JCLEC Meets WEKA!

A. Cano, J. M. Luna, J. L. Olmo, and S. Ventura

Dept. of Computer Science and Numerical Analysis,
University of Cordoba, Rabanales Campus, Albert Einstein building,
14071 Cordoba, Spain. Tel.:+34957212218. Fax:+34957218630
{i52caroa,i32luarj,juanluisolmo,sventura}@uco.es

Abstract. WEKA has recently become a very referenced DM tool. In spite of all the functionality it provides, it does not include any framework for the development of evolutionary algorithms. An evolutionary computation framework is JCLEC, which has been successfully employed for developing several EAs. The combination of both may lead in a mutual benefit. Thus, this paper proposes an intermediate layer to connect WEKA with JCLEC. It also presents a study case which samples the process of including a JCLEC's EA into WEKA.

Keywords: WEKA, JCLEC, Evolutionary Algorithms, Data Mining

1 Introduction

The huge amount of data in many society fields has attracted the need for obtaining useful information and knowledge from such data. Many application domains as marketing, sales, medical diagnosis or education, where it is essential to analyze many variables, have arised as typical domains to apply data mining (DM) techniques [8]. The main objective of such techniques is to support expert domains' decisions, especially border line decisions.

Likewise, tools for applying these techniques should allow expert domains to interact and visualize the DM process. Nowadays, there are a wide variety of DM tools as KEEL [1], WEKA [7], Knime [2], RapidMiner [10], etc., which have enough functionality to be used in a broad range of problems. WEKA has become one of the most popular DM workbenchs and its success in researcher and education communities is due to its constant improvement, development, and portability. In fact, it can be easily extended with new algorithms. This tool, developed in the Java programming language, comprises a collection of algorithms for tackling several DM tasks as data pre-processing, classification, regression, clustering, association rules, also visualizing all the DM process. However, these algorithms are hardly used in situations where there are a huge amount of instances and attributes, situations where the execution becomes computationally hard. Instead, evolutionary algorithms (EAs) are very useful in this kind of circumstances, because they are powerful for solving problems that require a considerable execution time. That is why it should be very interesting to harness the power of EAs also in the DM field, and WEKA appears to be a good platform to consider the inclusion of EAs.

Several customized EA classifiers have been previously integrated in WEKA, but there is no algorithm following an evolutionary schema in the standard WEKA distribution. Moreover, there is no evolutionary computation framework that has been included into this platform yet. An Open Source efficient and generic framework for evolutionary computation is JCLEC [12] (Java Class Library for Evolutionary Computation), which provides a high-level software environment to apply any kind of EA, with support for genetic algorithms (binary, integer and real encoding), genetic programming (Koza style, strongly typed, and grammar based) and evolutionary programming. This framework has been used in different problems, obtaining good results [3, 9]. If we focus in the opposite direction, WEKA tool can help JCLEC by providing different data pre-processing algorithms to be applied before the EA. Because all of this, it seems very interesting to join the capabilities of WEKA with JCLEC framework, in order to allow the development of any kind of EA by using WEKA.

In this work, an intermediate layer that connects both WEKA and JCLEC is presented, showing how to include any evolutionary learning algorithm coded in JCLEC into WEKA. This enables the possibility of running this kind of algorithms in this well-known software tool as well as it provides JCLEC additional features and a graphical user interface.

This paper is structured as follows: Section 2 presents the architectonic design; Section 3 explains how to include a JCLEC EA in WEKA, illustrating this by means of an example; finally, some concluding remarks and future work are adumbrated.

2 Architectonic design

WEKA's design allows to include new algorithms easily. Any new class is picked up by the graphical user interface without additional coding needed to deploy it in WEKA. To do so, new algorithms should inherit some properties from certain classes, which also indicate the methods that should be implemented. Though we focus on classification and association tasks, a similar approach could be followed for any other DM task such as clustering. New classification and association algorithms should extend the `AbstractClassifier` and `AbstractAssociator` classes, respectively. An abstract class called `ClassificationAlgorithm` that collects the common properties of classification EAs has been developed, which extends from `AbstractAlgorithm`. Thus, any classification EA will inherit from `ClassificationAlgorithm` the common properties and methods, and it will just specify its particular properties. Similarly, association algorithms also inherit from an abstract class called `AssociationAlgorithm`.

The architectonical design, developed to include JCLEC in WEKA, follows the schema represented in Figure 1, where it is depicted that WEKA provides two abstract classes from which any association or classification algorithm should inherit, `AbstractAssociator` and `AbstractClassifier`. The abstract class `ClassificationAlgorithm` extends `AbstractClassifier`, and it is in charge of defining the properties and methods that any classification

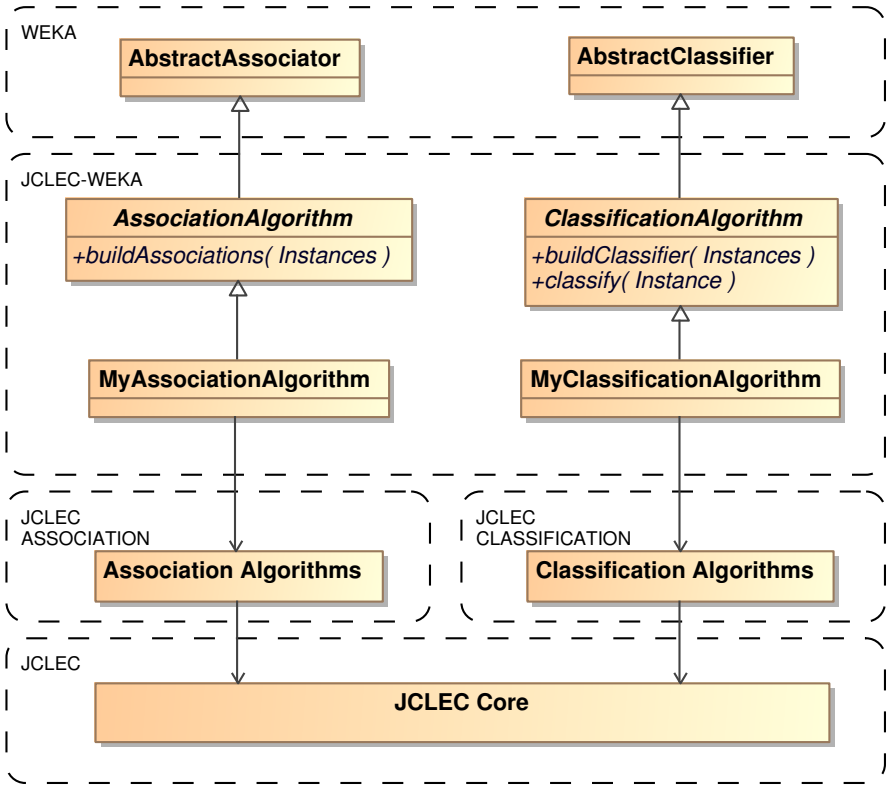

Fig. 1. Architectonical design.

EA shares, e.g., population size, number of generations, crossover and mutation operators, etc. Similarly, regarding association task, the abstract class `AssociationAlgorithm` extends `AbstractAssociator`. Finally, any EA extends from one of these classes and calls the corresponding execution method of JCLEC. This way, the JCLEC algorithm is executed in WEKA tool.

Next, the methods of the intermediate layer that have to be taken into account when including any classification or association algorithm are described. In case of classification, the following two particular methods should be implemented:

- `void buildClassifier(Instances data)`. This method generates a classifier from a training set.
- `double classifyInstance(Instances instance)`. This method classifies a new instance using the classifier learned previously.

On the other hand, for association algorithms, only one particular method should be implemented:

– `void buildAssociations(Instances data)`. This method generates a set of association rules from a dataset.

Independently of whether a classification or an association algorithm is going to be included, it should implement the following methods:

– `Capabilities getCapabilities()`. This method determines if the algorithm is compatible with the type of each attribute in the dataset (e.g., numeric, nominal, etc.).
– `String globalInfo()`. Returns information about the algorithm, which will appear when selecting the *About* option in the graphical user interface.
– `TechnicalInformation getTechnicalInformation()`. Shows information about the author, year of the publication, etc.
– `void setOptions(String[] options)`. This method establishes the parameters of the algorithm, e.g., -P 50 -G 100—the former indicates the population size and the latter the number of generations.
– `String [] getOptions()`. Returns the set of parameters previously established.
– `Setters` and `getters` methods that set and get the parameter values.
– `String toString()`. This method shows the results obtained in the graphical user interface.

An additional and very useful tool contained in WEKA is the package manager. This tool allows the inclusion of any external library or code necessary to run new algorithms or features in WEKA, incorporating the files into the properly structure so that it avoids the developer to modify WEKA's source code. The directory structure of any new package has to fulfill a fixed anatomy. Hence, the JCLEC-WEKA intermediate layer is necessary in order to be able to instantiate the execution code of the algorithm, as shown in Figure 1. This way, a connection between WEKA and JCLEC is established.

3 Case study

This section presents a sample case study that shows the functionality and process of a given classification algorithm. The considered algorihtm was presented in [11] by *Tan et al.*. This algorithm has been developed in JCLEC and it is based in a grammar guided genetic programming approach.

In order to show the advantages of incorporating JCLEC into WEKA tool, the classification algorithm presented by *Tan et al.*, has been added to a sample package[1] that can be easily included into WEKA. This package is the intermediate layer, which connects the WEKA interface with the JCLEC algorithm.

In order to create this package, a new class with the name of the algorithm is created, which extends the `ClassificationAlgorithm` class explained in Section 2. In the `buildClassifier()` method, the algorithm is instantiated and

[1] This new package is also available in JCLEC site, `http://jclec.sourceforge.net`

the configuration parameters that are necessary for its execution are established. Then, the execution method of the algorithm is called and the classifier model is inferred using the training set. These steps make up the `buildClassifier()` method, as depicted in the following code:

```
public void buildClassifier(Instances instances)
{
    algorithm = new TanAlgorithm();
    configureMetadata(instances);
    algorithm.execute();
}
```

The `classifyInstance()` method receives a WEKA instance. It is necessary to turn this instance into a JCLEC instance. Then it is applied to the classifier, which returns the class predicted for this instance.

```
public double classifyInstance(Instance ins)
{
    ArffDataSet.Instance instance = dataset.new Instance();
    instance.setValues(ins.toDoubleArray());
    return algorithm.getClassifier().classify(instance);
}
```

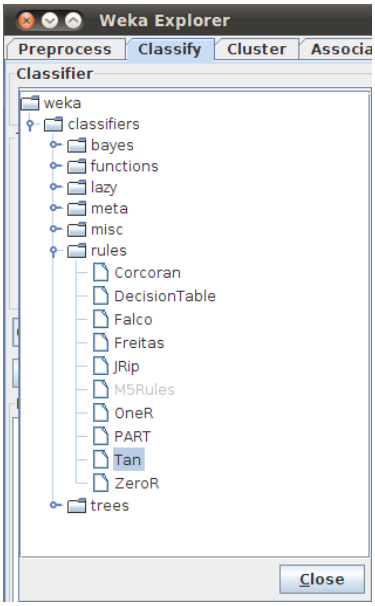

Fig. 2. WEKA's rules classifiers with EAs.

The next step is to build the package using the Ant file provided by WEKA, which generates a zip file. This file has to be imported using the package manager. Once the package is charged, the classification algorithm appears beside the other classification algorithms available in WEKA, as illustrated in Figure 2. Notice that other EAs have been included along with *Tan et al.*—*Corcoran and Sen* [4], *Falco et al.* [5] and *Freitas et al.* [6].

The next step is to specify the parameters needed to run the algorithm. Like EAs, this algorithm has a series of parameters such as population size, number of generations, crossover and mutation probability, seed, etc. All these parameters should be specified in a dialog box, as shown in Figure 3.

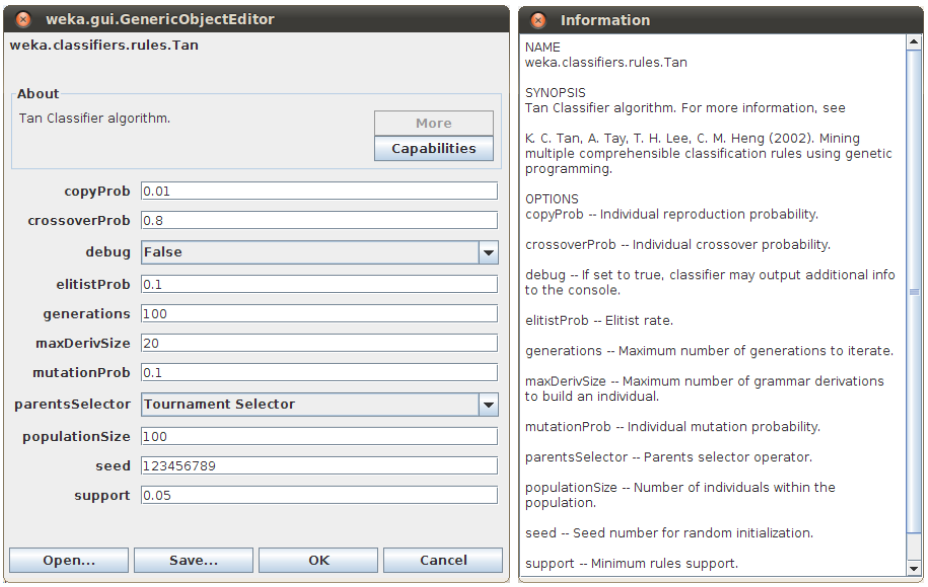

Fig. 3. Parameters configuration dialog box.

Finally, once the algorithm execution is carried out, the classifier obtained, computed metrics and the confusion matrix are displayed. Figure 4 shows the results of running the *Tan et al.* algorithm over the Iris dataset. This dataset along with other well-known datasets are available in the WEKA data folder.

4 Concluding remarks and future work

This paper presents an intermediate layer that provides the possibility of connecting an evolutionary computing framework as JCLEC with WEKA. This synergy makes easier the final user to harness the power of EAs in the DM field, by using on the one hand WEKA's graphical user interface and pre-processing

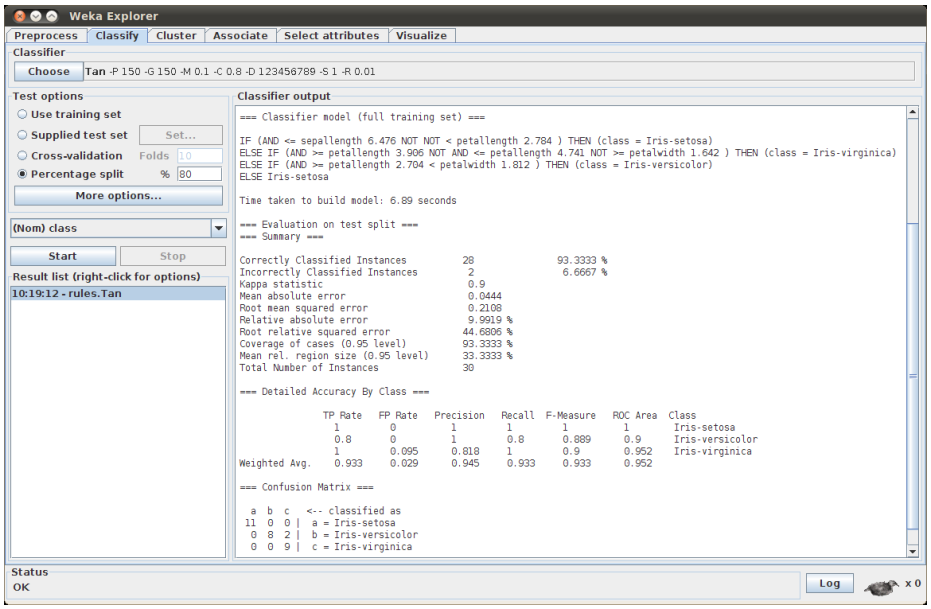

Fig. 4. Results of running *Tan et al.* EA under WEKA

tools, and on the other hand the power of EAs when applying them specifically to computationally expensive problems.

This work also opens up the chance of adding new features to WEKA. For instance, JCLEC has recently incorporated a new module that allows the execution of any genetic programming algorithm using graphics processing units [3]. Thus, the intermediate layer developed also offers WEKA the speed up and parallel processing capabilities inherent to graphics processing units architecture. In addition, this connection between JCLEC and WEKA permits not only the implementation of classification and association algorithms, but the implementation of EAs devoted to other DM tasks —multi-instance learning, multi-label classification, feature selection, etc.—also following binary, integer, real, expression tree and syntax tree codification schemes.

Acknowledgments.

This work has been supported by the Regional Goverment of Andalucia and the Ministry of Science and Technology projects, P08-TIC-3720 and TIN2008-06681-C06-03 respectively, and FEDER funds.

References

1. Alcalá-Fdez, J., Sánchez, L., García, S., del Jesus, M.J., Ventura, S., Garrell, J.M., Otero, J., Romero, C., Bacardit, J., Rivas, V.M., Fernández, J.C., Herrera, F.: KEEL: a software tool to assess evolutionary algorithms for data mining problems. Soft Computing 13, 307–318 (2008)
2. Berthold, M.R., Cebron, N., Dill, F., Gabriel, T.R., Kötter, T., Meinl, T., Ohl, P., Sieb, C., Thiel, K., Wiswedel, B.: KNIME: The Konstanz Information Miner. In: Data Analysis, Machine Learning and Applications, chap. 38, pp. 319–326. Springer Berlin Heidelberg (2008)
3. Cano, A., Zafra, A., Ventura, S.: Solving classification problems using genetic programming algorithms on GPUs. In: Hybrid Artificial Intelligence Systems. Lecture Notes in Computer Science, vol. 6077, pp. 17–26. Springer (2010)
4. Corcoran, A.L., Sen, S.: Using real-valued genetic algorithms to evolve rule sets for classification. In: Proceedings of 1st IEEE Conference on Evolutionary Computation. pp. 120–124 (1994)
5. De Falco, I., Della Cioppa, A., Tarantino, E.: Discovering interesting classification rules with genetic programming. Applied Soft Computing 1(4), 257–269 (2001)
6. Freitas, A.A.: Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2002)
7. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. SIGKDD 11, 10–18 (2009)
8. Han, M.S., Yu, J., Chen, P.S.: Data mining: An overview from a database perspective. IEEE Transactions on Knowledge and Data Engineering 8(6), 866–883 (1996)
9. J.M. Luna, J.R., Ventura, S.: Analysis of the effectiveness of G3PARM algorithm. In: Hybrid Artificial Intelligence Systems. Lecture Notes in Computer Science, vol. 6077, pp. 27–34 (2010)
10. Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., Euler, T.: Yale: Rapid prototyping for complex data mining tasks. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 935–940. ACM, New York, NY, USA (2006)
11. Tan, K.C., Tay, A., Lee, T.H., Heng, C.M.: Mining multiple comprehensible classification rules using genetic programming. In: Proceedings of the Evolutionary Computation CEC '02. pp. 1302–1307. IEEE Computer Society, Washington, DC, USA (2002)
12. Ventura, S., Romero, C., Zafra, A., Delgado, J.A., Hervás, C.: JCLEC: a Java framework for evolutionary computation. Soft Computing 12, 381–392 (2007)