Table of Contents

Introduction

As of version 3.4.4 it is possible for WEKA to dynamically discover classes at runtime (rather than using only those specified in the `GenericObjectEditor.props` (GOE) file). In some versions (3.5.8, 3.6.0) this facility was not enabled by default as it is a bit slower than the GOE file approach, and, furthermore, does not function in environments that do not have a CLASSPATH (e.g., application servers). Later versions (3.6.1, 3.7.0) enabled the dynamic discovery again, as WEKA can now distinguish between being a standalone Java application or being run in a non-CLASSPATH environment.

If you wish to enable or disable dynamic class discovery, the relevant file to edit is `GenericPropertiesCreator.props` (GPC). You can obtain this file either from the `weka.jar` or `weka-src.jar` archive. Open one of these files with an archive manager that can handle ZIP files (for Windows users, you can use 7-Zip for this) and navigate to the `weka/gui` directory, where the GPC file is located. All that is required, is to change the `UseDynamic` property in this file from `false` to `true` (for enabling it) or the other way round (for disabling it). After changing the file, you just place it in your home directory. In order to find out the location of your home directory, do the following:

- Linux/Unix
  - Open a terminal
  - run the following command:
    `echo $HOME`

- Windows
  - Open a command-primpt
  - run the following command:
    `echo %USERPROFILE%`

If dynamic class discovery is too slow, e.g., due to an enormous CLASSPATH, you can generate a new `GenericObjectEditor.props` file and then turn dynamic class discovery off again. It is assumed that you already place the GPC file in your home directory (see steps above) and that the `weka.jar` jar archive with the WEKA classes is in your CLASSPATH (otherwise you have to add it to the `java` call using the `-classpath` option).

For generating the GOE file, execute the following steps:

- generate a new `GenericObjectEditor.props` file using the following command:
  - Linux/Unix

```
java weka.gui.GenericPropertiesCreator \
   $HOME/GenericPropertiesCreator.props \
   $HOME/GenericObjectEditor.props
```

  - Windows (command must be in one line)

```
java weka.gui.GenericPropertiesCreator
   %USERPROFILE%\GenericPropertiesCreator.props
   %USERPROFILE%\GenericObjectEditor.props
```

- edit the `GenericPropertiesCreator.props` file in your home directory and set `UseDynamic` to `false`.

A limitation of the GOE prior to 3.4.4 was, that additional classifiers, filters, etc., had to fit into the same package structure as the already existing ones, i.e., all had to be located below `weka`. WEKA can now display multiple class hierarchies in the GUI, which makes adding new functionality quite easy as we will see later in an example (it is not restricted to classifiers only, but also works with all the other entries in the GPC file).

File Structure

The structure of the GOE so far was a key-value-pair, separated by an *equals*-sign. The *value* is a comma separated list of classes that are all derived from the superclass/superinterface *key*. The GPC is slightly different, instead of declaring all the classes/interfaces one need only to specify all the packages descendants are located in (only non-abstract ones are then

listed). E.g., the `weka.classifiers.Classifier` entry in the GOE file looks like this:

```
weka.classifiers.Classifier=\
 weka.classifiers.bayes.AODE,\
 weka.classifiers.bayes.BayesNet,\
 weka.classifiers.bayes.ComplementNaiveBayes,\
 weka.classifiers.bayes.NaiveBayes,\
 weka.classifiers.bayes.NaiveBayesMultinomial,\
 weka.classifiers.bayes.NaiveBayesSimple,\
 weka.classifiers.bayes.NaiveBayesUpdateable,\
 weka.classifiers.functions.LeastMedSq,\
 weka.classifiers.functions.LinearRegression,\
 weka.classifiers.functions.Logistic,\
 ...
```

The entry producing the same output for the classifiers in the GPC looks like this (7 lines instead of over 70 in WEKA 3.4.4!):

```
weka.classifiers.Classifier=\
 weka.classifiers.bayes,\
 weka.classifiers.functions,\
 weka.classifiers.lazy,\
 weka.classifiers.meta,\
 weka.classifiers.trees,\
 weka.classifiers.rules
```

Exclusion

It may not always be desired to list all the classes that can be found along the CLASSPATH. Sometimes, classes cannot be declared `abstract` but still shouldn't be listed in the GOE. For that reason one can list classes, interfaces, superclasses for certain packages to be excluded from display. This exclusion is done with the following file:

```
weka/gui/GenericPropertiesCreator.excludes
```

The format of this properties file is fairly easy:

```
<key>=<prefix>:<class>[,<prefix>:<class>]
```

Where the `<key>` corresponds to a key in the `GenericPropertiesCreator.props` file and the `<prefix>` can be one of the following:

- **S** - *Superclass*
  any class class derived from this will be excluded
- **I** - *Interface*
  any class implementing this interface will be excluded
- **C** - *Class*
  exactly this class will be excluded

Here are a few examples:

```
 # exclude all ResultListeners that also implement the ResultProducer
interface
 # (all ResultProducers do that!)
 weka.experiment.ResultListener=\
   I:weka.experiment.ResultProducer

 # exclude J48 and all SingleClassifierEnhancers
 weka.classifiers.Classifier=\
   C:weka.classifiers.trees.J48,\
   S:weka.classifiers.SingleClassifierEnhancer
```

Class Discovery

Unlike the `Class.forName(String)` method that grabs the first class it can find in the CLASSPATH, and therefore fixes the location of the package it found the class in, the dynamic discovery examines the complete CLASSPATH you're starting the Java Virtual Machine (= JVM) with. This means that you can have several parallel directories with the same WEKA package structure, e.g. the standard release of WEKA in one directory (= `/distribution/weka.jar`) and another one with your own classes (= `/development/weka/...`), and display all of the classifiers in the GUI. In case of a name conflict, i.e. two directories contain the same class, the first one that can be found is used. In a nutshell, your `java` call of the GUIChooser could look like this:

```
 j
av
a -classpath "/development:/distribution/weka.jar" weka.gui.GUIChooser
```

Note: Windows users have to replace the ":" with ";" and the forward slashes with backslashes.

Multiple Class Hierarchies

In case you're developing your own framework, but still want to use your classifiers within WEKA that wasn't possible so far. With the release **3.4.4** it is possible to have multiple class hierarchies being displayed in the GUI. If you've developed a modified version of J48, let's call it *MyJ48* and it's located in the package `dummy.classifiers` then you'll have to add this package to the classifiers list in the GPC file like this:

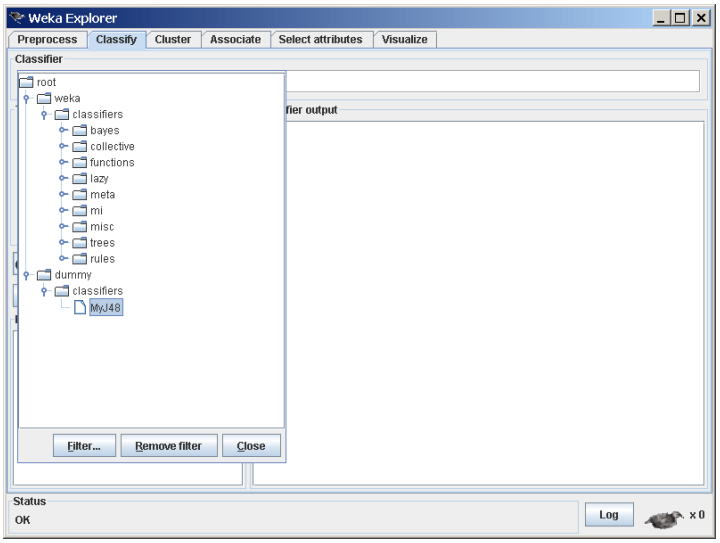
```
weka.classifiers.Classifier=\
 weka.classifiers.bayes,\
 weka.classifiers.functions,\
 weka.classifiers.lazy,\
 weka.classifiers.meta,\
 weka.classifiers.trees,\
 weka.classifiers.rules,\
 dummy.classifiers
```

Your `java` call for the GUIChooser might look like this:

```
java -classpath "weka.jar:dummy.jar" weka.gui.GUIChooser
```

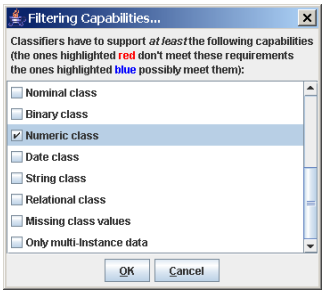
Note: Windows users have to replace the ":" with ";" and the forward slashes with backslashes.

Starting up the GUI you'll now have another root node in the tree view of the classifiers, called *root*, and below it the *weka* and the *dummy* package hierarchy as you can see here:

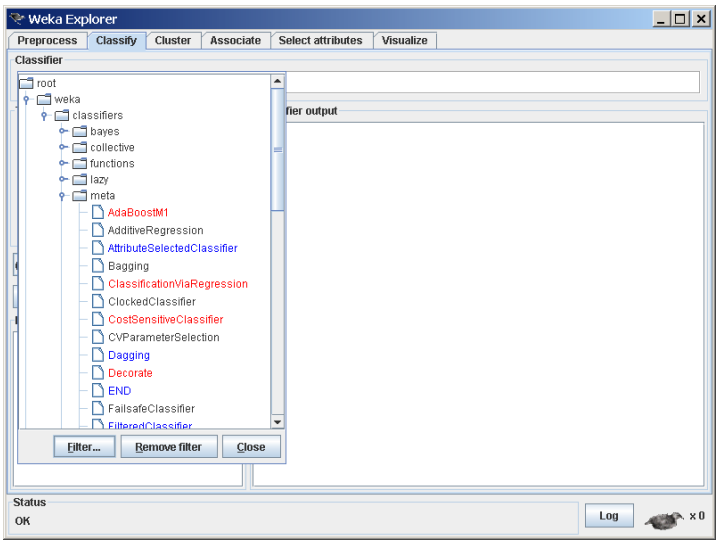


Capabilities

Version **3.5.3** of Weka introduces the notion of *Capabilities*. Capabilities basically list what kind of data a certain object can handle, e.g., one classifier can handle numeric classes, but another cannot. In case a class supports capabilities the additional buttons **Filter...** and **Remove filter** will be available in the GOE. The **Filter...** button pops up a dialog which lists all available Capabilities:



One can then choose those capabilities an object, e.g., a classifier, should have. If one is looking for classification problem, then the *Nominal class* Capability can be selected. On the other hand, if one needs a regression scheme, then the Capability *Numeric class* can be selected. This filtering mechanism makes the search for an appropriate learning scheme easier. After applying that filter, the tree with the objects will be displayed again and lists all objects that can handle all the selected Capabilities **black**, the ones that cannot **red** (starting with 3.5.8: **silver**) and the ones that might be able to handle them **blue** (e.g., meta classifiers which depend on their base classifier(s)).



Links

- GenericObjectEditor (book version)

- CLASSPATH
- Properties file
- GenericPropertiesCreator.props
- GenericPropertiesCreator.excludes