Open-Source Machine Learning: R Meets Weka

Kurt Hornik Christian Buchta Achim Zeileis

Weka?

Weka is not only a flightless endemic bird of New Zealand (Gallirallus australis, picture from Wekapedia)

but also the

Waikato Environment for Knowledge Analysis

(so clearly, this is the most domestic talk of this conference).

Weka

- Comprehensive collection of machine-learning algorithms for data mining tasks
- Provides tools for data preprocessing, regression, classification, clustering, association rules, and visualization
- Implemented in Java (initially: C/TclTk) and released under GPL
- 3 GUIs (“Explorer”, “Experimenter”, “KnowledgeFlow”) and a standardized CLI
- Started in 1992, funded by the NZ government since 1993
- Recently “acquired” by Pentaho, the world’s most popular open-source business intelligence platform
- Several other projects are based on Weka

Why bother?

- Complements the book “Data Mining” by Ian Witten and Eibe Frank (heavily used in CS curricula)
- Implements a variety of methods popular in machine learning and useful, but typically not available for statistical learning (e.g., rule, meta, and lazy learners, CobWeb, DBSCAN, Tertius, . . .)
- Provides de-facto reference implementations, including the key decision tree algorithms C4.5 (called J4.8) and M5 (called M5’)

Interfacing strategies

- Provide specific (hand-crafted) R functions interfacing Weka's functionality using the CLI or R/Java interfaces
- Create R versions of Weka classes and an OOP style interface for Weka's methods
- ...
- Provide general ways of interfacing Weka's core functionality which yield R functions/methods with "the usual look and feel", e.g., a customary formula interface for supervised learners which are called "classifiers" in Wekaspeak

The last is done in our R package **RWeka**, which uses package **rJava** for low-level direct R/Java interfacing.

Interface generators

We provide

- classes for interfaces to Weka classifiers, clusterers, associates, and filters
- functions which generate such interfaces, i.e., return a suitably classed function interfacing a given Weka class

The interface functions returned

- have formals "as usual"
- are suitably classed so that standard R methods can be provided

Implementation is based on the S3 object system.

```
R> library("RWeka")
R> foo <- make_Weka_classifier("weka/classifiers/trees/J48", "bar")
R> foo
An R interface to Weka class 'weka.classifiers.trees.J48',
which has information

  Class for generating a pruned or unpruned C4.5 decision tree. For
  more information, see

  Ross Quinlan (1993). C4.5: Programs for Machine Learning. Morgan
  Kaufmann Publishers, San Mateo, CA.

Argument list:
  (formula, data, subset, na.action, control = Weka_control())

Returns objects inheriting from classes:
  bar Weka_classifier
```

How does this work?

- `make_Weka_classifier()` creates an interface function `foo()` to the given Weka (classifier) class (JNI notation or Java class)
- `foo()` “knows” to be such an interface, and to return objects inheriting from the given “`bar`” and the general “`Weka_classifier`” classes
- All such classifier interfaces have formals

  ```
  formula  data  subset  na.action  control
  ```

- Printing such interface functions uses Weka’s `globalInfo()` method to provide a description of the algorithm being interfaced

- When the interface function is called, a model frame is set up in R which is transferred to a Weka instance object
- The `buildClassifier()` method of the Weka class interfaces is called with these instances
- The model predictions for the training data are obtained by calling the Weka `classifyInstances()` methods for the built classifier and each training instance
- A suitably classed object containing both a reference to the built classifier and the predictions is returned
- Such objects have at least a `print()` method (using Weka's `toString()`) and a `predict()` method for either "classes" (numeric for regression, factor for classification) or class probabilities (using Weka's `distributionForInstance()`)

```
R> m1 <- J48(Species ~ ., data = iris)
R> m1
J48 pruned tree
------------------

Petal.Width <= 0.6: setosa (50.0)
Petal.Width > 0.6
|   Petal.Width <= 1.7
|   |   Petal.Length <= 4.9: versicolor (48.0/1.0)
|   |   Petal.Length > 4.9
|   |   |   Petal.Width <= 1.5: virginica (3.0)
|   |   |   Petal.Width > 1.5: versicolor (3.0/1.0)
|   Petal.Width > 1.7: virginica (46.0/1.0)

Number of Leaves  :         5

Size of the tree :          9
```

Confusion matrix:

```
R> table(iris$Species, predict(m1))
             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         49         1
  virginica       0          2        48
```

Control arguments

Building the classifiers is controlled by Weka options.

These can be queried using `WOW()`, the Weka Option Wizard (works by calling the Weka `listOptions()` method for Weka class interfaced and doing some magic).

These can be set by using `Weka_control()` to set up the `control` argument of the interface function.

These "control lists" essentially produce Weka's command-line option style (`-R -M 5`) from R's typical tag-value style (`(R = TRUE, M = 5)`).

Query J4.8 options using the Weka Option Wizard:

```
R> WOW(foo)
-U      Use unpruned tree.
-C      Set confidence threshold for pruning.  (default 0.25)
        Number of arguments: 1.
-M      Set minimum number of instances per leaf.  (default 2)
        Number of arguments: 1.
-R      Use reduced error pruning.
-N      Set number of folds for reduced error pruning. One fold is used
        as pruning set.  (default 3)
        Number of arguments: 1.
-B      Use binary splits only.
-S      Don't perform subtree raising.
-L      Do not clean up after the tree has been built.
-A      Laplace smoothing for predicted probabilities.
-Q      Seed for random data shuffling (default 1).
        Number of arguments: 1.
```

Now build a J4.8 tree with reduced error pruning `R` and the minimal number `M` of instances set to 5:

```
R> m2 <- foo(Species ~ ., data = iris, control = Weka_control(R = TRUE, M = 5))
R> m2
J48 pruned tree
------------------

Petal.Width <= 0.6: setosa (34.0)
Petal.Width > 0.6
|   Petal.Width <= 1.5: versicolor (32.0/1.0)
|   Petal.Width > 1.5: virginica (34.0/2.0)

Number of Leaves  :         3

Size of the tree :          5
```

Interfaces

Similar interface generators exist for associators, clusterers and filters.

Filters have formula interfaces.

Available interfaces can be listed using `list_Weka_interfaces()`.

Users can register more interfaces as desired.

Users can also create different interfaces (e.g., to provide better `plot()` or `summary()` methods).

All of this works because Weka provides a consistent "functional" methods interface for its learner classes.

```
R> list_Weka_interfaces()
$Associators
[1] "Apriori" "Tertius"

$Classifiers
 [1] "AdaBoostM1"       "Bagging"          "DecisionStump"    "IBk"
 [5] "J48"              "JRip"             "LBR"              "LMT"
 [9] "LinearRegression" "Logistic"         "LogitBoost"       "M5P"
[13] "M5Rules"          "MultiBoostAB"     "OneR"             "PART"
[17] "SMO"              "Stacking"

$Clusterers
[1] "Cobweb"        "DBScan"        "FarthestFirst" "SimpleKMeans"
[5] "XMeans"

$Filters
[1] "Discretize" "Normalize"
```

Handlers

The general-purpose classifier interface approach can also be customized using handlers.

Can be specified as third argument to `make_Weka_classifier()`.

Currently, only option handlers are honored.

Useful for meta-learning (AdaBoostM1, LogitBoost, ...): Weka expects Weka class names for the base learners, but R users need not know these, and can give either the “base names” of the interfaces classes or simply the interface functions.

The standard meta learners are registered with an option handler expanding these class names.

Use AdaBoostM1 with decision stumps:

```
R> m3 <- AdaBoostM1(Species ~ ., data = iris,
+      control = Weka_control(W = "DecisionStump"))
```

(Printing would give lots of output . . .)

```
R> table(predict(m3), iris$Species)
            setosa versicolor virginica
  setosa        50          0         0
  versicolor     0         45         1
  virginica      0          5        49
```

Plotting

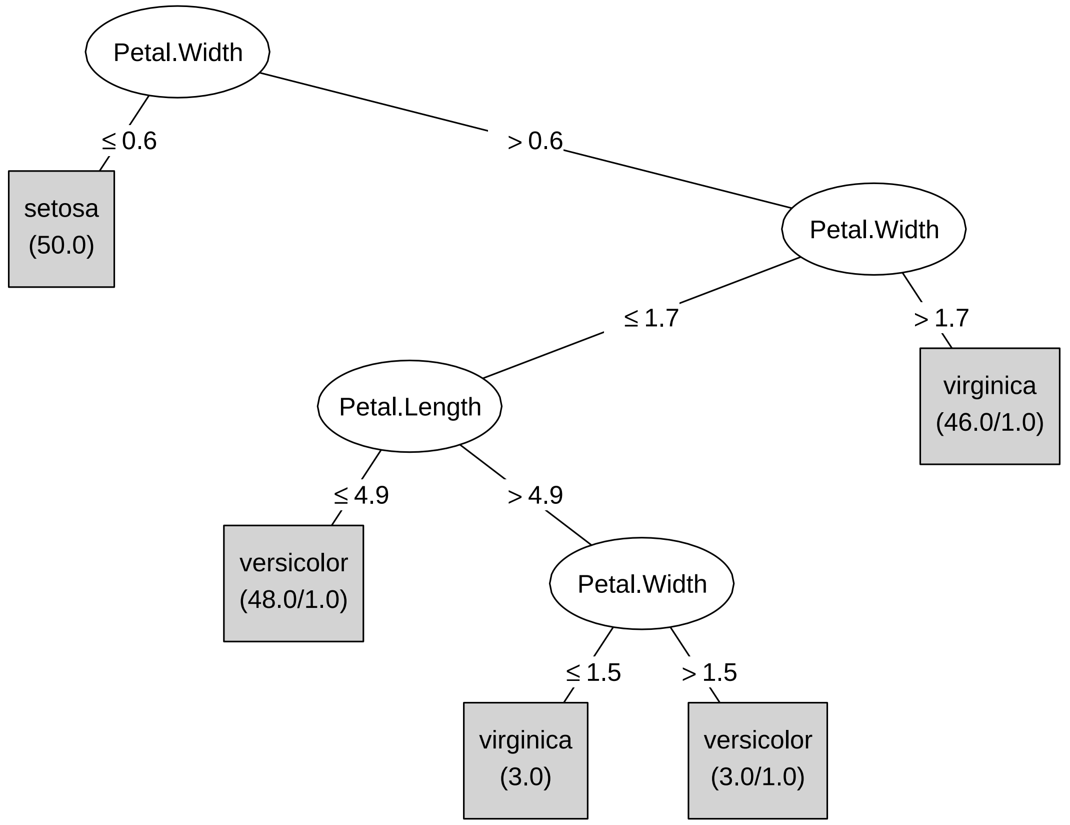
For the Weka tree learners (J48, M5P, LMT) registerd by default, `plot()` methods are based on the routines for “`BinaryTree`” objects in package **party**.

For Weka classifiers with a “Drawable” interface, i.e., providing a `graph()` method, one can create DOT language representations of the built classifiers for processing via **Graphviz**. This can also be used for visualization via **Rgraphviz**.

Could also use Weka’s native plotting facilities. Currently only experimental, as not integrated into R’s display lists (and hence possibly confusing . . .)

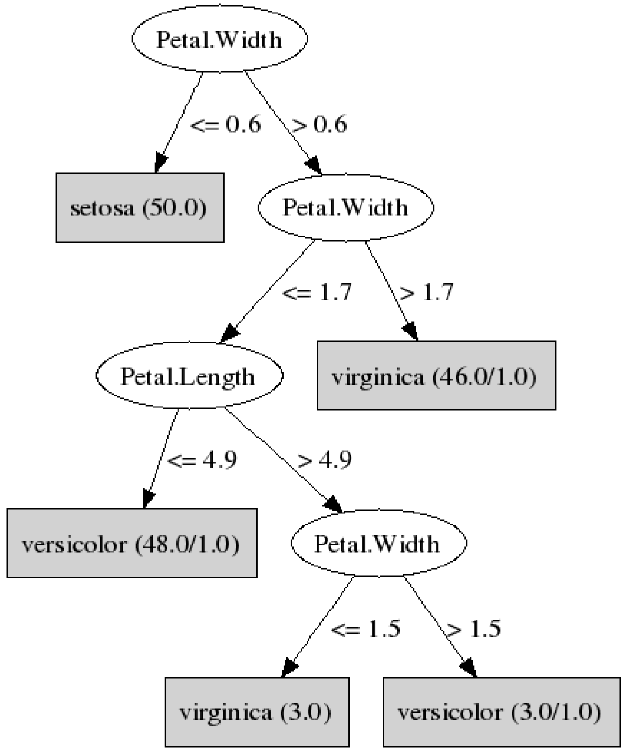
Plotting via **party**:

```
R> plot(m1)
```



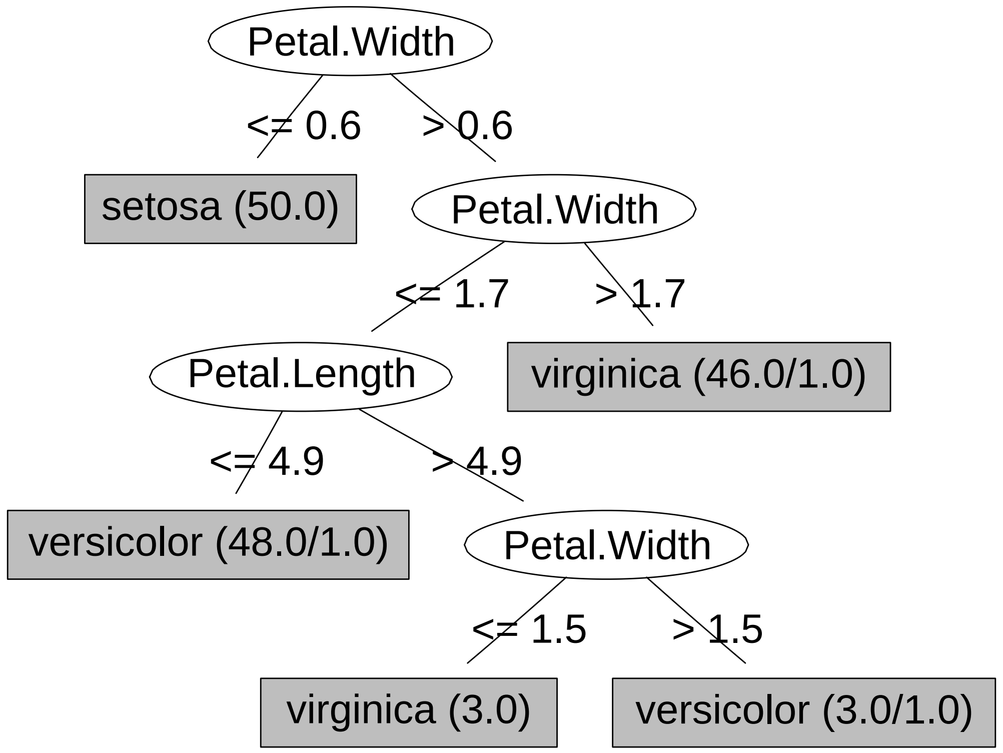
Plotting via **Graphviz**:

```
R> write_to_dot(m1, "m1.dot")
R> system("dot -Tpng m1.dot > m1.png")
```



Plotting via **Rgraphviz**:

```
R> library("Rgraphviz")
R> plot(agread("m1.dot"))
```



Performance measures

Function `evaluate_Weka_classifier()` employs Weka's powerful "`Evaluation`" class for computing model performance statistics for fitted classifiers (by default on the training data).

The default is currently used for `summary()` methods.

```
R> evaluate_Weka_classifier(m1)
=== Summary ===

Correctly Classified Instances         147               98      %
Incorrectly Classified Instances         3                2      %
Kappa statistic                          0.97
Mean absolute error                      0.0233
Root mean squared error                  0.108
Relative absolute error                  5.2482 %
Root relative squared error             22.9089 %
Total Number of Instances              150

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = setosa
  0 49  1 |  b = versicolor
  0  2 48 |  c = virginica
```

Internals

RWeka consists of

- High-level R code for interface generators and reporters, and useful methods, based on low level R/Java interface by package **rJava**
- The Weka jar file (currently, development release 3.5.4)
- A jar with some Java-level interface code (unlike R's data frames, Weka's instance objects are organized rowwise, and predictions are for single instances: for performance, we use Java code for looping over all instances).

Too much privacy

Weka does not provide all methods statisticians would typically expect, e.g.,

```
R> LinearRegression(weight ~ feed, data = chickwts)
Linear Regression Model

weight =

     73.4538 * feed=linseed,soybean,meatmeal,casein,sunflower +
     43.2552 * feed=meatmeal,casein,sunflower +
     49.3409 * feed=casein,sunflower +
    160.2
```

but one cannot access the terms structure (let alone control “the usual way”) short of reverse engineering the output.

Cooperation is necessary and possible: e.g., Weka 3.5.4 has added a `getAllTheRules()` method to access the found association rules (integration to package **arules** is under way).

Common data objects

When using Weka's filters (e.g., `Discretize()`), we do

$$\text{data frame} \rightarrow \text{Weka instances} \boxed{\text{filter}} \text{Weka instances} \rightarrow \text{data frame}$$

Even if data transfer is made efficient by avoiding temporary Weka's native ARFF files (currently experimental): transfers should be avoided, e.g., when filtering the data again or building a classifier on the filtered data.

Natural idea: common R/Java data objects which are references to the data and transfer to the other side only when necessary.

Simpler: suitably classed R objects which when evaluated transfer data back into R, but can be dispatched upon without evaluation.

Not possible given R's current semantics (says Master Luke).

General issue for R interfaces and proxy objects.

Weka interfaces to R

Why not use R classifiers as base learners for Weka's meta learners?

Would need a Weka class representing R regression/classification models which provides the basic methods (`buildClassifier()`, `classifyInstance()`) by calling back into R.

Can this be done both reasonably generally and efficiently? (Remember e.g. that `classifyInstance()` operates on one instance at a time.)

Coordinates

Kurt Hornik, Christian Buchta, Achim Zeileis
Wirtschaftsuniversität Wien
Augasse 2–6, A-1090 Wien, Austria

E-mail: *Firstname*.*Lastname*`@wu-wien.ac.at`
WWW: `http://statmath.wu-wien.ac.at/~hornik/`
`http://statmath.wu-wien.ac.at/~zeileis/`