Symbolic Regression with Genetic Programming

Andrew Miller and **Michael Robertson**
{amiller,mirobertson}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

In this paper, we use genetic programming with expression trees in order to successfully model an unknown data set. The implementation liberates the model from regression selection bias and instead, relies on the evolution process to reveal a best-fit function for forecasting.

1 Introduction

In this section, you should introduce the reader to the problem you are attempting to solve. For example, for the first project: describe the dataset, and the prediction problem that you are investigating. You should also cite and briefly describe other related papers that have tackled this problem (or similar ones) in the past — things that came up during the course of your research. In the AAAI style, citations look like (**?**) (see the comments in the source file `intro.tex` to see how this citation was produced). Conclude by summarizing how the remainder of the paper is organized.

In this experiment, we were given three data sets of 25,000 points and our goal was to use genetic programming with expression trees in order to generate an expression that best models the data and can forecast new data points. The expression tree format allowed us to preserve or alter parts of expressions through crossover and mutation across generations in order to develop better-fitting expressions. Dynamic Expression Trees have been studied by many, including Ferreira and Gepsoft. Expression trees are made up of operator, constant and variable nodes that form expressions when printed traversed in order. These nodes act as genes in our genetic algorithm as seen in Figure 1.

When implementing a genetic algorithm in order to develop a selection regression solution to our data sets, we had to make decisions concerning selection methods, bloat control and fitness evaluation. All of these factors have been previously studied and we based our work off of the work of Poli, Langdon and Blickle and Thiele.

2 Background

Preprocessing

Before running our genetic algorithm, we chose to split our data sets into training and testing data using the 80-20 rule

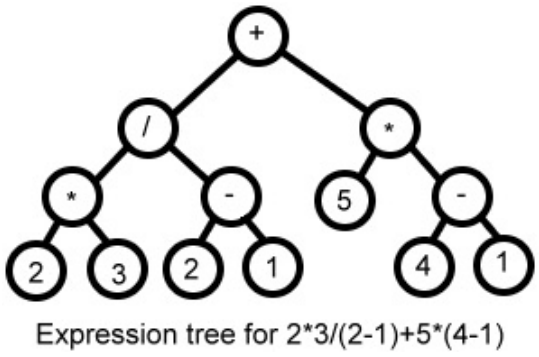

Figure 1: An example expression tree courtesy of `Stack Overflow`.

of thumb. This allowed us to develop our trees with data separate from the data used to evaluate the trees. This was done in order to avoid overfitting.

Fitness

We then ran our algorithm using a starting population size of 50 randomly generated trees. The fitness of trees was evaluated using a combination of mean square error and a penalty proportional to the tree's size to deter bloat. Mean Square Error is a commonly used method of determining the precision of a regression relative to the actual data. The bloat control method was adapted from the work of Poli

Selection

Many methods exist for choosing crossover pairs. We chose to use tournament selection to determine the breeding pairs of trees. Our methods were adapted from the work of Langdon.

3 Experiments

The difficulties of genetic programming present themselves to the designers in the form of infamously many adjustable parameters, all of which influence the end efficacy of the program in interdependent and inscrutable ways. We attempted to find a choice of population size, method of original tree generation, tree size penalty, mutation rate, mutation method, parent selection method (with its own consid-

erable set of variable parameters), and reproduction mechanism which both cause the program to converge to a solution and to maximize the rate of that convergence.

We defined convergence as reaching a below [x] mean-squared error from the test data set, and failure to converge as moving through [x] generations without producing a new child more fit than any of its ancestors. We measured the success of our program by both the percentage of runs that converged, and the number of generations that convergence took.

We investigated different methods of parent selection first, believe that parameter to influence the convergence most. Our research indicated that experts in the field used tournament selection and fitness-proportion ("roulette selection") most often.

Our best results came from a [x] proportion of population size to tournament size, and when we allowed only [y] survivors for each tournament to reproduce randomly with themselves.

Second, we ran fitness proportion selection, where we chose different methods for assigning reproduction probability.

For each of these methods, we allowed the selected parents to randomly reproduce with each other, producing one child per meeting, and repeated the selection process until the size of the child population reached the size of the original adult population.

[Present results] were the questions you were trying to answer? What was the experimental setup (number of trials, parameter settings, etc.)? What were you measuring? You should justify these choices when necessary. The accepted wisdom is that there should be enough detail in this section that I could reproduce your work *exactly* if I were so motivated.

4 Results

We hope to have a success rate of a program with specific parameters defined as percentage of a total number of runs for which it reached below error tolerance, and we will further report the average number of generations needed for it to converge. Finally, we hope to report the equation which models the dataset with a mean-squared-error below our tolerance.

5 Conclusions

In this section, briefly summarize your paper — what problem did you start out to study, and what did you find? What is the key result / take-away message? It's also traditional to suggest one or two avenues for further work, but this is optional.

6 Contributions

Due to the large volume of coding in this project, the authors worked separately to develop moths of the classes and methods. Andrew wrote most of the functionality of the Tree, including the crossover, mutate, evaluation, and calculation of fitness methods. He also wrote functions to read the supplied data into a usable format. Michael completed the basic implementation of the Tree that Andrew began, as well as the underlying Node class. Michael wrote the driver program to create new generations, move through generations, as well as the tournament selection method. Each partner researched the parts of the program the implemented.

References

Blickle, T., and Thiele, L. 1994. Genetic programming and redundancy. *choice* 1000:2.

Ferreira, C., and Gepsoft, U. 2008. What is gene expression programming.

Langdon, W. B. 2000. Size fair and homologous tree crossovers for tree genetic programming. *Genetic programming and evolvable machines* 1(1-2):95–119.

Poli, R. 2003. A simple but theoretically-motivated method to control bloat in genetic programming. In *European Conference on Genetic Programming*, 204–217. Springer.