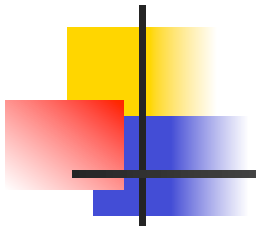
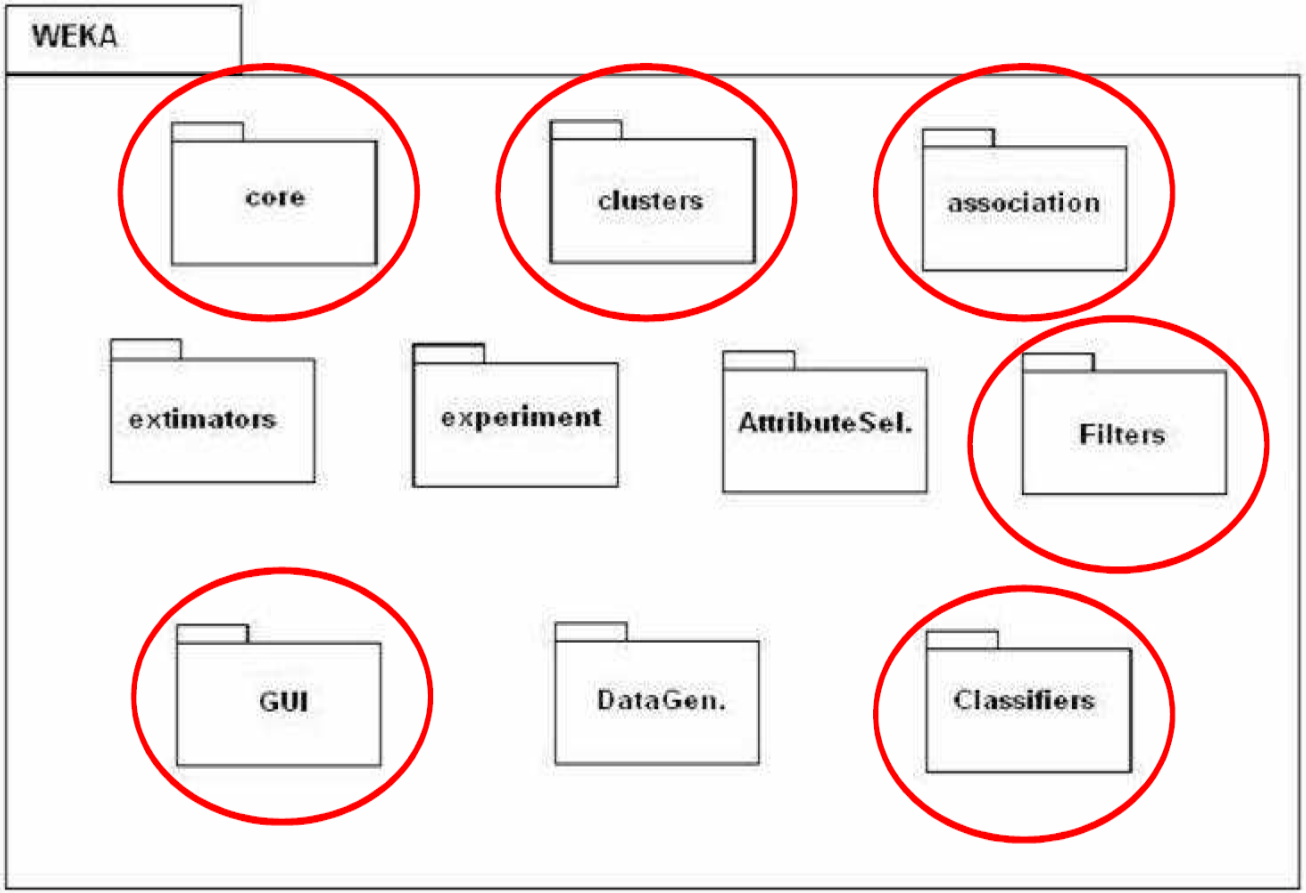


Working with WEKA Java Code I

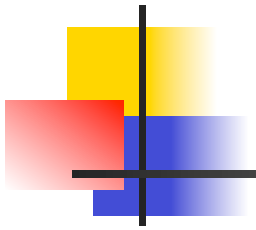
Prof. Pietro Ducange

WEKA Architecture (weka-3-8)



http://weka.sourceforge.net/doc.stable-3-8/

The structure of WEKA

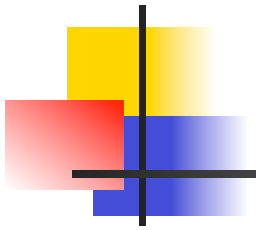


- Each learning algorithm is encapsulated in a class
- A collection of related class organized in a directory is a Package (es: the tree package contains the classes that implement decision trees)
- Packages are organized in a hierarchy: tree is a subpackage of the classifier package which is itself a subpackage of the overall weka package

Setup Eclipse IDE (i)

- Unpack weka-src.jar (jar xf weka-src.jar)
- Create a new Java application Project (File->New-> Java Project) and assign a name
- Import the code that you can find in weka-src\src\main \java\ into the src fold of your developing environment
- Right click on the src fold and select refresh (F5)

Setup Eclipse IDE (ii)



- Add to the library the .jar files that you can find in \weka-src\lib (select the project and then file->properties->Java Build Path->Libraries->Add External JARs)

- Try to build the project and run the GUI chooser (right click on the class weka.gui.GUIChooser.java and select Run As-> Java Application)

The weka.core package

- The core package is central to the WEKA system, its class are accessed from almost every other class.

- The key classes are
  - Attribute: it contains the attribute's name, its type and in the case of a nominal attribute, its possible values.
  - Instance: it contains the attribute values of a particular instance
  - Instances: it holds an ordered set of instances (i.e., a dataset)

Creating a dataset

```
Attribute num1 = new Attribute("num1");
Attribute num2 = new Attribute("num2");
ArrayList<String> labels = new ArrayList<String>();
labels.addElement("no");
labels.addElement("yes");
Attribute cls = new Attribute("class", labels);
ArrayList<Attribute> attributes = new ArrayList<Attribute>();
attributes.add       (num1);
attributes.add       (num2);
attributes.add       (cls);
Instances dataset = new Instances("Test-dataset", attributes, 0);
```

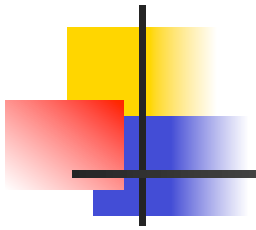
- The final argument in the Instances constructor above tells WEKA how much memory to reserve for upcoming weka.core.Instance objects

- If one knows how many rows will be added to the dataset, then it should be specified as it saves costly operations for expanding the internal storage

Adding data

```
double[] values = new double[data.numAttributes()];
values[0] = 1.23;
values[1] = data.attribute(1).parseDate("2001-11-09");
values[2] = data.attribute(2).indexOf("label_b");
values[3] = data.attribute(3).addStringValue("This is a string");
Instance inst = new DenseInstance(1.0, values);
data.add(inst);
```

- WEKA's internal format is using doubles for all attribute types. For nominal, string and relational attributes this is just an index of the stored values



Exercise I

Write a Java program, which performs the following steps:

- Generates a random dataset composed by numerical attributes
- Prints the generated dataset
- Saves the dataset in arff format by using the ArffSaver class methods (weka.core.converters package).

The number of instances and of attributes must be specified as parameters

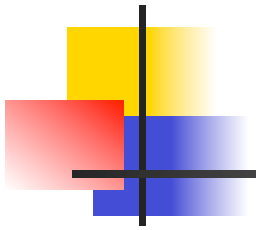
A solution can be found in the file creaDataset.java

An Example of Filter (I)

```
import weka.core.Instances;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.Remove;
...
String[] options = new String[2];
options[0] = "-R";                    // "range"
options[1] = "1";                     // first attribute
Remove remove = new Remove();         // new instance of filter
remove.setOptions(options);           // set options
remove.setInputFormat(data);          // inform filter about dataset
                                      // **AFTER** setting options
Instances newData = Filter.useFilter(data, remove); // apply filter
```

This method is used to determine the output format of the data

Static method for building the new Instances objects



An Example of Filter (II)

```
Instances train = ...   // from somewhere
Instances test = ...    // from somewhere
Standardize filter = new Standardize();
// initializing the filter once with training set
filter.setInputFormat(train);
// configures the Filter based on train instances and returns
// filtered instances
Instances newTrain = Filter.useFilter(train, filter);
// create new test set
Instances newTest = Filter.useFilter(test, filter);
```

Exercise II

Write a Java program, which performs the following steps:

- Reads a training and a test set from two specified file paths
- Defines the *Instances* objects for the datasets and set the class index (use the *setClassIndex* method of the class *Instances*)
- Prints the training and the test set
- Defines a supervised filter for discretizing both the training and the test sets
- Prints the discretized training and test set

A solution can be found in the file Discretizza.java