Genetic Programming for Automatically Constructing Data Mining Algorithms

Alex A. Freitas
University of Kent, UK

Gisele L. Pappa
Federal University of Minas Geras, Brazil

INTRODUCTION

At present there is a wide range of data mining algorithms available to researchers and practitioners (Witten & Frank, 2005; Tan et al., 2006). Despite the great diversity of these algorithms, virtually all of them share one feature: they have been *manually* designed. As a result, current data mining algorithms in general incorporate human biases and preconceptions in their designs.

This article proposes an alternative approach to the design of data mining algorithms, namely the automatic creation of data mining algorithms by means of Genetic Programming (GP) (Pappa & Freitas, 2006). In essence, GP is a type of Evolutionary Algorithm – i.e., a search algorithm inspired by the Darwinian process of natural selection – that evolves computer programs or executable structures.

This approach opens new avenues for research, providing the means to design novel data mining algorithms that are less limited by human biases and preconceptions, and so offer the potential to discover new kinds of patterns (or knowledge) to the user. It also offers an interesting opportunity for the automatic creation of data mining algorithms tailored to the data being mined.

BACKGROUND

An Evolutionary Algorithm (EA) is a computational problem-solving method inspired by the process of natural selection. In essence, an EA maintains a population of "individuals", where each individual represents a candidate solution to the target problem. EAs are iterative generate-and-test procedures, where at each "generation" (iteration) a population of individuals is generated and each individual has its "fitness" computed. The fitness of an individual is a measure of the quality of its corresponding candidate solution. The higher the fitness of an individual, the higher the probability that the individual will be selected to be a "parent" individual. Certain operations (often "crossover" and/or "mutation" operations inspired by their natural counterparts) are applied to the selected parent individuals in order to produce "children" individuals. The important point is that, since the children are in general produced from parents selected based on fitness, the children (new candidate solutions) tend to inherit parts of the good solutions of the previous generation, and the population as a whole gradually evolves to fitter and fitter individuals (better and better solutions to the target problem). For a comprehensive review of EAs in general the reader is referred to (De Jong, 2006; Eiben & Smith, 2003), and a comprehensive review of EAs for data mining can be found in (Freitas, 2002).

The basic principle of EAs – i.e., artificial evolution of candidate solutions, where parent solutions are selected based on their fitness in order to create new children solutions – still holds in Genetic Programming (GP). The main distinguishing feature of GP – by comparison with other types of EA – is that the candidate solutions represented by GP individuals are (or at least should be) computer programs or executable structures. GP has been an active research field for about 15 years (Koza, 1992; Banzhaf et al., 1998; Langdon & Poli, 2002; Koza, 2003), and Koza (2006) reports 36 instances where GP discovered a solution infringing or duplicating some patent, which led Koza to claim that GP is an automated invention machine that routinely produces human-competitive results. However, the creation of a GP system for automatically evolving a full data mining algorithm, as proposed in this article, is a new research topic which is just now starting to be systematically explored, as discussed in the next section.

MAIN FOCUS

The problem of automatically evolving a data mining algorithm to solve a given data mining task is very challenging, because the evolved algorithm should be generic enough to be applicable to virtually any data set that can be used as input to that task. For instance, an evolved algorithm for the classification task should be able to mine any classification data set (i.e., a data set having a set of records, each of them containing a class attribute and a set of predictor attributes – which can have different data types); an evolved clustering algorithm should be able to mine any clustering data set, etc.

In addition, the automatic evolution of a fully-fledged data mining algorithm requires a GP method that not only is aware of the basic structure of the type of data mining algorithm to be evolved, but also is capable of avoiding fatal programming errors – e.g. avoiding infinite loops when coping with "while" or "repeat" statements. Note that most GP systems in the literature do not have difficulties with the latter problem because they cope only with relatively simple operations (typically mathematical and logical operations), rather than more sophisticated programming statement such as "while" or "repeat" loops. To cope with the two aforementioned problems, a promising approach is to use a grammar-based GP system, as discussed next.

Grammar-Based Genetic Programming

Grammar-Based Genetic Programming (GGP) is a particular type of GP where a grammar is used to create individuals (candidate solutions). There are several types of GGP (Wong & Leung, 2000; O'Neill & Ryan, 2003). A type of GGP particularly relevant for this article consists of individuals that directly encode a candidate program in the form of a tree, namely a derivation tree produced by applying a set of derivation steps of the grammar. A derivation step is simply the application of a production rule to some non-terminal symbol in the left-handed side of the rule, producing the (non-terminal or terminal) symbol in the right-handed side of the rule. Hence, each individual is represented by a derivation tree where the leaf nodes are terminal symbols of the grammar and the internal nodes are the non-terminal symbols of the grammar.

The use of such a grammar is important because it not only helps to constrain the search space to valid algorithms but also guides the GP to exploit valuable background knowledge about the basic structure of the type of data mining algorithm being evolved (Pappa & Freitas, 2006; Wong & Leung, 2000).

In the context of the problem of evolving data mining algorithms the grammar incorporates background knowledge about the type of data mining algorithm being evolved by the GP. Hence, broadly speaking, the non-terminal symbols of the grammar represent high-level descriptions of the major steps in the pseudo-code of a data mining algorithm, whilst the terminal symbols represent a lower-level implementation of those steps.

A New Grammar-Based GP System for Automatically Evolving Rule Induction Algorithms

The previously discussed ideas about Grammar-Based Genetic Programming (GGP) were used to create a GGP system that automatically evolves a rule induction algorithm, guided by a grammar representing background knowledge about the basic structure of rule induction algorithms (Pappa & Freitas, 2006), (Pappa 2007). More precisely, the grammar contains two types of elements, namely:

a. general programming instructions – e.g. *if-then* statements, *for/while* loops; and
b. procedures specifying major operations of rule induction algorithms – e.g., procedures for initializing a classification rule, refining a rule by adding or removing conditions to/from it, selecting a (set of) rule(s) from a number of candidate rules, pruning a rule, etc.

Hence, in this GGP each individual represents a candidate rule induction algorithm, obtained by applying a set of derivation steps from the rule induction grammar. The terminals of the grammar correspond to modular blocks of Java programming code, so each individual is actually a Java program implementing a full rule induction algorithm.

This work can be considered a major "case study" or "proof of concept" for the ambitious idea of automati-