Breeding Expression Trees for Symbolic Regression

Ben Wiley and **Jackson Spell**
{bewiley, jaspell}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

Regression analysis is a widely-used technique for fitting known-type functions to data sets for the purposes of making accurate, generalizable projections. But what can we do if we do not know the type of function the data represents? Symbolic regression, using a genetic algorithm, is one way to answer this question. We apply evolutionary principles to this problem by treating a generated set of diverse functions (determined as fit by a fitness function) as ”parents,” combining traits between randomly chosen pairs to render a child set, and repeating over the course of several generations to produce a well-fitting function. This methodology is used over two different sets of data which represent unknown functions. Our results . . . were mixed.

1 Introduction

Linear, polynomial, and other forms of regression analysis are commonly used in the field of statistics in order to fit a function to a data set. Produced functions can be used to predict behavior at arbitrary input values. Typically, regression analysis is used upon data sets for which the approximate function *type* (e.g. linear, quadratic, etc.) is known. Traditional regression analysis is not well-suited, however, for data sets which approximate an unknown function type.

One tried solution for such scenarios is symbolic regression by way of a genetic algorithm — an algorithm that generates sets of candidate solutions, randomly splices together traits from pairs of fit ”parents” into ”children” over successive generations, and occasionally mutates pieces of those candidates, until a strongly fit solution is found.

In our paper we introduce expression trees as a means of representing functional expressions in such a way that they can be easily spliced by a genetic algorithm. We discuss our own implementation of symbolic regression for two known data sets representing unknown functions, inspired by the work of John Koza (**?**).

2 Expression Trees

For the purposes of our experiment, we represent functions as expression trees, in which each node represents either a variable, some constant, or some binary operation (addition, subtraction, multiplication or division). Powers are represented, when applicable, as multplications of variables.

figs/exprtrees.jpg

Figure 1: An example set of expression trees representing functions (**?**).

Trees are randomly constructed with a depth limit of eight, and with strictly downward-pointing references. The root node is always some binary operation, and subsequent nodes have a 25% probability of being a constant, a 25% probability of being a variable, and a 50% probability of being some binary operation. The tree is recursively filled in any direction either until a terminal node is chosen (constant or variable), or until the depth limit has been reached, in which case the chosen node will always be either a constant or variable. Trees are evaluated from the top, so that higher nodes receive higher precedence, and are able to accept one or three variables, depending on the scenario.

Two important operations must be performable on expression trees for purposes of the genetic algorithm: crossover and mutation. During crossover, a random downward reference is picked once from two different trees (below the root), and the subtrees below each are swapped with the assistance of a placeholder reference. During mutation, a random node in the tree is picked to be replaced with another node. The caveat is that mutations must be restricted to categorical similarity. Binary operations may only be swapped with binary operations and variables or constants can only be swapped with variables or constants, so that all tree connections remain relevant and intact.

3 Experiments

This experiment was designed to use a genetic algorithm to adapt a population of expression trees to correctly identify two unknown functions. In separate runs, populations of 1000 expression trees were adapted through 50 generations, with the goal of matching the goal functions.

Genetic crossover was achieved by probabilistically combining expression trees based on their fitness functions. The fitness score of each tree is calculated as the reciprocal of the total error between the expression tree and goal function, evaluated at a large number of points. The reciprocal is taken in order to give more accurate trees, with lower sum errors, higher fitness scores. Trees were then chosen randomly according to a probabilistic distribution weighted by the fitness scores, with trees replaced back into the population after each crossover.

One of the main concepts behind genetic algorithms is that parents in the algorithm pass on traits recognizably to children. By choosing the most fit expression trees to reproduce, the algorithm should, after a number of generations, converge on a locally optimal solution. In order for preferable characteristics from parent expression trees to be transferred to child trees, the crossover operation chose a random node from each tree, then swapped the subtrees extending from the chosen nodes. Child trees therefore were composed of partial sections of each of the parent trees. Because trees were chosen with replacement, and combine randomly during each crossover, populations may have multiple identical trees or trees made by combining the same two parent trees at different nodes.

After each crossover operation, a portion of the population was mutated in order to introduce random variation. For a given expression tree, the mutation operation randomly selected a node and altered the value or mathematical operation at that node based on the original value. To avoid damaging the behavior of the tree too much, thereby potentially lowering the accuracy of the algorithm, the mutated operation or value was chosen to structurally match the original node. Leaf nodes were only replaced with terminals: randomly chosen constants or appropriate variables. Nodes originally containing binary operations ($+$, $-$, $\times$, $/$) were only replaced with other operations of that nature.

It is of note that none of the nodes of the expression trees contain the power operation, e.g. x^2. This was a code optimization choice made because power operations can be created through multiplication of appropriate variables. Removing power operations decreased the chance that an expression tree would become arbitrarily complicated through high powers, instead favoring low integer powers – x, x^2, x^3.

4 Results

In order to help the algorithm converge on the correct solutions, the parameters guiding the behavior of the algorithm were tailored to increase population diversity, maintain successful trees, and correctly adapt to the goal functions. The sampling range of the functions also had significant impact on the algorithm's accuracy. When comparing an expression tree to the goal functions, which determined the fitness of the trees, best results were seen when the error was calculated over a range where the goal function was significantly non-zero. If the fitness was calculated over larger ranges of input values, the expression trees tended strongly towards constant expressions that minimize total error but describe only the asymptotic tails of the function.

Data points taken from the first goal function are displayed in Table **??**. The second goal function is four dimensional, and cannot be conveniently displayed.

Despite careful adjustment of parameters governing tree construction, mutation rate, and the sampling range of the fitness function, we were unable to develop any non-constant expressions for the goal functions as given. While closing the sampling range to only include the area of the fitness function displaying interesting, non-asymptotic behavior did result in lower error amongst the most accurate trees, the genetic algorithm was unable to determine a more accurate function.

In order to attempt to match the first goal function, we inverted the output of the function in order to alter the expression needed to produce it. Surprisingly, this met with success, as the algorithm consistently finds x^2 solutions which closely approximate the goal function. Unlike the runs with the original goal function, the fitness scores converge when evaluating based on the in-

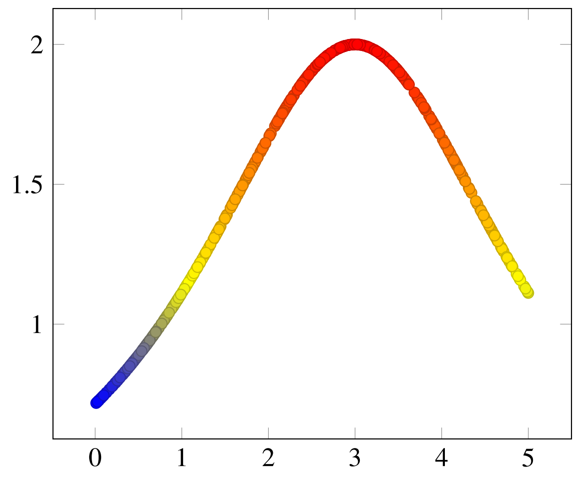

Figure 2: Non-zero section of goal function 1

verted function. Through this technique, we were able to arrive at an accurate function, which we then invert to conclude that the first goal function is

$$y = \frac{10}{x^2 - 6x + 14}$$

Unfortunately, this technique has not been successful for the second goal function, which has 3 input variables. For that reason, we were also unable to plot the data in order to determine an appropriate sampling region or to visualize the function in general. When attempting to fit the second goal function, our algorithm always relies on a constant function. Unfortunately, this is clearly not a correct solution, as the error associated with such trees is massive.

5 Conclusions

In this experiment, we used a genetic algorithm for symbolic regression against two unknown goal functions. We developed a population of expression trees through 50 generations by evaluating the trees' fitness via their error against the goal function. Though this approach did not autonomously solve the goal functions, data manipulation – in the form of taking the reciprocal of the output of the goal function – allowed the algorithm to correctly solve one of the functions. While the variability and potential solving power of genetic programming can seem a catch-all solution, our results indicate that careful data analysis may be needed to supplement the algorithm in order for it to be effective. This problem deserves further research by this team in order to determine a more robust solution to the single input problem and to solve the more complex three input problem.

References

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press.