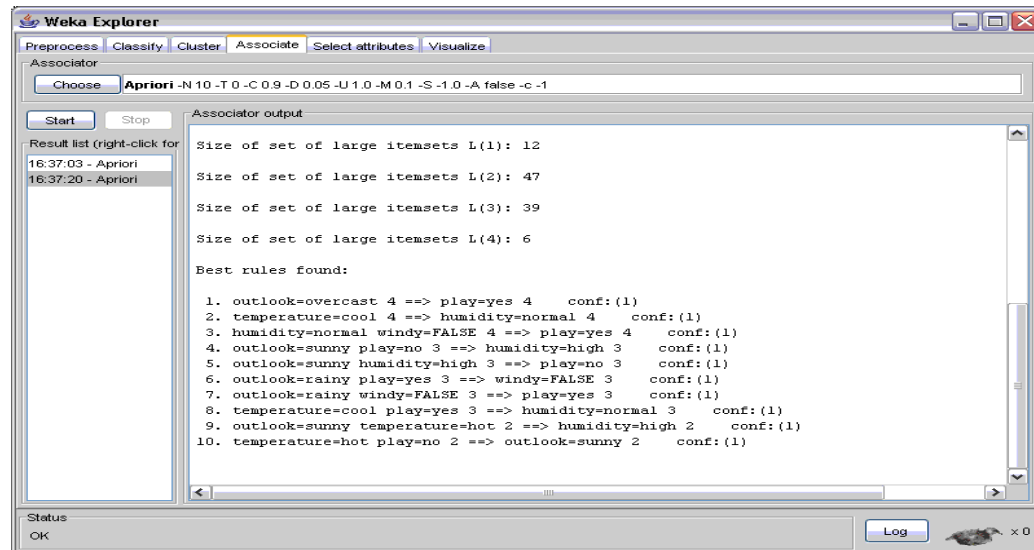
Clustering

The clustering option is very similar to the classification described above, with a few differences regarding the options you select. For instance, there is an easy way to discard undesired attributes.



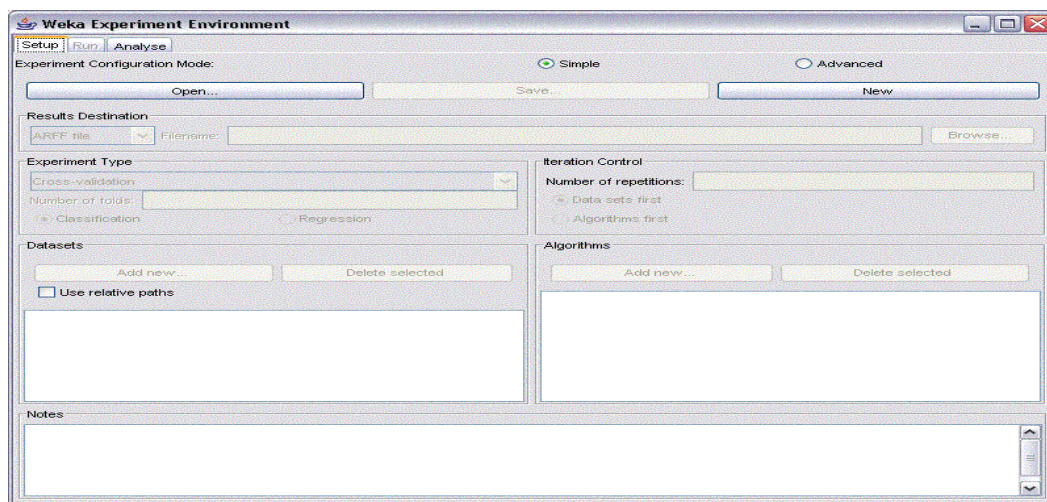
Association Rules

Weka also provides three algorithms to extract association rules from non-numerical data as shown in the picture below.



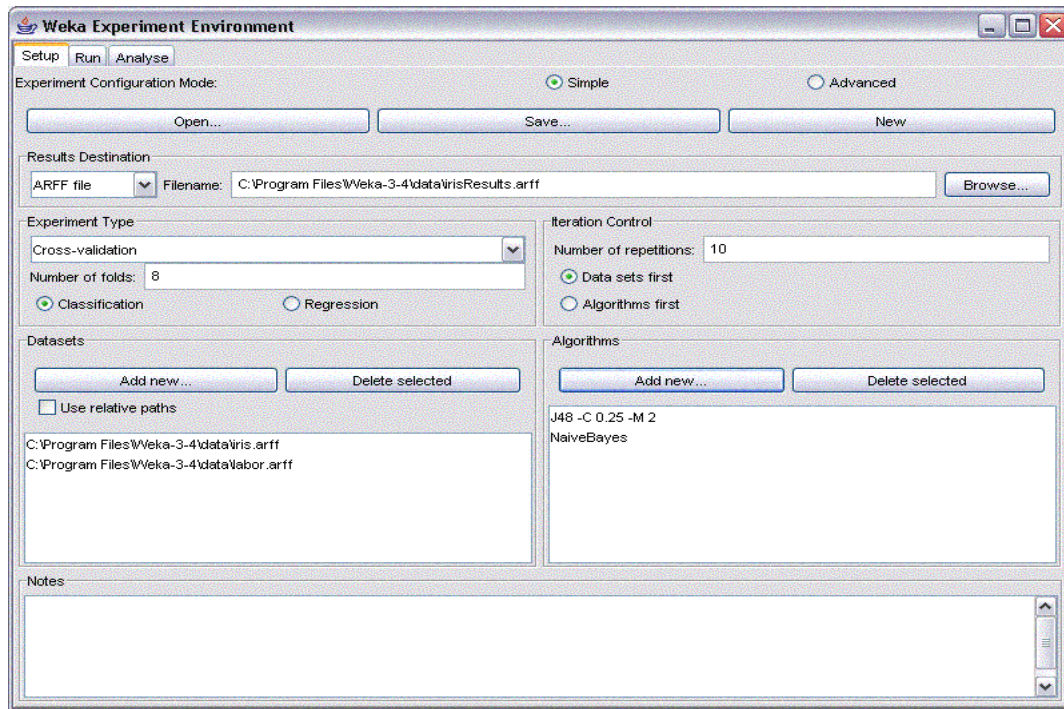
3.3. Experimenter

The experimenter, which can be run from both the command line and a GUI, is a tool that allows you to perform more than one experiment at a time, maybe applying different techniques to a datasets, or the same technique repeatedly with different parameters. The Weka homepage provides a link to a tutorial for an earlier version of the Experimenter, which can be downloaded from [here](#). If you choose the experimenter after starting Weka, you get the following screen.

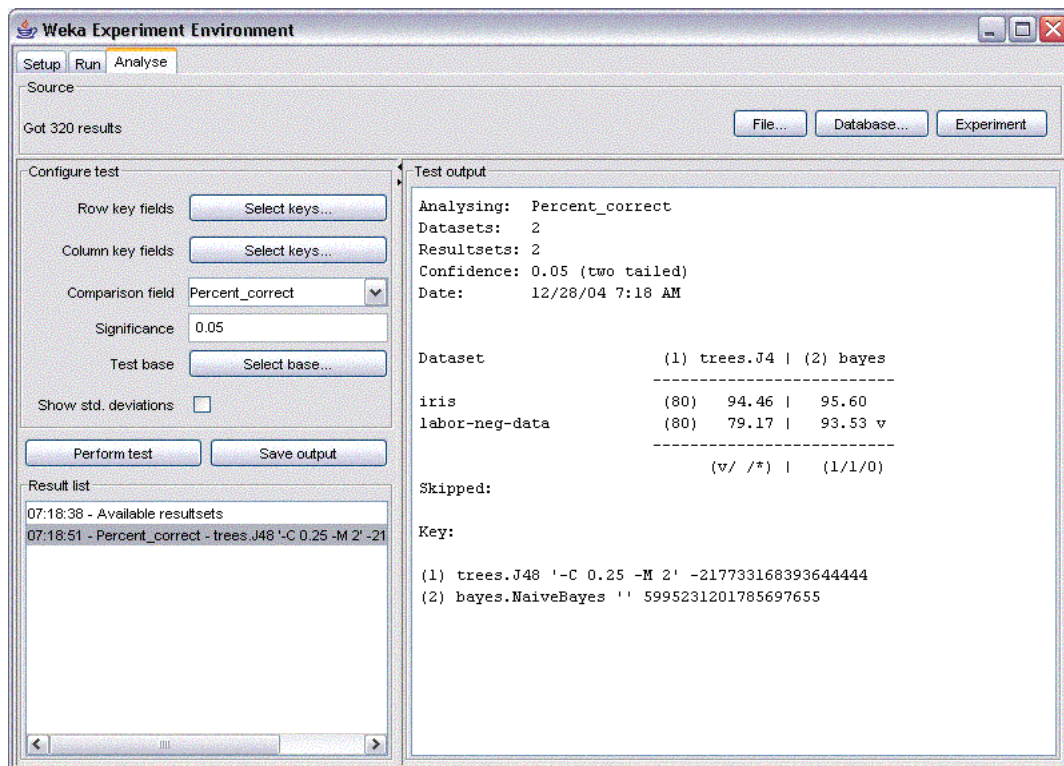


After selecting new, which initializes a new experiment with default parameters, you can select where you want to store the results of your experiment by using browse (there are a number of choices available for the format of your results file). You can then change the default parameters if desired (watch out for the option of selecting classification or regression). For example, you can add more datasets, delete the ones you already selected as well as add and delete algorithms applied to your selected datasets. You can also the type of experiment (cross validation or a percentage split for the training and test set).

The following picture shows the setup for a n 8 fold cross validation, applying a decision tree and Naive Bayes to the iris and labor dataset that are included in the Weka Package. The results are to be stored in an ARFF file called MyResults.arff in the specified subfolder.



After running your experiment by selecting Start from the Run tab, your results will be stored in the specified Results file if the run was successful. You then need to load this file into Weka from the Analysis pane to see your results. The picture below shows the Analysis pane after loading the results file for the experiment set up above.

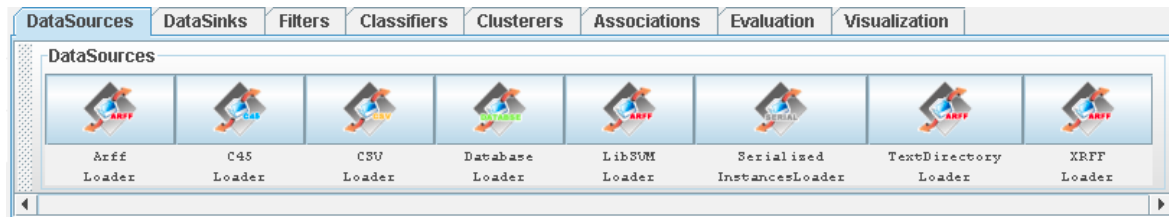


3.4. Knowledge Flow

The knowledge flow is an alternative interface to the functionality provided by the Weka data mining package. WEKA components are selected from a tool bar, positioned a layout canvas, and connected into a directed graph to model a complete system that processes and analyzes data.

Components available in the KnowledgeFlow:

3.4.1. DataSources



- used to indicate where data is coming from
- supports various file types and sources
- configurable for
 - file name of data source
 - dataset or instance (incremental) loading

All of WEKA's loaders are available.

3.4.2. DataSinks



- used to indicate where data is going to be written
- supports various file types and sources
- configurable for
 - file name of data source

All of WEKA's savers are available.

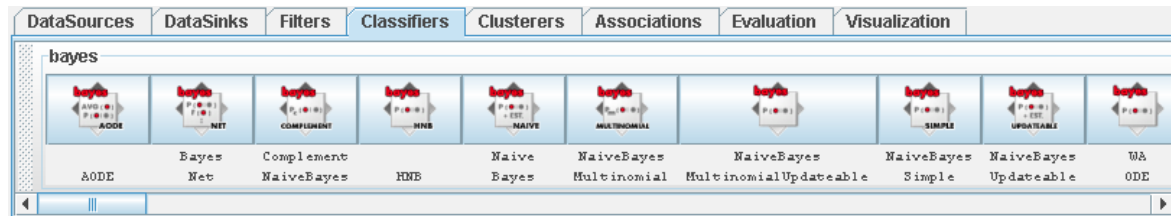
3.4.3. Filters



- used to preprocess data prior to classification or learning
- supports both supervised and unsupervised filters
- configurable depending on filter type

All of WEKA's filters are available.

3.4.4. Classifiers



- supports all classification algorithms presented in the textbook
- parameters are configurable depending on classification algorithm

All of WEKA's classifiers are available.

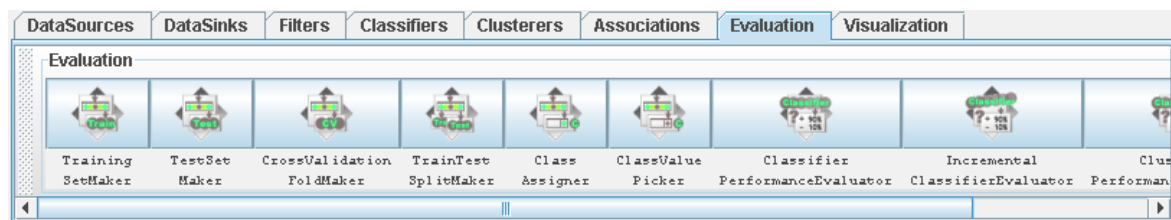
3.4.5. Clusterers



- supports all clustering algorithms presented in the textbook
- parameters are configurable depending on clustering algorithm

All of WEKA's clusterers are available

3.4.6. Evaluation



- used to configure both inputs to and outputs from algorithms
- supports various algorithm performance evaluators
- output format fairly "standardized"
 - TrainingSetMaker - make a data set into a training set
 - TestSetMaker - make a data set into a test set.

- CrossValidationFoldMaker - split any data set, training set or test set into folds.
- TrainTestSplitMaker - split any data set, training set or test set into a training set and a test set.
- ClassAssigner - assign a column to be the class for any data set, training set or test set.
- ClassValuePicker - choose a class value to be considered as the “positive” class. This is useful when generating data for ROC style curves (see ModelPerformanceChart below and example 4.2).
- ClassifierPerformanceEvaluator - evaluate the performance of batch trained/tested classifiers.
- IncrementalClassifierEvaluator - evaluate the performance of incrementally trained classifiers.
- ClustererPerformanceEvaluator - evaluate the performance of batch trained/tested clusterers.
- PredictionAppender - append classifier predictions to a test set. For discrete class problems, can either append predicted class labels or probability distributions.

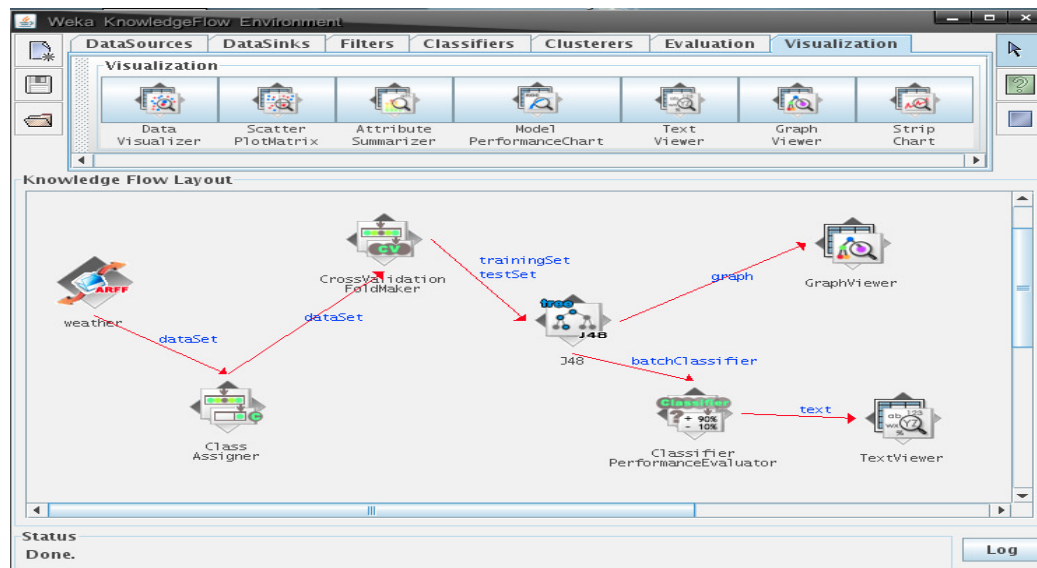
3.4.7. Visualization



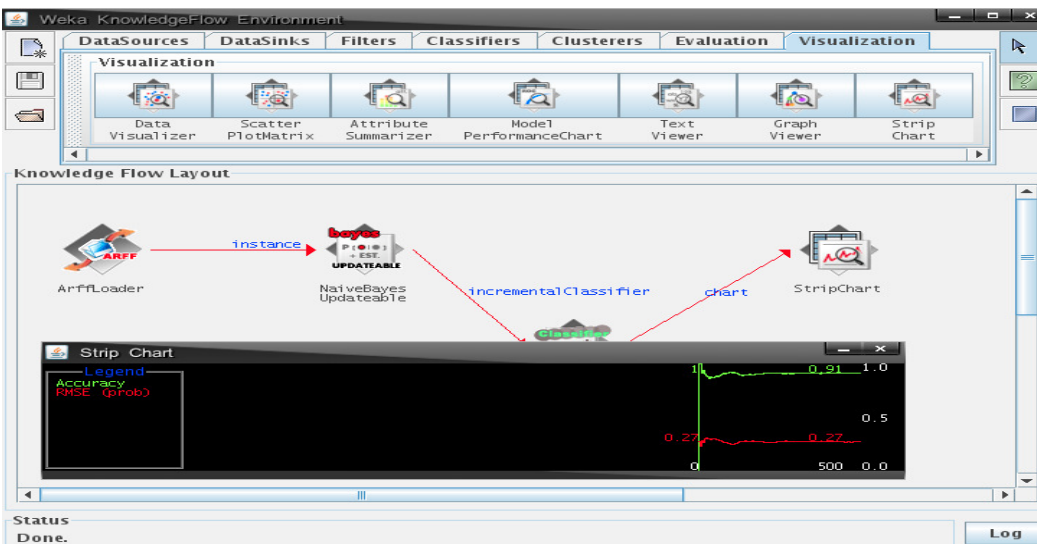
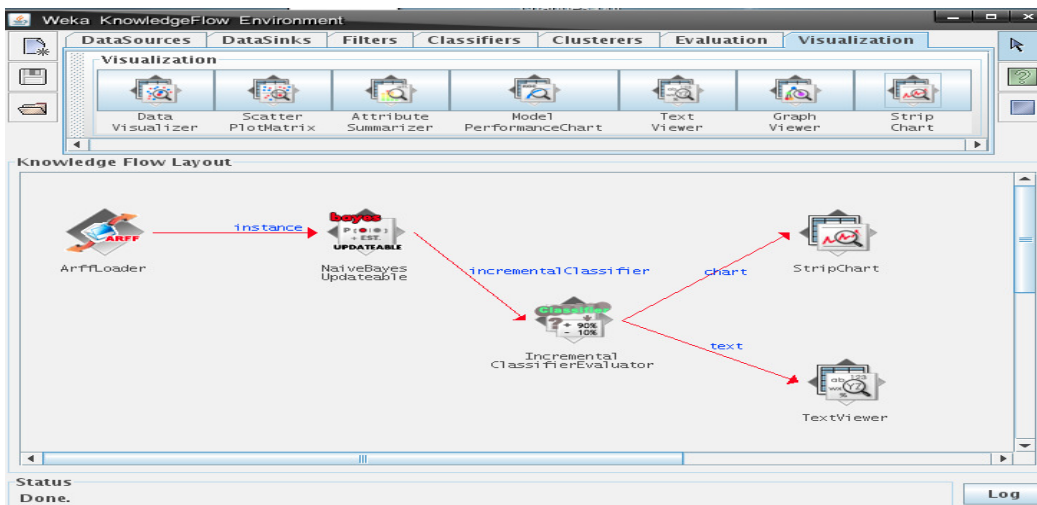
- used to visually display outputs
- supports performance and summaries
- comparable to options from Explorer interface
- DataVisualizer - component that can pop up a panel for visualizing data in a single large 2D scatter plot.
- ScatterPlotMatrix - component that can pop up a panel containing a matrix of small scatter plots (clicking on a small plot pops up a large scatter plot).
- AttributeSummarizer - component that can pop up a panel containing a matrix of histogram plots - one for each of the attributes in the input data.
- ModelPerformanceChart - component that can pop up a panel for visualizing threshold curves.
- TextViewer - component for showing textual data. Can show data sets, classification performance statistics
- GraphViewer - component that can pop up a panel for visualizing tree based models.
- StripChart - component that can pop up a panel that displays a scrolling plot of data (used for viewing the online performance of incremental classifiers).

Example1: Decision Tree Classifier

- 1) Specify a data source
- 2) Specify which attribute is the class
- 3) Specify crossvalidation
- 4) Specify decision tree algorithm
- 5) Specify evaluation
- 6) Specify evaluation output
- 7) To allow viewing of decision trees per fold
- 8) Run experiments



Example2: Incremental Learning



Applications

Weka was originally developed for the purpose of processing agricultural data, motivated by the importance of this application area in New Zealand. However, the machine learning methods and data engineering capability it embodies have grown so quickly, and so radically, that the workbench is now commonly used in all forms of data mining applications—from bioinformatics to competition datasets issued by major conferences such as Knowledge Discovery in Databases.

They worked on:

- predicting the internal bruising sustained by different varieties of apple as they make their way through a packing-house on a conveyor belt;
- predicting, in real time, the quality of a mushroom from a photograph in order to provide automatic grading;
- classifying kiwifruit vines into twelve classes, based on visible-NIR spectra, in order to determine which of twelve pre-harvest fruit management treatments has been applied to the vines;

Weka has been used extensively in the field of bioinformatics. Published studies include automated protein annotation, probe selection for gene expression arrays, plant genotype discrimination, and classifying gene expression profiles and extracting rules from them. Text mining is another major field of application, and the workbench has been used to automatically extract key phrases from text, and for document categorization, and word sense disambiguation.

There are many projects that extend or wrap WEKA in some fashion. There are 46 such projects listed on the Related Projects web page of the WEKA site³. Some of these include:

- *Systems for natural language processing.* There are a number of tools that use WEKA for natural language processing: GATE is a NLP workbench; Balie performs language identification, tokenization, sentence boundary detection and named-entity recognition; Senseval-2 is a system for word sense disambiguation; Kea is a system for automatic keyphrase extraction.
- *Knowledge discovery in biology.* Several tools using or based on WEKA have been developed to aid data analysis in biological applications: BioWEKA is an extension to WEKA for tasks in biology, bioinformatics, and biochemistry; the Epitopes Toolkit (EpiT) is a platform based on WEKA for developing epitope prediction tools; maxdView and Mayday provide visualization and analysis of microarray data.
- *Distributed and parallel data mining.* There are a number of projects that have extended WEKA for distributed data mining; Weka-Parallel provides a distributed cross-validation facility; GridWeka provides distributed scoring and testing as well as crossvalidation; FAEHIM and Weka4WS make WEKA available as a web service.
- *Open-source data mining systems.* Several well known open-source data mining systems provide plugins to allow access to WEKA's algorithms. The Konstanz Information Miner (KNIME) and RapidMiner are two such systems. The R statistical computing environment also provides an interface to WEKA through the Rweka package.
- *Scientific workflow environment.* The Kepler Weka project integrates all the functionality of WEKA into the Kepler open-source scientific workflow platform.

Many future applications will be developed in an online setting. Recent work on data streams has enabled machine learning algorithms to be used in situations where a potentially infinite source of data is available. These are common in manufacturing industries with 24/7 processing. The challenge is to develop models that constantly monitor data in order to detect changes from the steady state. Such changes may indicate failure in the process, providing operators with warning signals that equipment needs re-calibrating or replacing.