Journal of Machine Learning Research 01 (2014) 1-4 Submitted 11/11; Revised 07/14; Published 10/14

A Classification Module for Genetic Programming Algorithms in JCLEC

Alberto Cano ACANO@UCO.ES
José María Luna JMLUNA@UCO.ES
Amelia Zafra AZAFRA@UCO.ES
Sebastián Ventura SVENTURA@UCO.ES
Department of Computer Science and Numerical Analysis
University of Córdoba, Spain

Editor: Editor name

Abstract

JCLEC-Classification is a usable and extensible open source library for genetic programming classification algorithms. It houses implementations of rule-based methods for classification based on genetic programming, supporting multiple model representations and providing to users the tools to implement any classifier easily. The software is written in Java and it is available from `http://jclec.sourceforge.net/classification` under the GPL license.

Keywords: Classification, Evolutionary Algorithms, Genetic Programming, JCLEC

1. Introduction

In the last decade, the increasing interest in storing information has led to its automatic processing, discovering knowledge that is potentially useful. Data mining involves the use of data analysis tools to discover this knowledge previously unknown, valid patterns, and close relationships in databases. One of the most used data mining tasks is classification, which learns from a set of training examples to produce predictions about future examples.

The classification models are being applied to enormous databases in areas such as bioinformatics, marketing, banks or web mining. Existing classification libraries provide algorithms following many different methodologies. However, it is difficult to find a library that contains GP (genetic programming) algorithms, an important evolutionary computation paradigm. The conceptual difficulty of GP makes it difficult to implement algorithms following this paradigm despite its algorithms perform well as it is proved by many researchers (Espejo et al., 2010).

GP is an efficient and flexible heuristic technique that uses complex representations such as trees. This technique provides comprehensible models, which are useful in different application domains. For instance, it is applied to supervised learning tasks like regression, classification and unsupervised learning tasks like clustering and association. In classification tasks, the application of GP is an important issue since it may offer results that are comprehensible to humans. Additionally, it offers interesting advantages such as flexibility, and the possiblity of using different kinds of representations, e.g., decision trees, rule-based systems, discriminant functions, etc. An extension of GP is grammar-guided genetic pro-

gramming (G3P), which makes the knowledge extracted more expressive and flexible by means of a context-free grammar (McKay et al., 2010).

This paper presents an open source software for researchers and end-users to develop classification algorithms based on GP and G3P models. It is an intuitive and usable tool which extends the JCLEC evolutionary computation library (Ventura et al., 2007). The software presented includes some GP and G3P proposals described in literature, and provides the necessary classes and methods to develop any kind of evolutionary algorithms for solving classification problems easily.

This paper is organized as follows. Firstly, Section 2 provides a description of the module, its structure and the way to use it. Finally, the documentation and the requirements of this module are outlined in Section 3.

2. Description of the module

The classification module is presented in this section, describing the library structure and its main characteristics.

2.1 Structure of the module

The *net.sf.jclec.problem.classification.base* package roots the hierarchical structure of the classification module, and provides the abstract classes with the properties and methods that any classification algorithm must contain, e.g., *ClassificationAlgorithm*, *ClassificationReporter*, *Rule* and *RuleBase*. A new algorithm included in the module should inherit from these classes regardless the classification model. In this context, we focus on rule-based classifiers which comprise one or more classification rules, each of them being a knowledge representation model consisting of an antecedent and a consequent. The antecedent of each classification rule is made up of a series of conditions to be met by an instance to consider that it belongs to the class specified by the consequent.

Based on whether an algorithm uses a GP or G3P encoding, JCLEC-Classification makes a differentiation between expression-tree and syntax-tree respectively. In such a way, each GP classification individual is represented by means of the *ExprTreeRuleIndividual* class, which represents an individual, comprising all the features required to do it: the genotype, the phenotype and the fitness function value. The nodes and functions in GP trees are defined by the *ExprTreeSpecies* class. Similarly to GP individuals, the *SyntaxTreeRuleIndividual* class specifies all the features required to represent a G3P individual, while the *SyntaxTreeSpecies* allows us to define the terminal and nonterminal symbols of the grammar used to generate individuals. Furthermore, the module allows to encode multiple syntax and expression trees for Pittsburgh style encodings or multi expression programming by means of the *MultiExprTree* and *MultiSyntaxTree* classes.

In order to represent the phenotype of a rule-base individual, crisp and fuzzy rules are generated by using the *CrispRule* and *FuzzyRule* classes, respectively. These classes provide the antecedent of the rule in an expression-tree shape and the consequent assigned to this antecedent. In addition, methods to classify a whole dataset or a particular instance are provided in these classes. These methods compute whether the antecedent of a rule satisfies an instance, returning the consequent of the rule, otherwise the instance is not covered by the antecedent and therefore no predictions can be made. Besides those packages that repre-

sent the main characteristics of any individual, the *net.sf.jclec.problem.classification.listener* package to make reports for the train and test classification processes is provided. This package contains the *RuleBaseReporter* class with methods to make reports specifying the classifier features such as the rule base, the number of rules, the average number of conditions, the percentage of correct predictions, the percentage of correct predictions per class, the geometric mean, the kappa rate and the confusion matrix.

Finally, it is noteworthy that several utility classes, which make it easy to load data from KEEL [1] and ARFF [2] formatted files, are provided by a *dataset* package. Three different attribute types may be represented by this package, integer, continuous and categorical, and a number of characteristics from the dataset are given, comprising type of attributes, number of classes, number of instances, etc.

The module houses three G3P classification algorithms (De Falco et al., 2001; Bojarczuk et al., 2004; Tan et al., 2002), which can guide developers to write new algorithms.

2.2 Usage of the module

Including new classification algorithms in this module is very simple. We focus on the algorithm described by Bojarczuk et al. (Bojarczuk et al., 2004). This algorithm, which is provided in the module (see the *net.sf.jclec.problem.classification.algorithm.bojarczuk* package), is constructed with only three additional classes. One of them, the *BojarczukAlgorithm* class is inherited from the *ClassificationAlgorithm* class and provides the own features of this algorithm.

Another class required to be implemented is the evaluator, which computes the fitness of the individuals. This class, named *BojarczukEvaluator* in this algorithm, inherits from the JCLEC core *AbstractParallelEvaluator* class or from the *AbstractEvaluator* class, depending on whether the individuals are evaluated in a sequential or parallel way.

Finally, a class to define the grammar to be followed in the individual generation stage is implemented. This class, named *BojarczukSyntaxTreeSpecies* in this example, inherits from the class *SyntaxTreeSpecies* since G3P individuals are defined in this algorithm.

Only defining these three classes, the complete classification algorithm is represented. Due to the core of this module is JCLEC, before an algorithm is ready to run, it is necessary to carry out a set-up process by using a configuration file as shown in Figure 1. This configuration file and the steps required to execute the algorithm are described in the JCLEC website. In this file we specify those parameters required such as the algorithm to be run, the parent selector, the genetic operators, the evaluator, etc. All the required parameters are provided by JCLEC, existing a numerous variety of them as it is described in the JCLEC specification (Ventura et al., 2007).

3. Documentation and requirements

The JCLEC-Classification online documentation [3] describes the software packages, presents a user oriented usage example, as well as developer information to include new algorithms, API reference and running tests. JCLEC requires Java 1.7, Apache commons logging 1.1,

1. http://www.keel.es
2. http://www.cs.waikato.ac.nz/ml/weka/arff.html
3. http://jclec.sourceforge.net/data/JCLEC-classification.pdf

```
<experiment>
   <process algorithm-type="net.sf.jclec.problem.classification.algorithm.bojarczuk.BojarczukAlgorithm">
      <rand-gen-factory seed="123456789" type="net.sf.jclec.util.random.RanecuFactory"/>
      <population-size>100</population-size>
      <max-of-generations>100</max-of-generations>
      <max-deriv-size>20</max-deriv-size>
      <dataset type="net.sf.jclec.problem.util.dataset.ArffDataSet">
         <train-data>data/iris/iris-10-1tra.arff</train-data>
          <test-data>data/iris/iris-10-1tst.arff</test-data>
         <attribute-class-name>Class</attribute-class-name>
      </dataset>
      <recombination-prob>0.8</recombination-prob>
      <copy-prob>0.01</copy-prob>
      <listener type="net.sf.jclec.problem.classification.listener.RuleBaseReporter">
         <report-dir-name>reports/reportFreitas</report-dir-name>
         <global-report-name>summaryFreitas</global-report-name>
         <report-frequency>10</report-frequency>
      </listener>
   </process>
</experiment>
```

Figure 1: Sample configuration file

Apache commons collections 3.2, Apache commons configuration 1.5, Apache commons lang 2.4, and JUnit 3.8 (for running tests).

Acknowledgments

This research was supported by the Spanish Ministry of Science and Technology project TIN-2011-22408, the Ministry of Education FPU grants AP2010-0041 and AP2010-0042, and FEDER funds.

References

C. C. Bojarczuk, H. S. Lopes, A. A. Freitas, and E. L. Michalkiewicz. A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets. *Artificial Intelligence in Medicine*, 30(1):27–48, 2004.

I. De Falco, A. Della Cioppa, and E. Tarantino. Discovering interesting classification rules with genetic programming. *Applied Soft Computing*, 1(4):257–269, 2001.

P. G. Espejo, S. Ventura, and F. Herrera. A Survey on the Application of Genetic Programming to Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 40(2):121–144, 2010.

R. McKay, N. Hoai, P. Whigham, Y. Shan, and M. O'Neill. Grammar-based Genetic Programming: a Survey. *Genetic Programming and Evolvable Machines*, 11:365–396, 2010.

K. C. Tan, A. Tay, T. H. Lee, and C. M. Heng. Mining multiple comprehensible classification rules using genetic programming. In *Proceedings of the Evolutionary Computation on 2002. CEC '02*, volume 2, pages 1302–1307, 2002.

S. Ventura, C. Romero, A. Zafra, J.A. Delgado, and C. Hervás. JCLEC: a Java Framework for Evolutionary Computation. *Soft Computing*, 12:381–392, 2007.