Weka: Practical machine learning tools and techniques with Java implementations

AI Tools Seminar

University of Saarland, WS 06/07

Rossen Dimov [1]

Supervisors:

Michael Feld, Dr. Michael Kipp,
Dr. Alassane Ndiaye and Dr. Dominik Heckmann [2]

April 30, 2007

[1] rdimov@mpi-sb.mpg.de
[2] {michael.feld, michael.kipp, alassane.ndiaye, dominik.heckmann}@dfki.de

Abstract

With recent advances in computer technology large amounts of data could be collected and stored. But all this data becomes more useful when it is analyzed and some dependencies and correlations are detected. This can be accomplished with machine learning algorithms.

WEKA (Waikato Environment for Knowledge Analysis) is a collection of machine learning algorithms implemented in Java. WEKA consists of a large number of learning schemes for classification and regression numeric prediction - like decision trees, support vector machines, instance-based classifiers, Bayes decision schemes, neural networks etc. and clustering. It provides also meta classifiers like bagging and boosting, evaluation methods like cross-validation and bootstrapping, numerous attribute selection methods and preprocessing techniques.

A graphical user interface provides loading of data, applying machine learning algorithms and visualizing the built models. A Java interface available to all algorithms enables embedding them in any user's program.

Contents

1 Introduction

Machine learning algorithms serve for inducing classification rules from a dataset of instances and thus broadening the domain knowledge and understanding.

WEKA is a workbench for machine learning that is intended to make the application of machine learning techniques more easy and intuitive to a variety of real-world problems. The environment targets not only the machine learning expert but also the domain specialist. That is why interactive modules for data processing, data and trained model visualization, database connection and cross-validation are provided. They go along with the basic functionality that needs to be supported by a machine learning system - classifying and regression predicting, clustering and attribute selection.

It is developed at the University of Waikato, New Zealand. The project started when the authors needed to apply machine learning techniques on an agricultural problem. This was about twelve years ago. Now version 3.5.5 is available and two years ago the authors have also published a book[4]. This book covers the different algorithms, their possible weak and strong points, all preprocessing and evaluating methods. It also covers a detailed description for all four graphical modules and some basic introduction on how to use the Java interface in your own programs. The project is developed and distributed under the GPL license and has a subdomain on the Sourceforge[1] portal.

This article coves a description of the main features of WEKA version 3.5.5 and an application for spam detection using the Java interface.

Some basic machine learning definitions, used in the forthcoming part follow:

- The ***instances*** are objects from a space of fixed dimension.
- Each dimension corresponds to a so-called ***attribute*** of the object.
- Most often attributes could be ***nominal*** (enumerated) or ***numerical*** (real number) or strings.
- One special attribute is the ***class attribute***, which determines the appurtenance of the instance to a specific group of instances.
- A ***dataset*** is a set of instances.
- ***Training set*** is a set that is used for building a classifier, which is the process of learning something from instances in order to predict the class attribute of new ones.
- ***Test set*** is a set that is used for evaluation a classifier.

[1] http://weka.sourceforge.net/wiki/index.php/Main_Page

An example of data set can be records of days when the weather conditions are appropriate for surfing. The temperature, the humidity, the speed of the wind are attributes that can be measured and can be enumerated and/or numerical. Surfing or not are the values of the class attribute. The record for one single day represents one instance. The classification is used for predicting the value of the class attribute for the future days.

2 WEKA's main features

WEKA consists of four graphical user interface modules available to the user. They are called Explorer, Experimenter, Knowledge Flow and Simple Command Line Interface[3].

2.1 The Explorer

The Explorer is the main module for visualizing and preprocessing the input data and applying machine learning algorithms to it.

2.1.1 Loading data

The data is usually stored in a spreadsheet or database and is also called dataset. Each dataset consists of instances, which are represented by a row in the spreadsheet or the database table.

The native data storage format of WEKA is ARFF (Attribute-Relation File Format)[2]. It consists of a header and data section. The first section contains metadata describing the second. It consists of all instances' attributes and their types. The second section consists of attribute values separated by commas. Starting with version 3.5.x this format is used and an XML-based extension is created - XRFF (eXtensible attribute-Relation File Format)[3]. The metadata and the data section are represented in XML and attribute and instance weights are added.

There are also other data formats supported. All of them can be found in *weka.core.converters* package, which can be extended for more. Data can also be loaded from a database using JDBC and from a URL.

2.1.2 Preprocessing Data

After data is loaded, it is shown in the 'Preprocess' panel of the Explorer. Summary statistics are available for every attribute from the dataset. If the attribute is nominal the distribution of the instances according the attribute values is shown. If the attribute is numerical the minimum, maximum, mean and standard deviation are given.

[2]http://www.cs.waikato.ac.nz/~ml/weka/arff.html
[3]http://weka.sourceforge.net/wekadoc/index.php/en:XRFF_(3.5.4)

Over the dataset simple editing operations, like editing single values for concrete instances and removing columns for all instances, can be done by hand. Automatic operations can be done by filters. Usually the data format needs to be transformed for various reasons depending on the machine learning scheme that will be used. For example a machine learning algorithm might only accept numeric values of the attributes, so all non-numeric attributes should be transformed in order for this algorithm to be used. A filter is chosen from a tree view, which contains all available filters - see figure 1. Each of them has a description about how it works and a reference on all parameters it uses. Most of the filters are explained in detail in the book but since there are newer versions of WEKA new filters are also implemented and can be chosen.

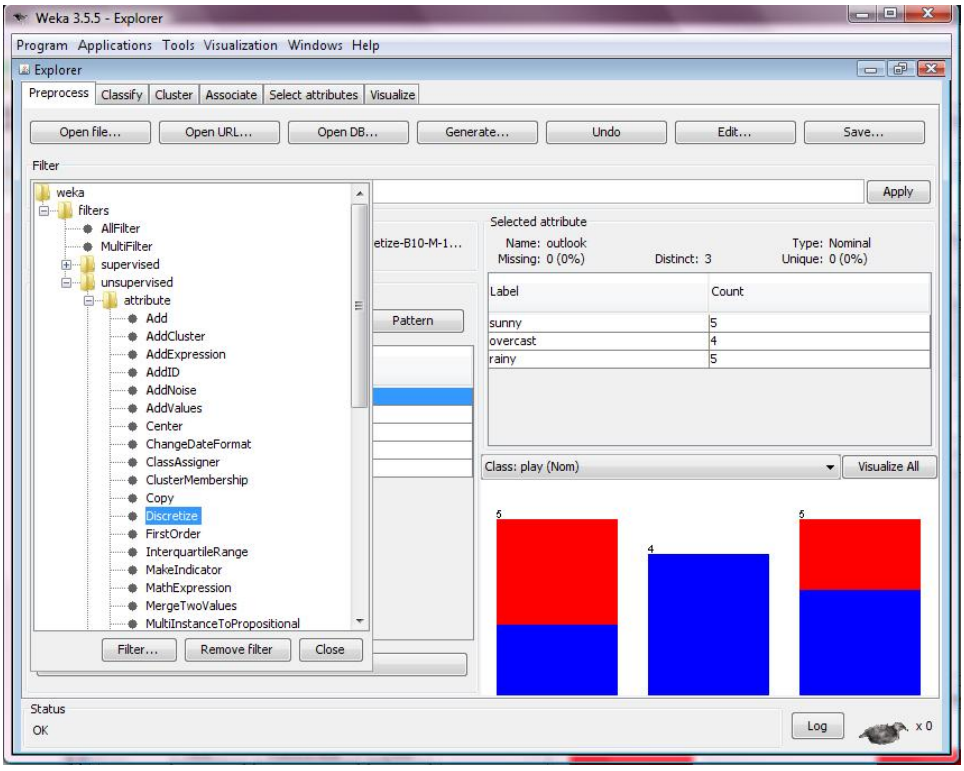

Figure 1: The 'Preprocess' panel.

A list of some filters follows:

Discretize	An instance filter that discretizes a range of numeric attributes in the dataset into nominal attributes
NominalToBinary	Converts all nominal attributes into binary numeric attributes
Normalize	Scales all numeric values in the dataset to lie within the interval [0, 1]
ReplaceMissingValues	Replaces all missing values for nominal and numeric attributes with the modes and means of the training data
StringToWordVector	Converts a string attribute to a vector that represents word occurrence frequencies

After choosing an appropriate filter it can be applied to the initial dataset. The result of this transformation is shown in the 'Preprocess' panel. Consecutive transformations can be applied in case additional preprocessing is needed. The transformed dataset can also be saved as a file.

2.1.3 Building classifiers

After the input dataset is transformed in the format that is suitable for the machine learning scheme, it can be fed to it. Building or training a classifier is the process of creating a function or data structure that will be used for determining the missing value of the class attribute of the new unclassified instances. The concrete classifier can be chosen from the 'Classify' panel of the Explorer. There are numerous classifiers available. Each of them has a description about how it works and a reference for all parameters it uses. Most of the classifiers are explained in detail in the book but since there are newer versions of WEKA new classifiers are implemented and can be chosen.

A list of some classifiers is shown here:

Naïve Bayes	Standard probabilistic Naïve Bayes classifier
J48	C4.5 decision tree learner
MultilayerPerceptron	Backpropagation neural network
IBk	k-nearest neighbour classifier
SVMReg	Support vector machine for regression

The trained classifier can be evaluated either with an additional test set or through k-fold cross validation, or by dividing the input dataset to a training and test set. The result of the evaluation is shown in the 'Classifier output' pane - figure 2. It contains a textual representation of the created

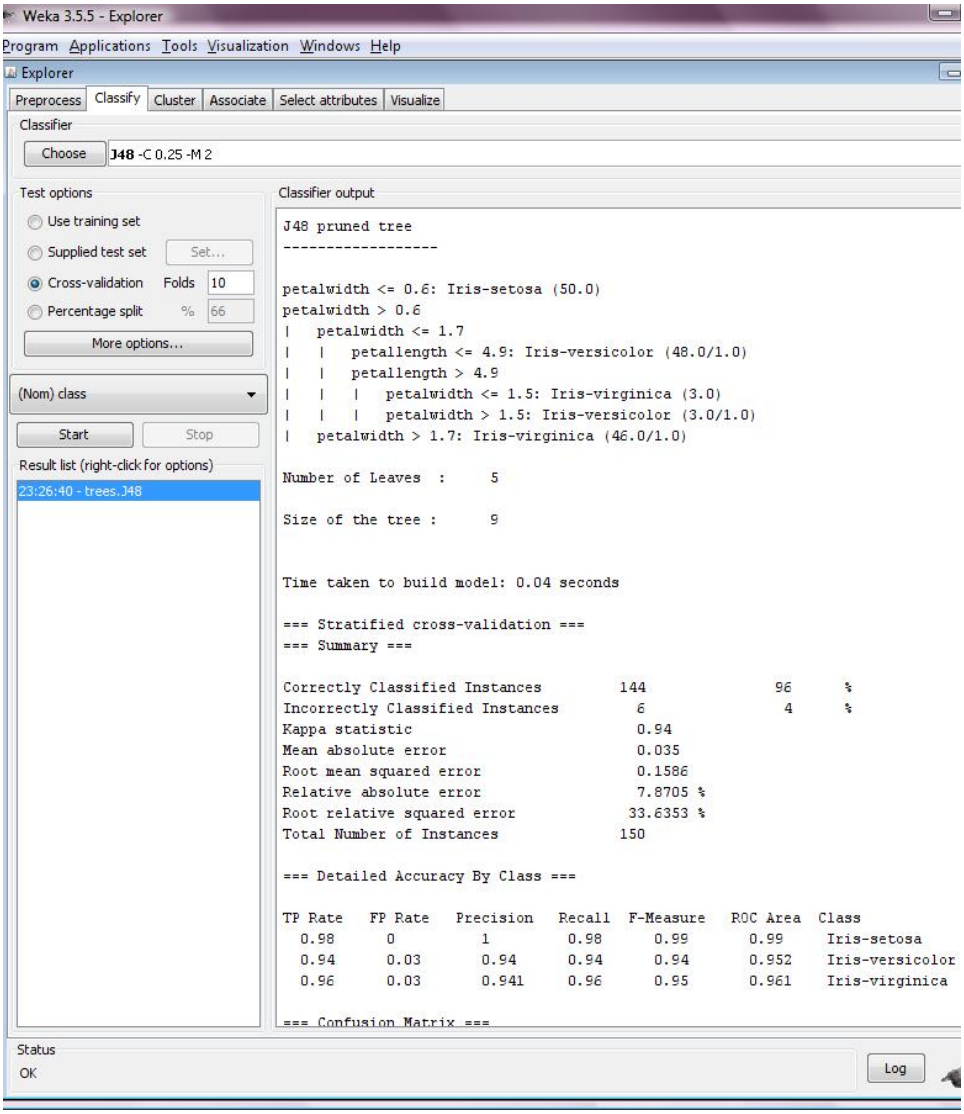

Figure 2: The 'Classify' panel. *J48* classifier with corresponding parameters used, evaluated with 10-fold cross validation. In the output pane a textual representation of the build classifier and some statistics are shown.

model and statistics about the accuracy of the classifier, such as TP (True Positive), FP (False Positive) rate and confusion matrix. TP rate shows the percentage of instances whose predicted values of the class attribute are identical with the actual values. FP rate shows the percentage of instances whose predicted values of the class attribute are not identical with the actual values. The confusion matrix shows the number of instances of each class that are assigned to all possible classes according to the classifier's prediction.

There is a special group of classifiers called *meta-classifiers*. They are used to enhance the performance or to extend the capabilities of the other classifiers.

A list of some important meta-classifiers is shown here:

AdaboostM1	Class for boosting a nominal class classifier using the Adaboost M1 method
Bagging	Class for bagging a classifier to reduce variance
AttributeSelectedClassifier	Dimensionality of training and test data is reduced by attribute selection before being passed on to a classifier
FilteredClassifier	Class for running an arbitrary classifier on data that has been passed through an arbitrary filter

The trained classifier can be saved. This is possible due to the serialization mechanism supported by the Java programming language.

Beside the classification schemes WEKA supply two other schemes - association rules and clustering.

2.1.4 Association rules

There are few association rules algorithms implemented in WEKA. They try to find associations between different attributes instead of trying to predict the value of the class attribute. The interface for choosing and configuring them is the same as for filters and classifiers. There is no option for choosing test and training sets. The results shown in the output pane are quite similar to these produced after building classifier. See figure 3.

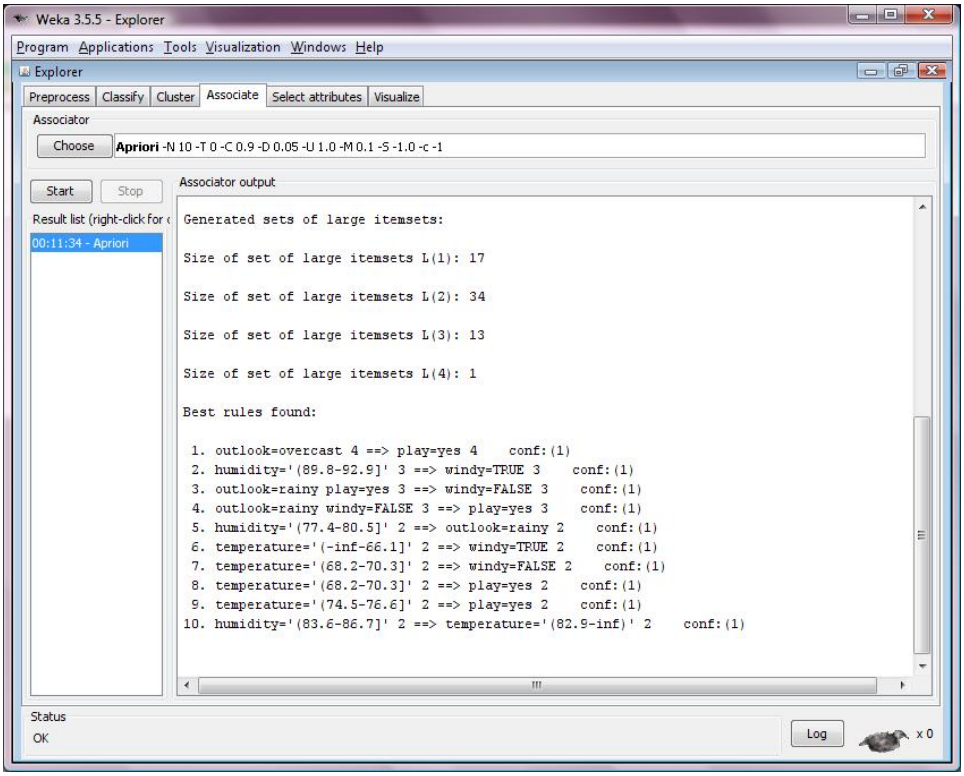

Figure 3: The *Apriori* association rule applied over the training data. All generated rules are shown in the 'Output' pane.

2.1.5 Clustering

There are nine clustering algorithms implemented in WEKA. They also do not try to predict the value of the class attribute but to divide the training set into clusters. All the instances in one group are close, according to an appropriate metric, to all instances in the same group and far from the instances in the other groups. The interface for choosing and configuring them is the same as for filters and classifiers. There are options for choosing test and training sets. The results shown in the output pane are quite similar to these produced after building classifier. See figure 4.

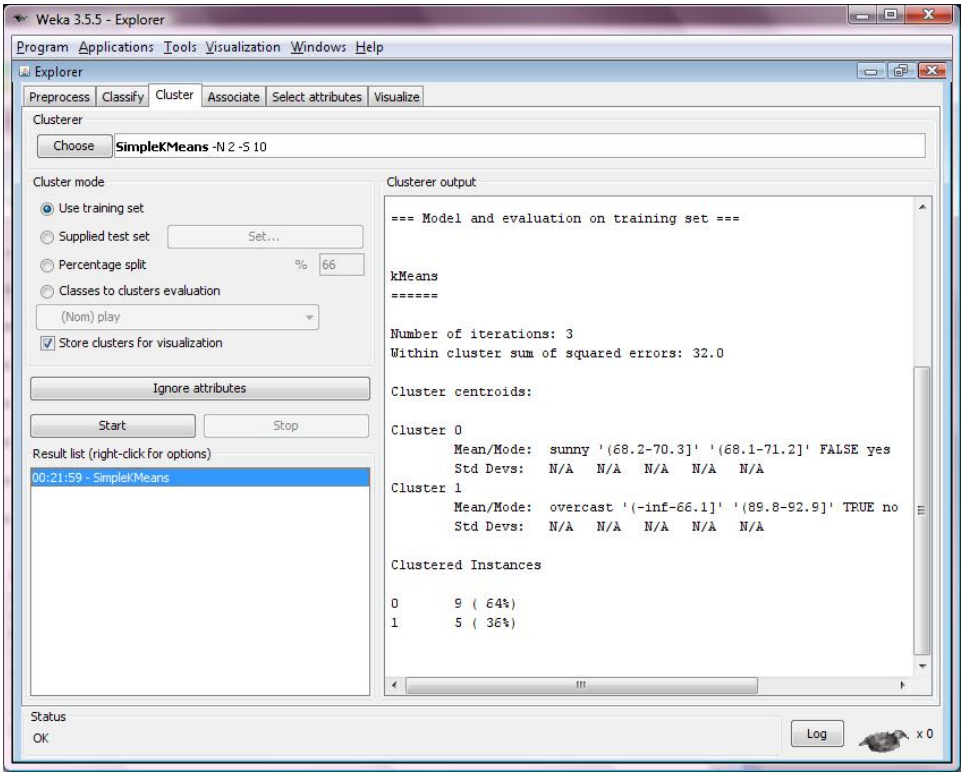

Figure 4: The *SimpleKMeans* algorithm applied over the training data and the two resulting clusters are shown in the 'Output' pane.

A list of some clustering algorithms is shown here:

Cobweb	Generates hierarchical clustering, where clusters are described probabilistically
SimpleKMeans	The algorithm attempts to find the centers of natural clusters in the training data
EM	EM assigns a probability distribution to each instance which indicates the probability of it belonging to each of the clusters

2.1.6 Attribute selection

One important feature of WEKA, which may be crucial for some learning schemes, is the opportunity of choosing a smaller subset from all attributes. One reason could be that some algorithms work slower when the instances have lots of attributes. Another could be that some attributes might not be relevant. Both reasons lead to better classifiers. Determining the relevance of the attributes is searching in all possible subsets of attributes and finding

the one subset that works best for classifying. Two operators are needed - subset evaluator and search method. The search method traverses the whole attribute subset space and uses the evaluator for quality measure. Both of them can be chosen and configured similar to the filters and classifiers. After an attribute selection is performed a list of all attributes and their relevance rank is shown in the 'Output' pane. See figure 5.

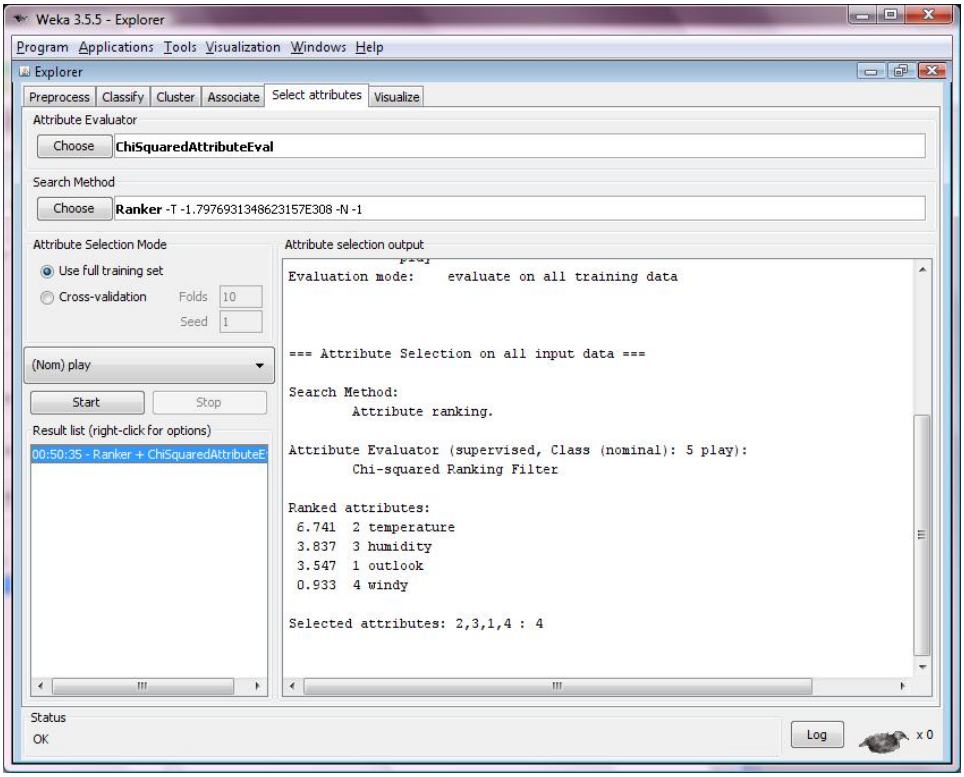

Figure 5: The *ChiSquaredAttributeEval* attribute evaluator and *Ranker* search method ordered the attributes according to their relevance.

Some of the attribute evaluators are shown here:

PrincipalComponents	Performs a principal component analysis and transformation of the data
ChiSquaredAttirbuteEval	Evaluates the worth of an attribute by computing the value of the chi-squared statistic with respect to the class
ReliefFAttributeEval	Evaluates the worth of an attribute by repeatedly sampling an instance and considering the value of the given attribute for the nearest instance of the same and a different class

2.1.7 Visualizing the data

The last panel of the explorer is used for visualizing input data. It shows the 2D distribution of the data. On the X and Y axes, any attribute can be plotted. This way by choosing the class attribute on one of the axes and changing the attribute on the other axis we can see how good or bad each attribute can separate the data. See figure 6.

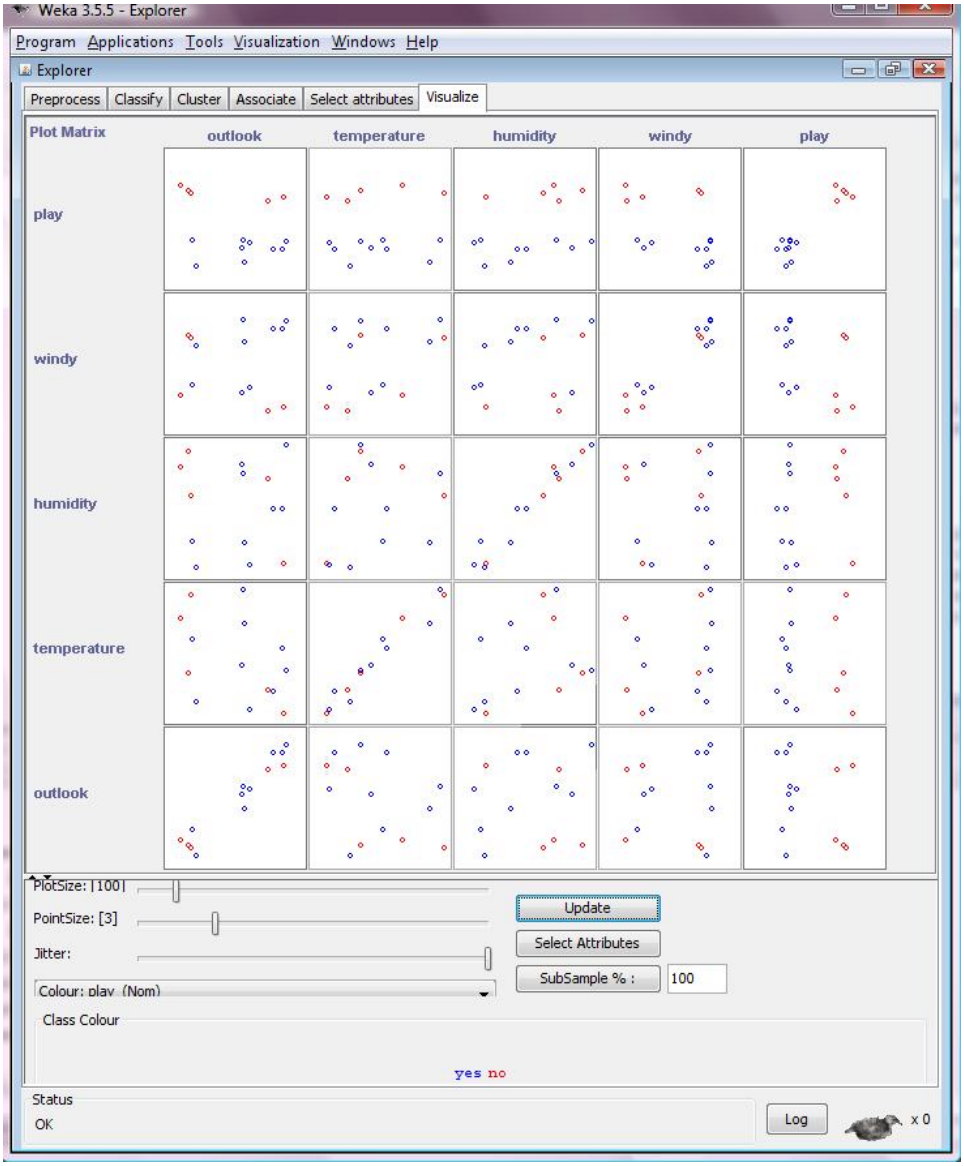

Figure 6: In the 'Visualize' panel all possible 2D distributions are shown.

2.2 The Knowledge flow

Another approach for accessing and using the same functionality but with a drag-and-drop style is the Knowledge flow module. All components, like data loaders, classifiers, clusterers, attribute selectors etc. can be placed on canvas, where they are connected to each other into a graph - see figure 7. Its functionality is similar to that which the Explorer offers, but also adds designing and execution of configurations for streamed data processing. The

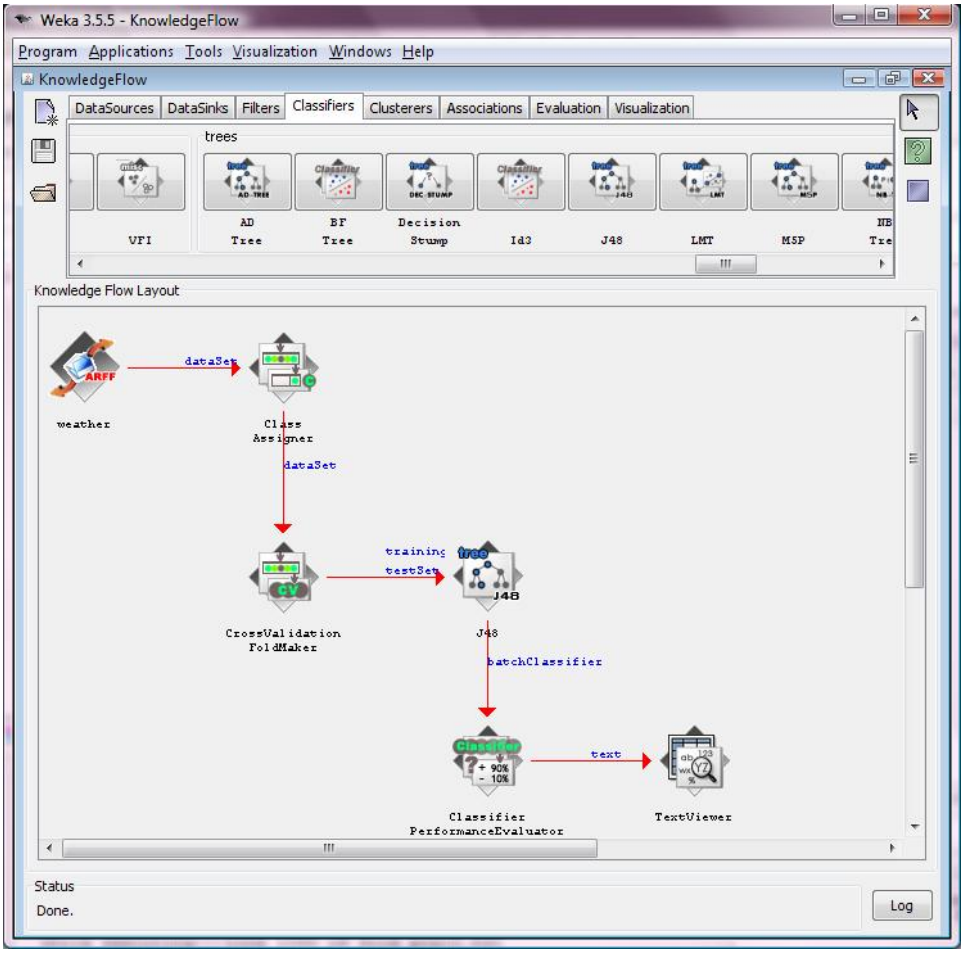

Figure 7: The Knowledge flow

visualization functionality is also more developed. The model of the built classifiers can be seen on each step of the cross validation process. For all classifiers that can handle data incrementally an additional visualizing component - the strip chart - plots the accuracy against the time.

2.3 The Experimenter

The Explorer and the Knowledge flow are not suitable for performing multiple experiments. In order to claim that a learning scheme is better than another they need to be evaluated on different datasets and with different parameter settings- see figure 8. The Experimenter can automate this process and allows analyzing the results.

There are two different approaches for setting up the experiments. The simple one allows choosing datasets, the method of splitting data into train-

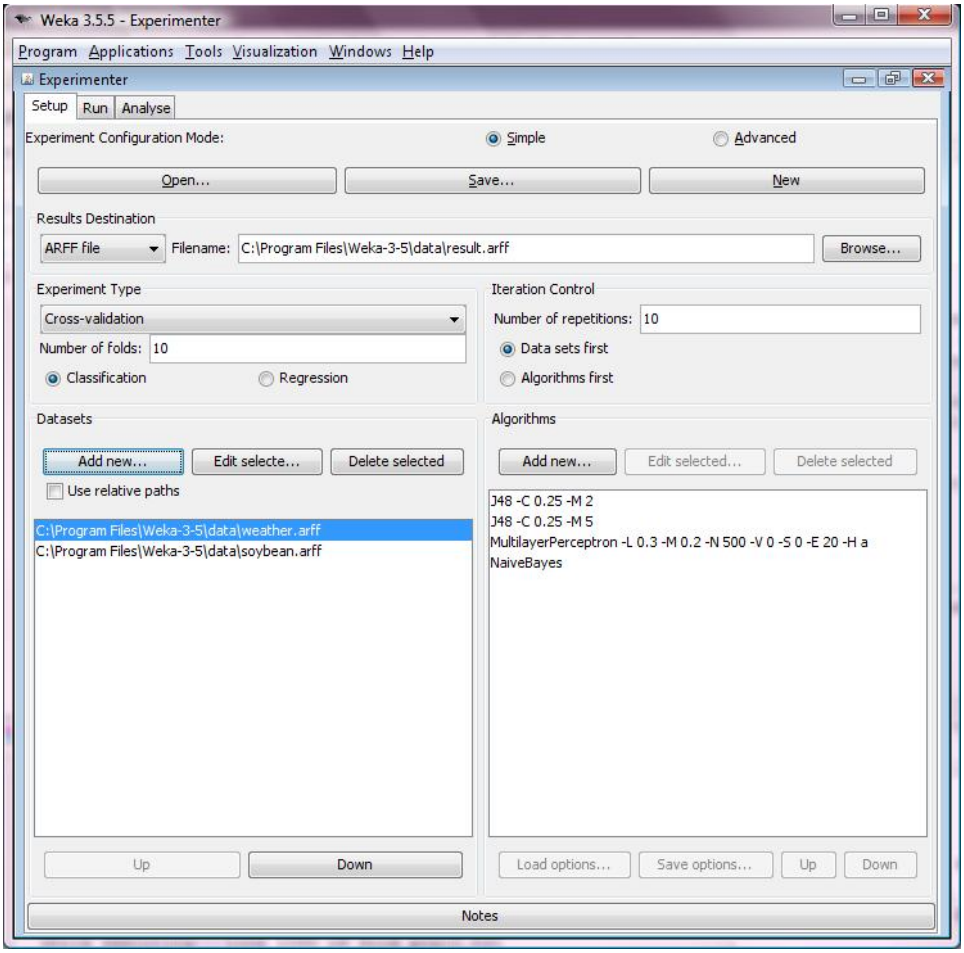

Figure 8: The Experimenter setted up with two different data sets, with three different classifiers while one of them is with two different sets of parameters.

ing and test set and all different learning schemes. In the more advanced one there is an option for performing a distributed experiment using RMI.

Various statistics such as error rate, percentage incorrect can be shown in the 'Analyze' panel and used for finding the better classifier.

2.4 The Simple command line interface

WEKA is not just front-end graphical tool. It is written entirely in Java and all learning schemes are organized in packages. They are used in all GUI components explained before. But any learning scheme has also a uniform command line interface. In the Simple CLI a command can be entered and the output is shown. A command is just invoking a classifier from a

certain package, specifying its options and providing input data. The output looks like the output in the 'Output' pane of the 'Classify' or 'Cluster' or 'Associate' panels of the Explorer.

2.5 The Java interface

WEKA's functionality can also be embedded in any Java application. All of the packages are in *weka.jar*, which comes with the installation package available on the WEKA homepage. The implementations of the algorithms are also available so they can be investigated and modified.

The developers of WEKA are using *javadoc* for the creating documentation of the classes and functions. This documentation is useful, but is missing some details and behaviors for some of the functions. In the book there are examples for the most trivial usage of some basic functionality of the classes. Altogether this means that making something more complicated will involve an extended effort in looking at the implementations of some algorithms and searching in FAQs in the internet. Another issue are some problems with memory management in WEKA. Running some algorithms with big amount of data often leads to *OutOfMemoryException*. On the other hand just using an existing learning scheme is really easy because of the abstraction layer provided by all classifiers. Every classifier extends *weka.classifiers.Classifier* and thus has *buildClassifier* and *classifyInstance*(for nominal classes) or *distributionForInstance* (for numerical classes) methods. These two methods are sufficient for classifying new instances.

3 Experimental results

A small spam filtering application was developed using WEKA's classifiers as a practical assignment for the seminar. The team included three students attending the seminar.

The system reads emails from a directory on the hard drive. Then it builds six classifiers, saves them on the hard drive and evaluates them. The classification step consists of loading one of the saved classifiers and putting labels on emails situated in a user defined folder.

Emails were collected from a repository, containing all emails of a US bankrupted company. They are parsed in order to extract the body and the subject of the email and some other features that we assumed could be important for correct predicting. Namely: are there any HTML tags, are there any Jscripts, is there NoToField, number of recipients and percentage of capital letters. Then from each email an instance was created using all the features as attributes. With the produced instances an initial sets of training and testing data were created.

Then follows data preprocessing in terms of applying WEKA's filter. The filter used is *weka.filters.unsupervised.attribute.StringToWordVector*. It

converts String attributes into a set of attributes representing word occurrence information from the text contained in the strings. A parameter for making some words that occur only in one of the classes more important is set. This way each instance is converted to an instance with about 4000 attributes. Adding the fact that we used 2000 spam and 2000 non-spam emails for building the classifiers this is already a big input training set. After having some *OutOfMemoryExceptions* a decision for applying some attribute selection was taken.

A meta-classifier for reducing the dimensionality of training and test data is used for combining a non-meta classifier with an attribute selection method. All five classifiers - *J48*, *IBk*, *NaiveBayesUpdateable*, *MultilayerPerceptron* and *SMO* - are used together with the *ChiSquaredAttributeEval* attribute evaluator and *Ranker* search method. 250 relevant attributers were chosen and used in the building of these classifiers.

We also implemented a brand new classifier. It extends the base *weka.classifiers.Classifier*, implements the *buildClassifier* and overrides *classifyInstance* methods. It combines the decision tree, the k-nearest neigbours and the support vector machine algorithms already built with reduced number of attributes. For classifying it performs majority voting of the three classifiers.

The evaluation of each classifier is performed with a 10-fold cross validation. The following results are obtained for building time and accuracy:

Classifier	Training time in min	Evaluation with 10-fold cross validation in %
k-nearest neighbours	3	91.8
decision tree	3	94.7
naïve bayes	3	86.6
neural network	20	92.8
support vector machine	3	95.6
meta	9	94.0

The lowest value for the naïve bayes classifier we got was suspicious enough on that phase to think, that somehow the training data is biased. This was proven when we tested our best classifier - support vector machine - with an independent test set. The accuracy was about 50% with a lot of true negative (non-spam mails classified as spam) predictions.

4 Conclusion

The WEKA machine learning workbench provides an environment with algorithms for data preprocessing, feature selection, classification, regression, and clustering. They are complemented by graphical user interfaces for input data and build modes exploration. It supports also experimental comparison of one with varying parameters or different algorithms applied on one or more datasets. This is done in order to facilitate the process of extraction of useful information from the data. The input dataset is in form of a table. Any row represents a single instance and any column represents a different attribute.

Perhaps the most important feature is the uniform Java interface to all algorithms. They are organized in packages and when the weka.jar is added to a standalone project only import section is needed in order to get access to any functionality of WEKA.

WEKA has some memory issues. It expects that the data set will be completely loaded into the main memory, which is not possible for some data mining tasks. It is very slow on large data sets. For k-cross fold validation WEKA creates k copies of the original data. Only one copy exists at a time, but recourses for copying are used in vain.

There are a lot of projects that are using WEKA to some extent or even extend it. One of them is BioWEKA[1], which is used for knowledge discovery and data analysis in biology, biochemistry and bioinformatics. It is an open source project by the Ludwig Maximilians-Universitaet Muenchen. For example, there are filters for translating DNA to RNA sequences and vice versa.

Another project is YALE (Yet Another Learning Environment)[2], which is implemented in the University of Dortmund. It supports composition and analysis of complex operator chains consisting of different nested preprocessing, building of classifiers, evaluating and complex feature generators for introducing new attributes.

References

[1] Jan E. Gewehr, Martin Szugat, and Ralf Zimmer. BioWeka -Extending the Weka Framework for Bioinformatics. *Bioinformatics*, page btl671, 2007.

[2] S. Fischer I. Mierswa und S. Felske O. Ritthoff, R. Klinkenberg. YALE - Yet Another Learning Environment. *In: LLWA 01 - Tagungsband der GI-Workshop-Woche Lernen - Lehren - Wissen - Adaptivitaet, Forschungsberichte des Fachbereichs Informatik, Universitaet Dortmund*, 2001.

[3] I. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. Cunningham. Weka: Practical machine learning tools and techniques with java implementations. 1999.

[4] Ian H. Witten and Eibe Frank. Data mining: Practical machine learning tools and techniques. 2005.