LAD-WEKA Tutorial

Version 1.0

March 25, 2014

Tibérius O. Bonates
tb@ufc.br
Federal University of Ceará, Brazil

Vaux S. D. Gomes
vauxgomes@gmail.com
Federal University of the Semi-Arid, Brazil

---

1 Requirements

LAD-WEKA version 1.0 is an extension of the official WEKA software, version 3.6.10. As such, it does contain all features of WEKA 3.6.10, in addition to a new algorithm: the LAD classifier. A certain level of familiarity with WEKA is required in order to proceed with the installation and use of LAD-WEKA. In particular, it is required that the user be able to run WEKA and to identify whether or not their current WEKA installation is functional[1].

To avoid compatibility issues related to the version of the Java Runtime Environment (JRE) running in your computer, we strongly advise you to install LAD-WEKA on top of an existing WEKA 3.6.10 installation, as described in the next section. In case you prefer not to follow this procedure, simply make sure that you have the Java Runtime Environment, version 1.6 or higher, installed in your computer.

The figures that accompany the installation instructions below were produced for the Windows platform. However, Linux and MacOS users should be able to proceed in an analogous way in order to install the software in their systems.

2 Installation

First, make sure that you have a functional installation of WEKA 3.6.10. The version number is important, as using a different version might result in an incorrect installation.

[1] We refer users with no familiarity with the WEKA package to the official WEKA website (at http://www.cs.waikato.ac.nz/ml/weka/), which contains a wealth of information on the existing algorithms, as well as installation and troubleshooting instructions.

WEKA can be obtained from its official download webpage at `http://www.cs.waikato.ac.nz/ml/weka/downloading.html`.

WEKA 3.6.10 is often installed in a folder called `Weka-3-6` (in Windows, the complete path will typically be `\Program Files\Weka-3-6`). During the installation process, you might be prompted to confirm the location (or select a different location) where WEKA will be installed. We shall refer to this location as the "WEKA folder."

Installing LAD-WEKA involves replacing the `weka.jar` file within the WEKA folder of your existing WEKA 3.6.10 installation. To accomplish that, simply follow the instructions shown in Figure 1.

1. Open the WEKA folder corresponding to your existing WEKA 3.6.10 installation.
2. Rename the file `weja.jar` to a name of your choosing (e.g., `weka-previous.jar`).
3. Copy the LAD-WEKA JAR file (called `LAD-WEKA.jar` and obtained from LAD-WEKA's website) to the WEKA folder.
4. Finally, rename the recently copied LAD-WEKA JAR file to `weka.jar`.

Figure 1: Instructions for installing LAD-WEKA.

In Figure 2, we show the outcome of this process.

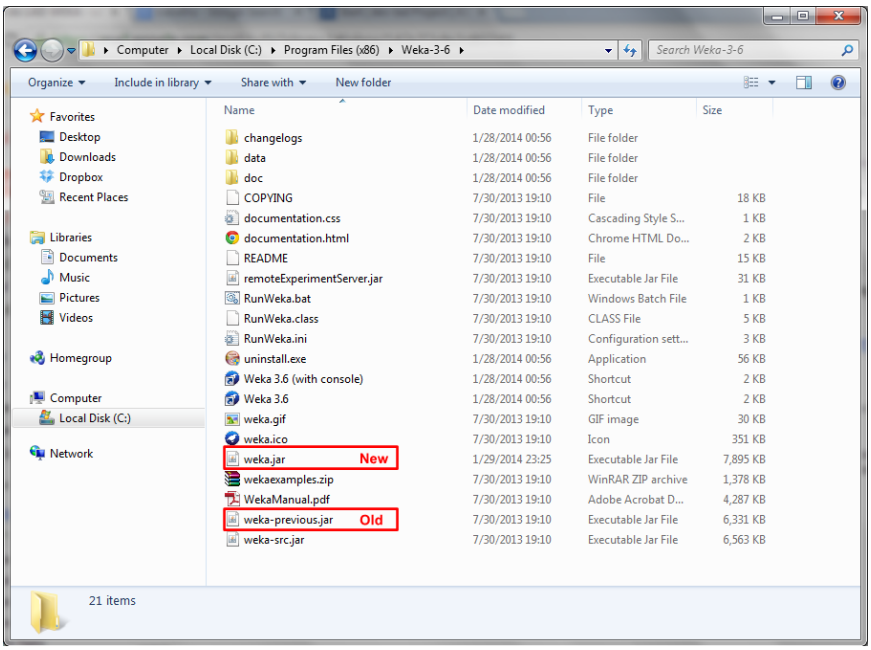

Figure 2: Replacing the `weka.jar` file.

3 Running LAD-WEKA

In order to run LAD-WEKA, simply proceed exactly as you did prior to executing the process described in the previous section. In other words, run WEKA as usual. This involves double-clicking on a WEKA icon, or choosing a WEKA shortcut within the operating system's main menu.

A less common way of running WEKA (or LAD-WEKA) is invoking it from the operating system's command line. In order to do that, the user is required to open a terminal window, change the working folder to the WEKA folder, and directly call Java to start the main GUI component of WEKA, by invoking

```
java -classpath weka.jar weka.gui.GUIChooser
```

Regardless of the procedure followed to launch LAD-WEKA, the outcome of the process is the window shown in Figure 3.

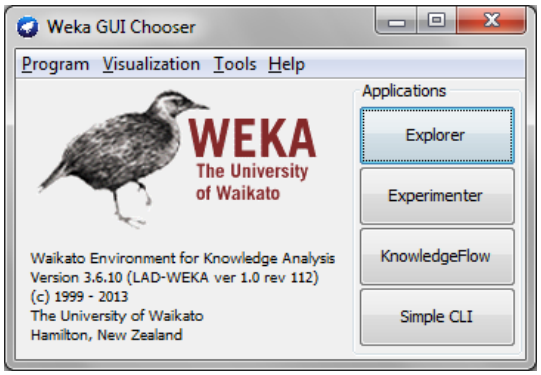

Figure 3: WEKA's main window (which looks precisely as LAD-WEKA's).

4 Using the LAD classifier

In this section we describe two ways of running the LAD classifer available in LAD-WEKA: using WEKA's graphical user interface (Explorer module) and the system's command line. Note that running the LAD classifier from the WEKA Experimenter module is essentially the same process as running it from the Explorer module.

4.1 Running LAD via the WEKA Explorer Window

From the main window shown in Figure 3, the user can click on the "Explorer" button to start the Explorer module of WEKA. The Explorer is an interactive environment from which one can access all classification algorithms available from WEKA. It is useful for running individual experiments with specific data sets and algorithms and for fine tuning the algorithms' parameters. The Experimenter module is the choice for running longer series of experiments in batch mode.

The classifier `weka.classifiers.rules.LAD` is our implementation of the Logical Analysis of Data (LAD) classifier. It can be found in the Explorer module, under the "Classify" tab. By clicking the "Choose" button and selecting the "rules" folder, as shown in Figure 4, the user can find the

LAD classifier.

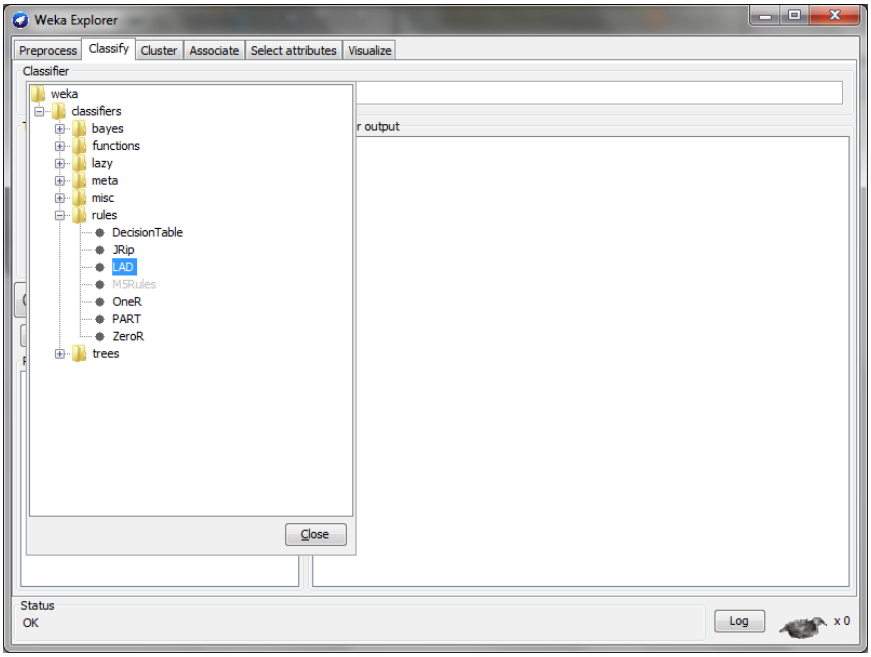

Figure 4: Selecting the LAD classifier in WEKA Explorer.

4.2 Running LAD via the Command Line

If you want to run just the LAD class through command line you will need to be familiar to the set of tokens used to set the WEKA's configurations and the LAD classifier itself. By invoking:

```
java -classpath weka.jar weka.gui.rules.LAD
```

You will be able to see the tokens related to the WEKA and to the LAD classifier.

On the next section you can get a better understanding about the tokens related to the LAD classifier. In this section we are showing just a simple example of how to run the LAD classifier through the command line using the standard configurations and one of the many data sets found in the WEKA's data folder (within WEKA's installation folder):

```
java -classpath weka.jar weka.gui.rules.LAD -t data\vote.arff
```

4.3 LAD Parameters

LAD-WEKA's implementation of LAD supports data sets with numerical features and with exactly **two classes**, i.e., data sets in which each observation, or data point, belongs to one of two groups. The LAD implementation is comprised of three phases: *binarization*, *feature selection* and *rule generation*. Thus, the implementation of the LAD classifier adheres to the standard practice of Logical Analysis of Data. For more details, we refer the user to the LAD literature (please verify the official

LAD-WEKA website at `http://www.lia.ufc.br/~tiberius/lad`).

Figure 5 shows the main LAD-WEKA configuration window. In the following sections we explain what each parameter means and provide additional details on each phase separately.

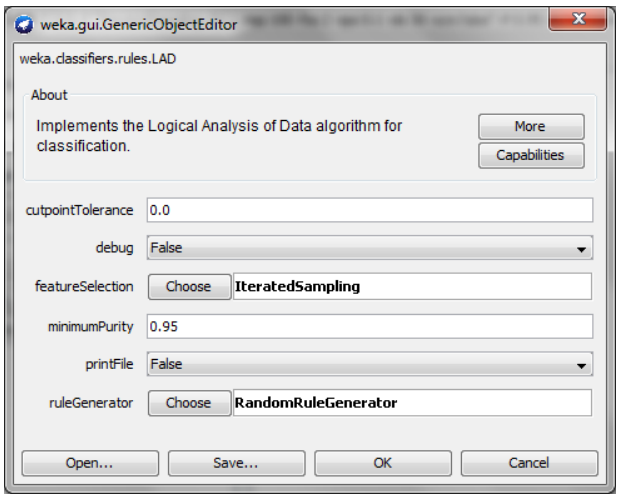

Figure 5: LAD's main configuration window.

4.3.1 Binarization

The binarization phase, as its name suggests, is responsible for transforming the training data set into a binary representation. This means replacing each numerical feature of the original data set by one or more binary features, i.e., features that assume one of two values: 0 and 1. If the original data set is already binary, the binarization phase has no effect.

We make use of *cutpoints* in order to establish threshold levels, based on which the numerical features are binarized. A cutpoint for a given feature is a value that can be used to (partially) distinguish between pair of instances of different classes. This means that each cutpoint is a value at which a transition from one class to the other takes place, at least as long as we are concerned with the given training data. In practical terms, this means that if α is a cutpoint for feature F, then most of the data points from the data set that satisfy $F \geq \alpha$ belong to the same class (while most of the points satisfying $F < \alpha$ belong to the opposite class). The same feature can admit more than one cutpoint. The set of cutpoints generated in the binarization phase is used to map the original instances of the training set to their binary form.

The only parameter setting related to the the binarization phase is:

- **cutpointTolerance**: Tolerance for cutpoint generation. A cutpoint will only be generated between two values of a given feature if such values differ by at least the value of *cutpointTolerance*. (Default = 0.0).

Using the deafult parameter may result in a large number of cutpoints and may resulting in a slowdown in the phases of feature selection and rule generation. The total number of cutpoints generated during the binarization phase can be reduced by increasing the value of *cutpointTolerance*.

4.3.2 Feature Selection

The set of cutpoints produced during the binarization phase might be redundant or excessively large. LAD-WEKA's feature selection phase provides two ways of reducing the number of cutpoints used, so that the rule generation phase can focus on a smaller set of "essential" cutpoints: **GreedySetCovering** (GSC); and **IteratedSampling** (IS). Both methods use Chvátal's greedy heuristic to produce feasible solutions to set covering problems. The former consists of a single application of the heuristic, while the latter builds many set covering problems by sampling the universe of cutpoints and instances from the training set. Figure 6 shows both configuration windows for those feature selection methods.

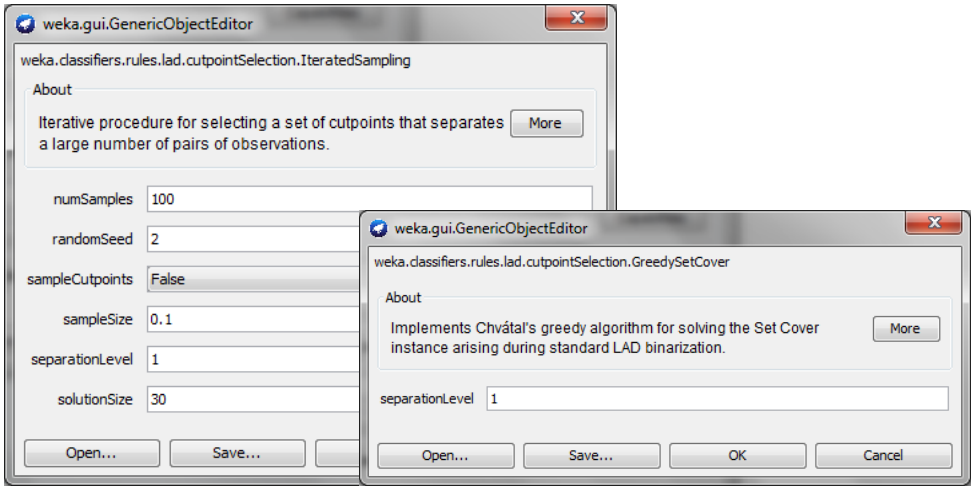

Figure 6: Feature selection configuration windows

Below we list the parameter settings related to the feature selection phase. The only parameter involved in the GSC algorithm is **separationLevel**; all the remaining paremeters are only relevant to the IS algorithm.

- **numSamples**: This is the number of times the cutpoint selection subproblem will be sampled.
- **randomSeed**: This value is used as seed to the pseudo-random number generator employed during the sampling process.
- **sampleCutpoints**: This is a flag (true/false) indicating whether or not cutpoints are sampled (in addition to observations) when forming each cutpoint selection subproblem. If set to false, all cutpoints separating the sampled observations are included in each subproblem.
- **sampleSize**: This specifies the fraction of the cutpoint selection problem that is sampled in order to create each subproblem. The same percentage factor is used to sample a subset of the constraints (pairs of observations) of the cutpoint selection problem. Half of the observations selected are positive, half are negative, and all pairs are used as constraints. If **sampleCutpoints** is set to true, a subset of the problem's variables (cutpoints) is also sampled. In either case, sampling is done uniformly and without replacement.
- **separationLevel** [*common to both methods*]: Feature selection separation level. This parameter informs how many cutpoints must distinguish each pair of observations. (Default = 0, i.e.,

no requirement is imposed. Therefore, there might exist observations belonging to different classes but having the same binarized representation).

- **solutionSize**: Number of cutpoints returned by the procedure. The best ranked cutpoints (with respect to frequency of participation in solutions to subproblems) are returned.

4.3.3 Rule Generation

The rule generation phase consists of building decision rules. LAD-WEKA's implementation of this phase consists of a randomized algorithm, referred to as *Random Rule Generator* (RRG). Figure 7 shows the configuration window of RRG.

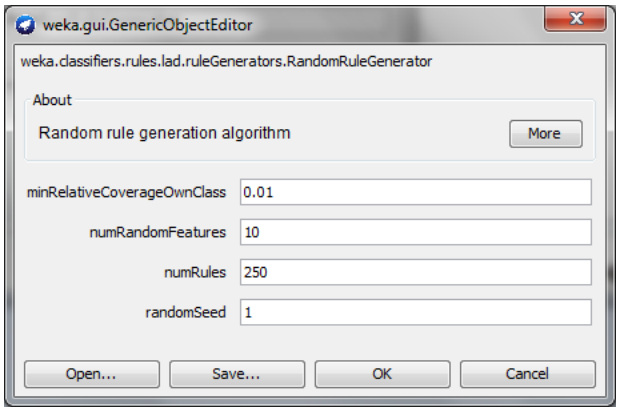

Figure 7: Random Rule Generator's configuration window.

Below we list the Random Rules Generators' parameters settings:

- **minimumPurity**: [seen on Figure 5] Minimum purity requirement for rules. This is an upper bound on the number of points from the opposite class that are covered by a rule of a given class (the value is given as a percentage of the total number of points covered by the rule).
- **minRelativeCoverageOwnClass**: Minimum Relative Coverage. How much coverage of its own class a rule must have in order to be accepted.
- **numRandomFeatures**: Number of Random Features. Number of features sampled at each iteration during rule construction.
- **numRules**: Number of Rules. Number of rules of each class that the algorithm attempts to generate.
- **randomSeed**: This value is used as seed to the pseudo-random number generator employed during the rule construction process.

4.4 Memory issues with LAD-WEKA

If the user is facing problems with not having enough memory to run experiments with LAD-WEKA, chances are that the data set in use is too large and, consequently, the LAD classifier built is consuming too much memory. In most cases, a simple solution is to re-run LAD-WEKA with an initialization parameter that requires the operating system to allow a larger portion of memory to

be used by LAD-WEKA. This can be accomplished in two ways, a discussed below.

First, the user can execute WEKA through command line and instruct the operating system to allocate more memory to the Java Virtual Machine (JVM) heap by using the `Xmx` token, followed by the desired amount of memory. Below we show how to execute LAD-WEKA's through the command line with 1024 Megabytes of memory:

```
java -Xmx1024m -classpath weka.jar weka.gui.GUIChooser
```

The second way of controlling how much memory will be allocated to LAD-WEKA is modifying the `RunWeka.ini` file. This file is found within WEKA's folder, as can be seen in Figure 2. By openning it with any plain text editor such as Notepad the user can modify the following line to reflect the desired amount of memory:

```
maxheap=1024m
```

Namely, the user can modify the line to read, for example: `maxheap=2048m`.

4.5 What to do if you found a bug

If you found a bug in LAD-WEKA, especially on the LAD classifier, please consider informing us via e-mail (please use the e-mails listed in the beginning of this document). Including the following information in your message will help tracking the bug:

1. Version of LAD-WEKA (e.g., 1.0 rev 112)
2. Java version (e.g., 1.6.0.24 64bit)

If the user is not sure about which Java version is running in their machine, they can easily look for this information at WEKA's GUI Chooser Help menu, or find it via the command line.

WEKA provides an easy way to identify the user's Java version along with many other information about the user's system. To access that information, one needs to open LAD-WEKA's GUI Chooser, go to the menu `Help -> SystemInfo` (or hit Ctrl + i) and check for the Java version entry in the table shown. From the command line, it is possible to obtain the Java version by typing

```
java -version
```

5 Acknowledgements

We gratefully acknowledge the partial financial support obtained from the Brazilian Council for Scientific and Technological Development (CNPq) and from the Federal University of the Semi-Arid (UFERSA).

We also appreciate your interest in using of LAD-WEKA and hope you can benefit from its features.