Intelligent Data Analysis Computer Laboratory
Practice on classification and clustering software

26th of January – 3rd of February 2012

Arnaud Quirin <arnaud.quirin@softcomputing.es>
http://aquirin.ovh.org/mastersoft/

Part I : The Weka Software (version 3.6.6)

Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes.

To access additional book and papers references, you will have to check the following website:

http://www.cs.waikato.ac.nz/ml/weka/

Typical use of Weka

The two main uses of Weka are:

- Classification of labelled data
- Clustering (unsupervised learning) of unlabelled data

These steps show the typical use of the Weka Software:

1. **Load the module "Explorer"**
2. **Open a data set.** The main format managed by Weka is the ARFF files (test files encoding the attributes names, the data values, etc), but other formats (CSV, XRFF, etc) are supported too.
    a. Go to Preprocess / Open File / Select an ARFF file
    b. ARFF files can be found in the data directory (for instance in Windows, in C:\Program Files\Weka-3-6\data)
    c. Additional data sets can be found on the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/)
3. Once loaded, **we can apply a filter** to remove unnecessary attributes, transform the data, etc.
    a. Select Preprocess / Choose / Filters / Unsupervised / Attribute / Remove to remove the irrelevant attributes. Sometimes, attributes such as "Instance number", "Datastore reference", etc have to be removed.

   b. Select Preprocess / Choose / Filters / Unsupervised / Attribute / Discretize to convert a real-valued attribute to a given set of integer-valued attributes.
   c. The button "Apply" has to be pressed to really filter the data.
4. Choose the Classify or the Cluster tab to **perform a classification or a clusterisation**.
   a. In each tab, use the "Choose" button to select an algorithm.
   b. Click on the grey bar with the name of the algorithm to modify its parameters.
   c. Choose an evaluation method (10-fold cross-validation, split the data, use of a provided test set, etc).
   d. The "Start" button launch the learning and print some statistics on the right panel (correctly/incorrectly classified instances rates, mean square error, confusion matrix for classification; cluster statistics for clustering)
   e. Models can be saved, loaded and visualized by right-clicking the name of the algorithm.
   f. **Note**: before trying a clusterisation algorithm, you will have to remove the class attribute of the dataset.
   g. In the following, the classifiers will be denoted using the class name. For instance, `weka.classifiers.trees.J48` refers to the C4.5 classifier available in Classify / Choose / Weka / Classifiers / Trees / J48.
   Note: read here why the name “J48” and not “C4.5”:
   https://list.scms.waikato.ac.nz/pipermail/wekalist/2009-February/015939.html

Assignments

- Download or copy from the installed folder, and load a dataset (any is suitable)
- Try at least two classification and two clustering algorithms you already know, and try to interpret the obtained results
- In classification mode, try to obtain a better correctly classified instance rate than the one obtained with the default parameters.
- **(Long, do this at the end of Part I)** Find a very huge dataset in Internet, cut it into a train (66%) and a test set (33%), save the test set as file apart, try to find a way to classify it properly. Use attribute selection, and perform different test with different classification algorithms.
- **(Long, do this at the end of Part I)** Find a dataset on Internet for which the number of instances for each class are completely different (an unbalanced dataset), and find a proper combination of filters and algorithms to classify it properly. Compare the results with those obtained without the selected filters.

Visualization

Access to **visualization** from the *Classifier*, *Cluster* and *Attribute Selection* panel is available from a popup menu. Click the right mouse button over an entry in the Result list to bring up the menu. You will be presented with options for viewing or saving the text output and - depending on the scheme - further options for visualizing errors, clusters, trees etc.

Assignments

- Load the "segment-challenge.arff" data

- In classification mode, train a C4.5 decision tree (trees / J48)
- Visualize the classifier errors and the learned tree
- Parameter the C4.5 algorithm in order to have a better compromise between accuracy (a low mean square error) and interpretability (a tree having a small number of nodes, less than 10-15 to be human-readable)
- In clusterisation mode, run a SimpleKMeans with the default parameters (do not forget to remove or ignore the class attribute)
- Visualize the cluster assignments
- Determine the couple of attributes in the 2D-visualisation which best discriminate the points of the two clusters (represented in blue or red)
- Check in the "Cluster Output" window what are the N-dimensional coordinates of the centroids of the two clusters. Is it possible to determine the couple of discriminative attributes using these coordinates?

Tune the parameters of a classifier

Since finding the optimal parameters for a classifier can be a rather tedious process, Weka offers some ways of automating this process a bit. The following meta-classifiers allow you to optimize some parameters of your base classifier:

- `weka.classifiers.meta.CVParameterSelection`
- `weka.classifiers.meta.GridSearch`

After finding the best possible setup, the meta-classifiers then train an instance of the base classifier with these parameters and use it for subsequent predictions.

CVParameterSelection

This meta-classifier can optimize over an arbitrary number of parameters, with only one drawback (apart from the obvious explosion of possible parameter combinations): one cannot optimize on nested options, only direct options of the base classifier. What does that mean? It means, that you can optimize the *C* parameter of `weka.classifiers.functions.SMO`, but not the *C* of an `weka.classifiers.functions.SMO` within a `weka.classifiers.meta.FilteredClassifier`.

Assignments

1. Load the dataset of your choice in the Explorer
2. Choose `weka.classifiers.meta.CVParameterSelection` as classifier
3. Open its parameters and select `weka.classifiers.trees.J48` as base classifier
4. As a example, we will optimize the confidence factor (parameter "-C"). Note that the short parameter name is indicated in front of the button "Choose", in the CVParameterSelection parameter window.
5. Open the ArrayEditor for *CVParameters* and enter the following string (and click on *Add*): "`C 0.1 0.5 5`". This will test the confidence parameter from 0.1 to 0.5 with step size 0.1 (= 5 steps).

6. Confirm all the choices and start the classifier. If you get an error "Cannot handle numeric class", that means that you need a nominal class in your dataset.
7. You will get and output similar to this one, with the selected parameter indicated on the end of the line:

```
Scheme: weka.classifiers.meta.CVParameterSelection -P "C 0.1 0.5 5.0" -X 10 -S 1 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2
```

   This means that the metaclassifier found "0.25" as the best value for parameter C.
8. Optimize the classifier "`weka.classifiers.functions.SMO`" and its complexity parameter (-C) with the RBF kernel.
9. Explore other classifiers, and optimize more than one parameter.

GridSearch

`weka.classifiers.meta.GridSearch` is a meta-classifier for exploring 2 parameters, hence the *grid* in the name. If one turns the log on, the classifier will create output suitable for gnuplot, i.e., sections of the log will contain script and data sections. Instead of just using a classifier, one can specify a base classifier **and** a filter, which both of them can be optimized (one parameter each). In contrast to `CVParameterSelection`, `GridSearch` is not limited to first-level parameters of the base classifier, since it's using Java Beans Introspection and one can specify *paths* to the properties one wants to optimize. A *property* here is the string of the parameter displayed in the GenericObjectEditor (generated though Introspection), e.g., `bagSizePercent` or `classifier` of `weka.classifiers.meta.Bagging`.

For each of the two axes, X and Y, one can specify the following parameters:

- **property**
  The dot-separated path pointing to the property to be optimized. In order to distinguish between paths for the filter or the classifier, one needs to prefix the path either with `filter.` or `classifier.` for filter or classifier path respectively.
- **expression**
  The mathematical expression to generate the value for the property, processed with the `weka.core.MathematicalExpression` class, which supports the following functions: `abs`, `sqrt`, `log`, `exp`, `sin`, `cos`, `tan`, `rint`, `floor`, `pow`, `ceil`. These variables are available in the expression: `BASE`, `FROM`, `TO`, `STEP`, `I`; with `I` ranging from `FROM` to `TO`. For instance, "`pow(BASE,I)`" can be used to optimize the expression BASE^I, where BASE is defined by the user.
- **min**
  The minimum value to start from.
- **max**
  The maximum value.
- **step**
  The step size used to get from *min* to *max*.
- **base**
  Used in `pow()` calculations.

`GridSearch` can also optimized based on the following measures:

- Correlation coefficient (= CC)
- Root mean squared error (= RMSE)
- Root relative squared error (= RRSE)
- Mean absolute error (= MAE)
- Root absolute error (= RAE)
- Combined: (1-abs(CC)) + RRSE + RAE
- Accuracy (= ACC)

Note: *Correlation coefficient* is only available for numeric classes and *Accuracy* only for nominal ones.

Assignments

1. Load the dataset of your choice in the Explorer (having nominal classes!)
2. Choose `weka.classifiers.meta.GridSearch` as classifier
3. In its options, set the evaluation to *Accuracy*.
4. Set the filter to `weka.filters.AllFilter` since we don't need any special data processing and we don't optimize the filter in this case (data gets always passed through filter!).
5. We will optimize the C and the gamma parameter of SMO (Support Vector Machine) with RBFKernel. Set `weka.classifiers.functions.SMO` as classifier with `weka.classifiers.functions.supportVector.RBFKernel` as kernel.
6. Set the `XProperty` to "classifier.c", `XMin` to "1", `XMax` to "16", `XStep` to "1" and the `XExpression` to "I". This will test the "C" parameter of `SMO` for the values from 1 to 16.
7. Set the `YProperty` to "classifier.kernel.gamma", `YMin` to "-5", `YMax` to "2", `YStep` to "1", `YBase` to "10" and `YExpression` to "pow(BASE,I)". This will test the gamma of the RBFKernel with the values 10^-5, 10^-4,..,10^2.
8. You will get and output similar to this one, with the selected parameter indicated on the last line (this took me 22 minutes on my 2.40GHz CPU):

```
weka.classifiers.meta.GridSearch:
Filter: weka.filters.AllFilter
Classifier: weka.classifiers.functions.SMO -C 7.0 -L 0.0010 -P
1.0E-12 -N 0 -V -1 -W 1 -K
"weka.classifiers.functions.supportVector.RBFKernel -C 250007 -
G 1.0"

X property: classifier.c
Y property: classifier.kernel.gamma

Evaluation: Accuracy
Coordinates: [7.0, 0.0]
Values: 7.0 (X coordinate), 1.0 (Y coordinate)
```

9. Use the GridSearch method to optimize a PLSFilter with LinearRegression, and the parameters "number of components" (from 5 to 20) and "ridge" (from 10^-10 to 10^5). If the experiment is too long, check the parameters of the filter and the algorithm to speed up it a bit.
10. Explore and optimize other classifiers.

Generate learning curves

The *Advanced mode* of the Experimenter can be used to generated learning curves for classifiers. These approaches can be setup in the *Simple mode* as well, but it is more cumbersome than in the advanced mode.

Number of instances

For varying the number of instances a classifier is trained on, we use the `FilteredClassifier` classifier (package `weka.classifiers.meta`) in conjunction with the `RemovePercentage` filter (package `weka.filters.unsupervised.instance`) and J48 as base classifier (package `weka.classifiers.trees`):

Assignments

1. Start the Experimenter (class `weka.gui.experiment.Experimenter`)
2. Select the configuration mode *Advanced* in the *Setup* panel
3. Choose as *Destination* either an ARFF file (= `InstancesResultListener`) or a database (= `DatabaseResultListener`) and configure the listener to your needs
4. Choose as *Result generator* the `CrossValidationResultProducer` (or leave the `RandomSplitResultProducer`)
5. Open the options dialog of the `CrossValidationResultProducer` by left-clicking on the edit field
6. In case of regression datasets, choose the `RegressionSplitEvaluator` instead of the `ClassifierSplitEvaluator` (the latter is used for classification problems)
7. Open the options dialog for the *splitEvaluator* by left-clicking on the edit field
8. Choose the `weka.classifier.meta.FilteredClassifier` and open its parameters
9. *Choose* the classifier that you want to analyze and setup its parameters, in our case this is `J48` as base classifier and `RemovePercentage` as filter
10. Close all dialogs again (accepting them with OK)
11. Set the *Generator properties* to *enabled*
12. Choose as property *percentage* and click on *Select*:

```
splitEvaluator -> classifier -> filter -> percentage
```

13. Now you can add all the percentages that you want to test, e.g. (NB: this is the percentage being *removed*!):

```
90, 80, 70, 60, 50, 40, 30, 20, 10
```

14. Add the datasets you want to generate the learning curve for.
15. Save the experiment.
16. Go to the *Run* panel and start the experiment.
17. After the experiment has finished, select the *Analyse* panel and perform your analysis on the results. For this you can save the results in CSV/Gnuplot/HTML/Plain text and edit them with an external program.

18. Do the same experiment setup, but to generate a learning curve that vary on a specific parameter (take the confidence factor of J48, from 0.1 to 0.5).

Classify text data

Import

Weka needs the data to be present in ARFF or XRFF format in order to perform any classification tasks.

Directories

Data can first be uploaded as a set of files and folders. Study the example "text_example".

Example directory layout for **TextDirectoryLoader**:

```
 ...
 |
 +- text_example
    |
    +- class1
    |  |
    |  + file1.txt
    |  |
    |  + file2.txt
    |  |
    |  ...
    |
    +- class2
    |  |
    |  + another_file1.txt
    |  |
    |  + another_file2.txt
    |  |
    |  ...
```

One can transform the text files with the following tool into ARFF format:

TextDirectoryLoader converter

This converter is based on the *TextDirectoryToArff* tool and located in the `weka.core.converters` package

The above directory structure can be turned into an ARFF file like this:

- Open the command line (Menu Start / Run / cmd)
- Go to the directory parent of the text data: `cd <folder\text_example\..>`
- Type:

```
java -classpath "C:\Program Files\Weka-3-6\weka.jar"
weka.core.converters.TextDirectoryLoader -dir text_example >
text_example.arff
```

"`text_example.arff`" is now a typical Weka file which can be open directly by the Explorer.

Conversion

Most classifiers in Weka cannot handle *String* attributes. For these learning schemes one has to process the data with appropriate filters, e.g., the StringToWordVector filter which can perform TF/IDF transformation. The `StringToWordVector` filter places the class attribute of the generated output data at the beginning.

It is available in the menu Filters
(`weka.filters.unsupervised.attribute.StringToWordVector`).

In case you'd to like to have it as last attribute again, you can use the Reorder filter with the following setup:

```
weka.filters.unsupervised.attribute.Reorder -R 2-last,first
```

Note that now, all the words are classified in one of the 3 classes: class1, class2 or class3, and that the number of instances of each class depend on the number of HTML files you had in the original folder structure.

And with the MultiFilter you can also apply both filters in one go, instead of subsequently. Makes it easier in the Explorer for instance.

UTF-8

In case you are working with text files containing non-ASCII characters, e.g., arabic, you might encounter some display problems under Windows. Java was designed to display UTF-8, which should include arabic characters. By default, Java uses code page 1252 under Windows, which garbles the display of other characters. In order to fix this, you will have to modify the java command-line with which you start up Weka:

```
java -Dfile.encoding=utf-8 -classpath ...
```

The **`-Dfile.encoding=utf-8`** tells Java to explicitly use UTF-8 encoding instead of the default CP1252.

If you are starting Weka via start menu and you use a recent version, then you will just have to modify the `fileEncoding` placeholder in the `RunWeka.ini` accordingly.

Assignments

- Load and convert the provided example: "text_example"
- Try several tree-based classification algorithms, and find which classify the best the data
- Try a personalized set of Internet webpages, as well as other classification algorithms.