Evolutionary Mining of Rule Ensembles

Jorge Muruzábal
University Rey Juan Carlos, Spain

INTRODUCTION

Ensemble rule based classification methods have been popular for a while in the machine-learning literature (Hand, 1997). Given the advent of low-cost, high-computing power, we are curious to see how far can we go by repeating some basic learning process, obtaining a variety of possible inferences, and finally basing the global classification decision on some sort of ensemble summary. Some general benefits to this idea have been observed indeed, and we are gaining wider and deeper insights on exactly why this is the case in many fronts of interest.

There are many ways to approach the ensemble-building task. Instead of locating ensemble members independently, as in Bagging (Breiman, 1996), or with little feedback from the joint behavior of the forming ensemble, as in Boosting (see, e.g., Schapire & Singer, 1998), members can be created at random and then made subject to an evolutionary process guided by some fitness measure. Evolutionary algorithms mimic the process of natural evolution and thus involve populations of individuals (rather than a single solution iteratively improved by hill climbing or otherwise). Hence, they are naturally linked to ensemble-learning methods. Based on the long-term processing of the data and the application of suitable evolutionary operators, fitness landscapes can be designed in intuitive ways to prime the ensemble's desired properties. Most notably, beyond the intrinsic fitness measures typically used in pure optimization processes, fitness can also be endogenous, that is, it can prime the context of each individual as well.

BACKGROUND

A number of evolutionary mining algorithms are available nowadays. These algorithms may differ in the nature of the evolutionary process or in the basic models considered for the data or in other ways. For example, approaches based on the genetic programming (GP), evolutionary programming (EP), and classifier system (CS) paradigms have been considered, while predictive rules, trees, graphs, and other structures been have evolved. See Eiben and Smith (2003) for an introduction to the general GP, EP, and CS frameworks and Koza, Keane, Streeter, Mydlowec, Yu, and Lanza (2003) for an idea of performance by GP algorithms at the patent level. Here I focus on ensemble –rule based methods for classification tasks or supervised learning (Hand, 1997).

The CS architecture is naturally suitable for this sort of rule assembly problems, for its basic representation unit is the rule, or classifier (Holland, Holyoak, Nisbett, & Thagard, 1986). Interestingly, tentative ensembles in CS algorithms are constantly tested for successful cooperation (leading to correct predictions). The fitness measure seeks to reinforce those classifiers leading to success in each case. However interesting, the CS approach in no way exhausts the scope of evolutionary computation ideas for ensemble-based learning; see, for example, Kuncheva and Jain (2000), Liu, Yao, and Higuchi (2000), and Folino, Pizzuti, and Spezzano (2003).

Ensembles of trees or rules are the natural reference for evolutionary mining approaches. Smaller trees, made by rules (leaves) with just a few tests, are of particular interest. Stumps place a single test and constitute an extreme case (which is nevertheless used often). These rules are more general and hence tend to make more mistakes, yet they are also easier to grasp and explain. A related notion is that of support, the estimated probability that new data satisfy a given rule's conditions. A great deal of effort has been done in the contemporary CS literature to discern the idea of adequate generality, a recurrent topic in the machine-learning arena.

MAIN THRUST

Evolutionary and Tree-Based Rule Ensembles

In this section, I review various methods for ensemble formation. As noted earlier, in this article, I use the ensemble to build averages of rules. Instead of averaging, one could also select the most suitable classifier in each case and make the decision on the basis of that rule alone (Hand, Adams, & Kelly, 2001). This alternative idea may provide additional insights of interest, but I do not analyze it further here.

It is conjectured that maximizing the degree of interaction amongst the rules already available is critical for efficient learning (Kuncheva & Jain, 2000; Hand et al., 2001). A fundamental issue concerns then the extent to which tentative rules work together and are capable of influencing the learning of new rules. Conventional methods like Bagging and Boosting show at most moderate amounts of interaction in this sense. While Bagging and Boosting are useful, well-known data-mining tools, it is appropriate to explore other ensemble-learning ideas as well. In this article, I focus mainly on the CS algorithm. CS approaches provide interesting architectures and introduce complex nonlinear processes to model prediction and reinforcement. I discuss a specific CS algorithm and show how it opens interesting pathways for emergent cooperative behaviour.

Conventional Rule Assembly

In Bagging methods, different training samples are created by bootstraping, and the same basic learning procedure is applied on each bootstrapped sample. In Bagging trees, predictions are decided by majority voting or by averaging the various opinions available in each case. This idea is known to reduce the basic instability of trees (Breiman, 1996).

A distinctive feature of the Boosting approach is the iterative calling to a basic weak learner (WL) algorithm (Schapire & Singer, 1998). Each time the WL is invoked, it takes as input the training set — together with a dynamic (probability) weight distribution over the data — and returns a single tree. The output of the algorithm is a weighted sum itself, where the weights are proportional to individual performance error. The WL learning algorithm needs only to produce moderately successful models. Thus, trees and simplified trees (stumps) constitute a popular choice. Several weight updating schemes have been proposed. Schapire and Singer update weights according to the success of the last model incorporated, whereas in their LogitBoost algorithm, Friedman, Hastie, and Tibshirani (2000) let the weights depend on overall probabilistic estimates. This latter idea better reflects the joint work of all classifiers available so far and hence should provide a more effective guide for the WL in general.

The notion of abstention brings a connection with the CS approach that will be apparent as I discuss the match set idea in the followins sections. In standard boosting trees, each tree contributes a leaf to the overall prediction for any new x input data vector, so the number of expressing rules is the number of boosting rounds independently of x. In the system proposed by Cohen and Singer (1999), the WL essentially produces rules or single leaves C (rather than whole trees). Their classifiers are then maps taking only two values, a real number for those x verifying the leaf and 0 elsewhere. The final boosting aggregation for x is thus unaffected by all abstaining rules (with $x \notin C$), so the number of expressing rules may be a small fraction of the total number of rules.

The General CS-Based Evolutionary Approach

The general classifier system (CS) architecture invented by John Holland constitutes perhaps one of the most sophisticated classes of evolutionary computation algorithms (Hollandet et al., 1986). Originally conceived as a model for cognitive tasks, it has been considered in many (simplified) forms to address a number of learning problems. The nowadays standard stimulus-response (or single-step) CS architecture provides a fascinating approach to the representation issue. Straightforward rules (classifiers) constitute the CS building blocks. CS algorithms maintain a population of such predictive rules whose conditions are hyperplanes involving the wild-card character #. If we generalize the idea of hyperplane to mean "conjunctions of conditions on predictors where each condition involves a single predictor," We see that these rules are also used by many other learning algorithms. Undoubtedly, hyperplane interpretability is a major factor behind this popularity.