Implementation and Integration of Algorithms into the KEEL Data-Mining Software Tool

Alberto Fernández*, Julián Luengo, Joaquin Derrac, Jesús Alcalá-Fdez, and Francisco Herrera

Dept. of Computer Science and A.I., University of Granada
Tel:+34-958-240598; Fax: +34-958-243317
{alberto,julianlm,jderrac,jalcala,herrera}@decsai.ugr.es

Abstract. This work is related to the KEEL[1] (Knowledge Extraction based on Evolutionary Learning) tool, a non-commercial software that supports data management, design of experiments and an educational section. The KEEL software tool is devoted to assess evolutionary algorithms for Data Mining problems including regression, classification, clustering, pattern mining and so on. These features implies an advantage for the research and educational field.

The aim of this contribution is to present some guidelines for including new algorithms in KEEL, helping the researchers to make their methods easily accessible for other authors and to compare the results of many approaches already included within the KEEL software. By providing a source code template, the developer does not need to take into account the basic requirements of the KEEL software tool, and he or she has only to focus in the designing and encoding of his or hers approach.

Keywords: Data Mining, Evolutionary Algorithms, Java, Knowledge Extraction, Machine Learning.

1 Introduction

Data Mining (DM) is the process for automatic discovery of high level knowledge by obtaining information from real world, large and complex data sets [1]. This idea of automatically discovering knowledge from databases is a very attractive and challenging task, both for academia and industry.

The use of DM in problem solving is a widespread practice [2,3,4]. However, their use requires a certain programming expertise along with considerable time and effort to write a computer program for implementing the often sophisticated algorithm according to user needs. This work can be tedious and needs to be done before users can start focusing their attention on the issues that they should be really working on. In the last few years, many software tools have been developed to reduce this task. Although a lot of them are commercially distributed (some of the leading commercial software are mining suites such as SPSS Clementine[2],

* Corresponding author.
[1] http://www.keel.es
[2] http://www.spss.com/clementine

H. Yin and E. Corchado (Eds.): IDEAL 2009, LNCS 5788, pp. 562–569, 2009.

Oracle Data Mining[3] and KnowledgeSTUDIO[4]), a few are available as open source software such as Weka [5] or Java-ML [6] (we recommend visiting the KDnuggets software directory[5] and The-Data-Mine site[6]). Open source tools can play an important role as is pointed out in [7].

KEEL (Knowledge Extraction based on Evolutionary Learning) [8] is a non-commercial Java software tool which empowers the user to assess the behaviour of evolutionary learning for different kinds of DM problems: regression, classification, clustering, pattern mining and so on. This tool can offer several advantages:

- It reduces programming work. It includes a library with evolutionary learning algorithms based on different paradigms (Pittsburgh, Michigan and IRL) and simplifies the integration of evolutionary learning algorithms with different pre-processing techniques. It can alleviate researchers from the mere "technical work" of programming and enable them to focus more on the analysis of their new learning models in comparison with the existing ones.
- It extends the range of possible users applying evolutionary learning algorithms. An extensive library of Evolutionary Algorithms (EAs) together with easy-to-use software considerably reduce the level of knowledge and experience required by researchers in evolutionary computation. As a result researchers with less knowledge, when using this tool, would be able to apply successfully these algorithms to their problems.
- Due to the use of a strict object-oriented approach for the library and software tool, these can be used on any machine with Java. As a result, any researcher can use KEEL on his or hers machine, independently of the operating system.

KEEL has been developed with the idea of being easily extended with new algorithms. Therefore, our aim in this work is to offer the possibility for researchers to integrate their own approaches in this software tool, introducing some basic guidelines that the developer may have into account for managing the specific constraints of the KEEL tool. Hence, we have made available a source code template which manages all the restrictions of the KEEL software, including the input and output functions, the parameters parsing, and the class structure. We will describe in detail this template showing a simple example of algorithm codification using the Chi et al.'s rule generation method [9].

This contribution is complemented with a number of online resources as a PDF Manual that describes the main KEEL features, the template for integrating new algorithms and the source code samples for the Chi et al.'s method which is presented in this work. Furthermore, the user may found more specific information and advanced topics at the KEEL website (`http://www.keel.es`).

In order to do that, the contribution is organized as follows. Section 2 presents an introduction on the KEEL software tool, including a short description of its

[3] http://www.oracle.com/technology/products/bi/odm
[4] http://www.angoss.com/products/studio/index.php
[5] http://www.kdnuggets.com/software
[6] http://the-data-mine.com/bin/view/Software

structure and design of experiments. In Section 3 we describe how to implement or to import an algorithm into the KEEL software tool. Next, Section 4 shows an example of codification using the KEEL template. Finally, in Section 5 some concluding remarks are pointed out.

2 KEEL Description

KEEL is a software tool to assess EAs for DM problems including regression, classification, clustering, pattern mining and so on. The presently available version of KEEL consists of the following function blocks (see Fig. 1):

- *Data Management*: This part is made up of a set of tools that can be used to export and import data in other formats to or KEEL format, data edition and visualization, to apply partitioning to data and so on.
- *Design of Experiments*: The aim of this part is the design of the desired experimentation over the selected data sets and providing for many options in different areas: type of validation, type of learning (classification, regression, unsupervised learning) and so on. Once the experiment has been generated, the user may execute it in batch mode.
- *Educational Experiments*: With a similar structure to the previous part, this allows for the design of experiments that can be run step-by-step in order to display the learning process of a certain model by using the software tool for educational purposes.

This structure makes KEEL software useful for different types of user, who expect to find different functionalities in a DM software. Shortly, we describe the main features of KEEL:

- It presents a large collection of EAs for predicting models, pre-processing (evolutionary feature and instance selection) and post-processing (evolutionary tuning of fuzzy rules). It also contains some state-of-the-art methods for

Fig. 1. Screenshot of the main window of KEEL software tool

different areas of DM such as decision trees, fuzzy rule based systems or interval rule-based learning.
- It includes data pre-processing algorithms proposed in specialized literature: data transformation, discretization, instance selection and feature selection.
- It has a statistical library to analyze algorithms' results. It comprises a set of statistical tests for analyzing the suitability of the results and performing parametric and non-parametric comparisons among the algorithms.
- Some algorithms have been developed by using Java Class Library for Evolutionary Computation (JCLEC) [10].
- It provides a user-friendly interface, oriented to the analysis of algorithms.
- The software is aimed to create experimentations containing multiple data sets and algorithms connected among themselves to obtain an expected results. Experiments are independently script-generated from the user interface for an off-line run in the same or other machines.
- KEEL also allows creating experiments in on-line mode, aiming an educational support in order to learn the operation of the algorithm included.

For more information about the main features of the KEEL tool, such as the *Data Management*, the *Design of Experiments* function block, or the *On-Line Module for Computer-Based Education*, please refer to [8].

3 Integration of New Algorithms into the KEEL Tool

This section is devoted to describe in detail how to implement or to import an algorithm into the KEEL software tool. The KEEL philosophy tries to include the less possible constrains for the developer, in order to ease the inclusion of new algorithm within this tool. In fact, each algorithm has its source code on a single folder and does not depends on a specific structure of classes, making straightforward the integration of new methods.

We enumerate the list of details to take into account before codifying a method for the KEEL software, which is also detailed at the KEEL Reference Manual (`http://www.keel.es/documents/KeelReferenceManualV1.0.pdf`).

- The programming language used is Java. In this manner, algorithms which are encoded in C, C++, Perl and so on, must be translated.
- The parameters are read from a single file, which includes the name of the algorithm, the path of the training, validation and test input files, together with the path for the training and test output files and a list of parameters-values for the algorithm. A complete description of the parameters file can be found at Section 3 of the KEEL Manual.
- The input data-sets follow a specific format that extends the "arff" files by completing the header with more information about the attributes of the problem. Next, the list of examples is included, which is given by rows with the attribute values separated by commas. For more information about the input data-sets files please refer to Section 4 of the KEEL Manual. Furthermore, in order to ease the data management, we have developed an API data-set whose main features are described at Section 7 of the Manual.

– The output format consists in a header, which follows the same scheme as the input data, and two columns with the output values for each example separated with a whitespace. The first value corresponds to the expected output, and the second one to the predicted value. All methods must generate two output files: one for training and another one for test. For more information about the obligatory output files please refer to Section 5 of the KEEL Manual.

Although the list of constraints is short, from the KEEL development team we have created a simple template that manages all these features. Our KEEL template includes four classes:

1. **Main:** This class contains the main instructions for launching the algorithm. It reads the parameters from file and builds the "algorithm object".
2. **ParseParameters:** This class manages all the parameters, from the input and output files, to every single parameter stored in the parameters file.
3. **myDataset:** This class is an interface between the classes of the API dataset and the algorithm. It contains the basic options related to the data access.
4. **Algorithm:** This class is devoted to store the main variables of the algorithm and to call the different procedures for the learning stage. It also contains the functions for writing the obligatory output files.

The template can be downloaded following the link `http://www.keel.es/software/KEEL_template.zip`, which additionally supplies the user with the whole API data-set together with the classes for managing files and the random number generator.

Most of the functions of the classes presented above are self-explicative and fully documented for helping the developer to understand their use. Nevertheless, in the next section we will explain in detail how to encode a simple algorithm within the KEEL software tool.

4 Encoding Example Using the Chi et al.'s Method

Including new algorithms in the KEEL software tool is very simple by using the source code template presented in the previous section. We will show how this template enables the programming within KEEL to be straightforward, since the user does not need to pay attention to the specific KEEL constraints because they are completely covered by the functions implemented in the template. To illustrate this, we have selected one classical and simple method, the Chi et al.'s rule learning procedure [9].

Neither the Main class nor the ParseParameters nor myDataset need to be modified, and we just need to focus our effort in the Algorithm class. We enumerate below the steps for adapting this class for this specific algorithm:

1. First of all, we must store all the parameters values within the constructor of the algorithm. Each parameter is selected with the `getParameter` function

using its corresponding position in the parameter file, whereas the optional output files are obtained using the function `getOutputFile`. Furthermore, the constructor must check the capabilities of the algorithm, related to the data-set features, that it, whether it has missing values, real or nominal attributes, and so on.

```
public Fuzzy_Chi(parseParameters parameters) {
  train = new myDataset(); val = new myDataset(); test = new myDataset();
  try { System.out.println("\nReading the training set: " +
    parameters.getTrainingInputFile());
    train.readClassificationSet(parameters.getTrainingInputFile(), true);
    System.out.println("\nReading the validation set: " +
    parameters.getValidationInputFile());
    val.readClassificationSet(parameters.getValidationInputFile(), false);
    System.out.println("\nReading the test set: " + parameters.getTestInputFile());
    test.readClassificationSet(parameters.getTestInputFile(), false);
  } catch (IOException e) {
    System.err.println( "There was a problem while reading the input data-sets: + e);
    somethingWrong = true;}

  //We may check if there are some missing attributes
  somethingWrong = somethingWrong || train.hasMissingAttributes();
  outputTr = parameters.getTrainingOutputFile();
  outputTst = parameters.getTestOutputFile();
  fileDB = parameters.getOutputFile(0);
  fileRB = parameters.getOutputFile(1);

  //Now we parse the parameters
  nLabels = Integer.parseInt(parameters.getParameter(0));
  String aux = parameters.getParameter(1); //Computation of the compatibility degree
  combinationType = PRODUCT;
  if (aux.compareToIgnoreCase("minimum") == 0) {combinationType = MINIMUM;}
  aux = parameters.getParameter(2); ruleWeight = PCF_IV;
  if (aux.compareToIgnoreCase("Certainty_Factor") == 0) {ruleWeight = CF;}
  else if (aux.compareToIgnoreCase("Average_Penalized_Certainty_Factor") == 0) {
  ruleWeight = PCF_II;}
  else if (aux.compareToIgnoreCase("No_Weights") == 0){ruleWeight = NO_RW;}
  aux = parameters.getParameter(3);inferenceType = WINNING_RULE;
  if (aux.compareToIgnoreCase("Additive_Combination") == 0) {
  inferenceType = ADDITIVE_COMBINATION;}
}
```

2. Next, we execute the main process of the algorithm (procedure `execute`). The initial step is to abort the program if we have found some problem during the building phase of the algorithm (constructor). If everything is right, we perform the algorithm's operations. In the case of the Chi et al.'s method we must first build the Data Base (DB) and then generate the Rule Base (RB). When this process is complete, we perform the final output operations:

```
public void execute() {
  if (somethingWrong) { //We do not execute the program
    System.err.println("An error was found, the data-set have missing values");
    System.err.println("Please remove those values before the execution");
    System.err.println("Aborting the program");
    }//We should not use the statement: System.exit(-1);
  else { //We do here the algorithm's operations
    nClasses = train.getnClasses();
    dataBase = new DataBase(train.getnInputs(), nLabels,
    train.getRanges(),train.getNames());
    ruleBase = new RuleBase(dataBase, inferenceType, combinationType,
    ruleWeight, train.getNames(), train.getClasses());
    System.out.println("Data Base:\n"+dataBase.printString());
    ruleBase.Generation(train);
```

```
    dataBase.writeFile(this.fileDB);
    ruleBase.writeFile(this.fileRB);
    //Finally we should fill the training and test output files
    double accTra = doOutput(this.val, this.outputTr);
    double accTst = doOutput(this.test, this.outputTst);
    System.out.println("Accuracy obtained in training: "+accTra);
    System.out.println("Accuracy obtained in test: "+accTst);
    System.out.println("Algorithm Finished");}
}
```

3. We write in an output file the DB and the RB to save the generated fuzzy model, and then we continue with the classification step for both the validation and test files. The `doOutput` procedure simply iterates all examples and returns the predicted class as a string value (in regression problems it will return a double value). This prediction is carried out in the `classificationOutput` function, which only runs the Fuzzy Reasoning Method of the generated RB:

```
private double doOutput(myDataset dataset, String filename) {
  String output = new String(""); int hits = 0;
  output = dataset.copyHeader(); //we insert the header in the output file
  //We write the output for each example
  for (int i = 0; i < dataset.getnData(); i++) { //for classification:
    String classOut = this.classificationOutput(dataset.getExample(i));
    output += dataset.getOutputAsString(i) + " " + classOut + "\n";
    if (dataset.getOutputAsString(i).equalsIgnoreCase(classOut)){
      hits++; }}
  Files.writeFile(filename, output);
  return (1.0*hits/dataset.size());
}

private String classificationOutput(double[] example) {
  String output = new String("?");
  /** Here we should include the algorithm directives to generate the
  classification output from the input example */
  int classOut = ruleBase.FRM(example);
  if (classOut >= 0) { output = train.getOutputValue(classOut);}
  return output;
}
```

Once the algorithm has been implemented, it can be executed directly on a terminal with the parameters file as argument. Nevertheless, when included within the KEEL software, the user can create a complete experimentation with automatically generated scripts for a batch-mode execution. Furthermore, we must clarify that the so-named "validation file" is used when an instance-selection preprocessing step is peformed, and contains the original training set data; hence, the training and validation files match up in the remaining cases.

Finally, we must remark that the complete source code for the Chi et al.'s method (together with the needed classes for the fuzzy rule generation step) can be downloaded at `http://www.keel.es/software/Chi_source.zip`.

5 Conclusions

The objective of this work was to facilitate the implementation and integration of new approaches within the KEEL software tool. In brief, we have shown the simplicity of adding a new algorithm into the KEEL software with the aid of a

Java template specifically designed with this aim. In this manner, the developer has only to focus on the inner functions of his or hers algorithm itself and not to the specific requirements of the KEEL tool, namely the input and output functions, the parameter parsing, or the class structure.

This situation eases the compilation by different researchers of many approaches that can be easily employed for future empirical studies in different fields of Machine Learning.

We have shown that the implementation of the any algorithm (Chi et al.'s in this case) is practically independent of the KEEL software tool when the Java template is used as a basis for the codification.

Acknowledgment

This work has been supported by the Spanish Ministry of Education and Science under Project TIN2008-06681-C06-01.

References

1. Han, J., Kamber, M.: Data mining: Concepts and Techniques, 2nd edn. Morgan Kaufmann Publishers Inc., San Francisco (2006)
2. Romero, C., González, P., Ventura, S., del Jesus, M., Herrera, F.: Evolutionary algorithms for subgroup discovery in e-learning: A practical application using moodle data. Expert Systems with Applications 36(2), 1632–1644 (2009)
3. Otero, J., Sánchez, L., Alcalá-Fdez, J.: Fuzzy-genetic optimization of the parameters of a low cost system for the optical measurement of several dimensions of vehicles. Soft Computing 12(8), 751–764 (2008)
4. Mucientes, M., Moreno, D., Bugarín, A., Barro, S.: Design of a fuzzy controller in mobile robotics using genetic algorithms. Applied Soft Computing 7(2), 540–546 (2007)
5. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, San Francisco (2005)
6. Abeel, T., de Peer, Y.V., Saeys, Y.: Java-ML: A machine learning library. Journal of Machine Learning Research 10, 931–934 (2009)
7. Sonnenburg, S., Braun, M.L., Ong, C.S., Bengio, S., Bottou, L., Holmes, G., LeCun, Y., Müller, K.R., Pereira, F., Rasmussen, C.E., Rätsch, G., Schölkopf, B., Smola, A., Vincent, P., Weston, J., Williamson, R.: The need for open source software in machine learning. Journal of Machine Learning Research 8, 2443–2466 (2007)
8. Alcalá-Fdez, J., Sánchez, L., García, S., del Jesus, M., Ventura, S., Garrell, J., Otero, J., Romero, C., Bacardit, J., Rivas, V., Fernández, J., Herrera, F.: KEEL: A software tool to assess evolutionary algorithms to data mining problems. Soft Computing 13(3), 307–318 (2009)
9. Chi, Z., Yan, H., Pham, T.: Fuzzy algorithms with applications to image processing and pattern recognition. World Scientific, Singapore (1996)
10. Ventura, S., Romero, C., Zafra, A., Delgado, J.A., Hervás, C.: JCLEC: a java framework for evolutionary computation. Soft Computing 12(4), 381–392 (2007)