Classification Using an Online Genetic Algorithm

Brian D. Davison

Department of Computer Science
Rutgers, The State University of New Jersey
Piscataway, NJ 08855 USA
davison@cs.rutgers.edu

Genetic Algorithms (GAs) purport to mimic the behavior of natural selection. Many GAs, however, try to optimize their populations by means of a static fitness function — one that is derived from performance on a fixed set of examples. We propose an architecture for an *online genetic algorithm* (OLGA) for classification. An OLGA differs from standard genetic algorithms in that it does not repeatedly evaluate individuals against a fixed set of training examples. Instead, it is presented with a series of training examples, one at a time, and does not retain the entire set for training.

Being online and incremental, OLGAs, like evolution strategies (Dasgupta & Michalewicz 1997), are applicable to tasks that require continuous learning to handle *concept drift* such as in adaptive systems as well as tasks in which the dataset is too large to be kept on hand for repeated evaluation (as in many online and interactive problems). By evaluating individuals on recent examples, OLGAs also better mimic the behavior of natural selection, as real organisms live in environments that are not identical to that of their ancestors.

An OLGA is a GA, complete with a population of size p, fitness function, recombination operators, and mutation operators. However, unlike a traditional GA, the fitness of an individual changes over time, as it is exposed to more examples. In order to do so, we track the number of examples seen and the number of examples classified correctly for each individual and class. The fitness is calculated by summing over all classes the weight of the class times a function of $correctcount$ and $totalseen$. The key idea in creating such a fitness function is to support newly created individuals so that they are not replaced before they have seen a reasonable number of examples (and thus have some estimate of their true fitness).

The individuals can represent entire solutions, or can form a solution as a group, as is done in most classifier systems (Goldberg 1989). When an individual is not a complete solution (such as rule that fires only when it applies), we track the number of times the rule was applied (instead of the number of examples seen). The overall classification is by majority vote of those individuals making a classification.

We have an implementation of an OLGA that we have applied to a set of related sample problems to test the feasibility of the architecture. The examples are a set of 6 boolean attributes, $a_1, a_2, ..., a_6$, (given the values 0 or 1), with the class set to true only when certain attributes are all 1. The value of a_i is set to 1 with a fixed probability.

The individuals in the GA for these problems represent the weights in weighted sum of attributes form, with a threshold of 0 to generate classifications ($\sum_i w_i a_i > 0$). Thus each individual has 7 genes (6 weights for the attributes, plus a bias weight). The possible values for each weight are -1, 0, and 1, with equal probability. The fitness function is as follows: $fitness_i = \sum_{c \in classes} \frac{1+correctcount_{i,c}}{1+totalseen_{i,c}}$

The worst performing individual is replaced with a new individual created by a single operator that combines multipoint cross-over with mutation. Parents are randomly selected from the population but with fairly strong pressure toward the better scoring individuals. Each gene in the resulting child has a small chance (the mutation rate) of being set to one of the three values randomly. For speed of testing, we set $p = 20$, and we wait at startup until we've seen at least $1.5p$ before we start replacing individuals.

The system was tested on seven variations of this problem. Some classification functions were representable and easily learned, such as: $class(\overline{x}) = \cap(a_2, a_3)$. Others were not fully realizable by these individuals, such as $class(\overline{x}) = \cap(a_2, a_3, a_4)$. We also varied the attribute setting probability, which varied the overall class distribution (ranging from 22% to 64% positive examples). Throughout these variations, the system was able to achieve reasonable classification performance (86% to 98% accuracy). In addition, we attempted some more unusual tests, such as combining multiple conflicting classifiers and sharp changeovers from one distribution to another. Even on these problems, the OLGA was able to generate fairly high cumulative classification accuracies. While this is just one artificial domain, we take this success as initial, albeit weak, evidence for the applicability of the OLGA architecture. In future work we plan to apply this approach to non-artificial problems as well as compare it to other online algorithms.

References

Dasgupta, D., and Michalewicz, Z., eds. 1997. *Evolutionary algorithms in engineering applications*. New York, NY: Springer-Verlag.

Goldberg, D. E. 1989. *Genetic algorithms in search, optimization, and machine learning*. New York, NY: Addison-Wesley.