Symbolic Regression Using Genetic Programming

Jenny Zhong and **Dimitrios Chavouzis**
{yuzhong,dichavouzis}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

This paper explores genetic programming in the context of symbolic regression. We attempt to emulate the functions given by two datasets of x and $f(x)$ values with 25000 data entries each. The goal is to identify the best-fit to each given datapoints function. Our implementation involves generating random functions, determining their fit, and then modifying them until we produce a function close to the mystery function.

1 Introduction

Symbolic regression is a popular application of genetic programming. The symbolic regression of a polynomial problem consists of finding a way to connect x and $f(x)$ values with a function that produces results which align with the mystery function. The dataset is given in a CSV-file of the following format:

	x	f(x)	x1	x2	x3	f(x1,x2,x3)
2	-0.66	18.37	9609691	2256113	3.2	141.56
3	2.99	5	9383433	8226381	3.33	465.22
4	2.65	5.12	7537404	128622	1.01	63.79
5	-1.72	27.31	2501726	2662524	8.06	6.84
6	4.33	6.76	2020688	5365951	4.85	30.78
7	1.94	6.13	727650	8965739	0.31	4613.66
8	-1.02	21.17	8372570	8319333	6.65	105.05
9	0.05	13.73	3865646	3830570	6.8	21.35
10	-1.77	27.78	223434	8473848	4.6	5.98
11	1.06	8.76	1743763	686496	8.55	1.09
12	2.25	5.56	6782237	2961672	4.06	81.31
13	-2.26	32.66	5716946	4429665	8.92	21.25
14	3.55	5.3	2045348	8910762	0.95	1354.25
15	-2.85	39.19	3699192	6490698	0.73	2981.42
16	-0.56	17.69	9793204	8777094	2.13	1266.45
17	3.69	5.47	6877284	2290667	6.34	26.13
18	-4.2	56.83	2066352	1673647	8.13	3.49
19	-1.67	26.77	940006	8347333	9.04	6.41
20	-0.54	17.53	4560160	1379963	5.51	13.83

Figure 1: Samples from given Dataset 1 (left) and Dataset 2 (right), on which the experiment was run.

Similar problems have been described and analyzed by many computer scientists. In our implementation, we followed the steps given by Dr John Koza in his tutorial "Example of a Symbolic of Genetic Programing (Symbolic Regression of a Quadratic Polynomial)" (Koza 2003). In this tutorial, Dr Koza explains how we should set up the program, defines generation, crossover, mutation and fitness, and suggests how to use these concepts for optimal results. The problems tackled in this tutorial include:

I: Individual representation of our items in the population. Trees including nodes of constants, variables and arithmetic signs are used to formulate a function.

II: Selection of individuals according to fitness value and some probabilistic approaches (for further information please look at *Background* and *Experiments*).

III: Introduction of reproduction, mutation and crossover, the randomly applied manipulations of the population and the functions, in order to create the population for the next generation (for further information look at *Background*).

The remainder of this paper will provide further background information on some of the techniques used in our research, describe our experimental setup, and finally, report and analyze our results.

2 Background

Our symbolic regression approach involves generating random functions, evaluating them, and manipulating our functions in order to proceed to the next generation of functions until we reach the generations limit. Each generation is an iteration of our program and consists of the initial population and the manipulations applied to it. To randomly manipulate the populations, we implemented a *crossover* (C) and a *mutation* (M) function:

C: The crossover operation is done between two trees in our population. For each tree, a random part of the function is selected and swapped with a random part of the other function. The result of that manipulation is a new offspring used and evaluated in the next generation. We perform crossovers on 90% of the trees in our current population (Koza 2003).

M: The mutation operation is done by randomly selecting a function in our current generation and manipulating it by changing one of its terminal nodes. These changes include altering constants or changing them to variables. We perform mutations on 1% of the functions in our current population (Koza 2003).

Besides manipulating the existing population, we also move some of the trees of the current population to the next generation using the *reproduction* (R) operation.

R: The reproduction operation copies 9% of the current population's trees and adds them to the new generation by utilizing the probabilistic tournament selection algorithm explained in the *Experiments* section (Koza 2003).

Our algorithm also uses a *fitness* function to evaluate the randomly generated and manipulated functions, in order to determine the best-fit. Our fitness function uses the *Mean Squared Error* approach and can be described as following:

F: For each x value from our dataset, we calculate the $f(x)'$ from our function, subtract it from its actual $f(x)$ value in our dataset, square the result and add it to the tree's general fitness value. We then divide the summation of all the fitness values for each x by the number of data entries (n). The formula for determining the mean squared error is given below:

$$F.1\text{: } (\tfrac{1}{n}) \sum_{i=1}^{n} (f(x)_i - f(x)'_i)^2$$

3 Experiments

This experiment is meant to explore genetic programming and one of its most popular fields, symbolic regression. The goal is to find a function that emulates the polynomial function given by each dataset. To that end, we decided to create an initial population of randomly generated functions represented as trees. Each of these trees had to include an x value, some arithmetic symbols and some constants. To ensure that our trees are random, each node is assigned a random value or symbol. We randomly picked a depth of three or four for every tree.

For each tree in our population, we calculate its fitness value using the formula *F.1*. We use that value as the measure to rank each tree in the population from most to least fit in order to later run the tournament selection on them as well as the mutations and the crossovers to be performed

In order to find the best-fit function for our dataset, we manipulate the current population and proceed to the next generation. We implement a tournament based approach to select the individuals for our next generation (Koza 2003). Our approach is probabilistic: trees with lower fitness values have a higher probability of being selected due to their better-fit nature. However, not only the best-fit individuals are chosen since even the less-fit functions have some probability of being selected.

Between each generation, we apply the modifications defined in the previous section to randomly selected trees in our current population. By performing random mutations and crossovers we create offsprings that will be used in our new generation and will be later evaluated as possible solutions to our given problem.

Another important part of the experiment is deciding on the size of the population as well as the number of generations created. From our experience, the population should be somewhat large for the manipulations to be random. If we consider a small population, crossover between the same function on different generations could produce duplicate results, making our program less efficient. We ran various experiments with different sizes and number of iterations in order to determine the optimal size and generation number that produces the best results as quickly as possible. Keeping efficiency and time in mind, we decided not to experiment with very high values for population and number of iteration since such a choice would be very time-consuming and wouldn't significantly affect the quality of our results, as explained in the *Results* section.

Lastly, to ensure the success of our experiment, we had to tackle a possible overfitting problem. Overfitting is a problem that occurs when the program is trained very-well and in detail over a specific part of the data but when new data is added, the model can not emulate or predict the results anymore. Throughout our research, we found many different approaches to prevent overfitting in symbolic regression. We decided to use the "Data Splitting" approach, which suggests that the program should be trained over 80% of the data and then get tested over the rest of it. (Vipul K. Dabhi 2012)

4 Results

For each of the datasets we performed multiple experiments of different population and iteration values and compared the results with each other and with the function implied by the data given to us.

Dataset 1

Figure 2 provides the three best-fit functions that came up during our experiment for the first dataset. As shown by the table, the three best functions have overall fitness around 1.0, with the first one being as close as 0.00062.

Evaluation		Parameters		
Function	Fitness	Pop.	Gen.	Tourn. Size
$x^2 - 6x + 14$	0.00062	100	20	5
$x^2 - 6x + 13$	1.00018	100	15	5
$x^2 - 6x + 15$	1.00092	100	20	5

Figure 2: The 3 best-best fit functions for Dataset 1. Pop. refers to the number of trees per generation and Gen. to the number of generations.

In order to illustrate how close our best solution emulates the function of the given data, we decided to graph it using the Desmos Graphing Calculator. To calculate the actual function, we used most of the given data and graphed Excel's trendline function (Figure 3). Notice that the two functions are identical. We consider the experiment for Dataset

1 a success since $x^2-6x+14$ successfuly emulates the given mystery function's results as shown in Figure 3. Of course, we had to make sure that our function does not encounter the problem of overfitting. When tested with the remaining 20% of the dataset, the fitness value was 0.00013, which is very close to 0.00062 (our function's fitness value). Thus, we conclude that our model can accurately emulate the data without overfitting.

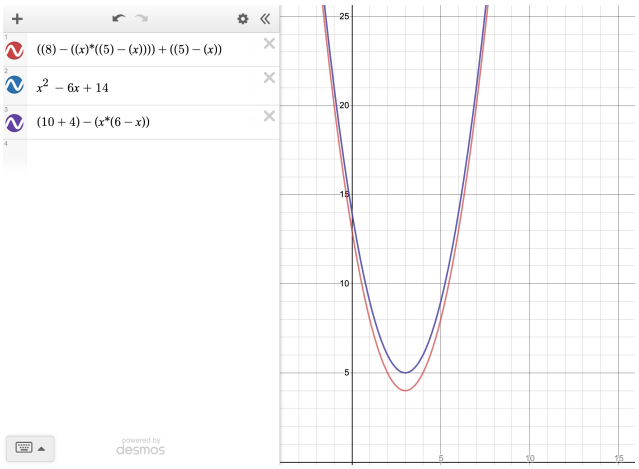

Figure 3: Desmos software comparison of given data's function approximation (blue) and the best fit functions that emulates it (purple and red). Notice that you can only see two lines, that's because the purple one is the exact same with the mystery function.

Dataset 2

Figure 4 provides the best fit functions that were obtained throughout the experiment for Dataset 2. The fitness values for those functions are much higher than those of the previous experiment (see *Figure 2*), which is the result of the much higher $x1$,$x2$, and $x3$ values in Dataset 2 compared to those of Dataset 1 (see *Figure 1*). Refer to the fitness formula described in the *Background* section; extremely high x values could possibly result in a big $f(x)'$ value, which when subtracted by the dataset's $f(x)$ and then squared, could result in such large fitness values.

Function	Fitness
$(301.25*x3+15.06*x1)/x2$	33442193643775.445
$y*(-x+3.43154)+(z^2+yz+xz+2yx)/(x+z)$	$1.0333081477340067e+18$

Figure 4: The top 2 best-fit functions for Dataset 2. All results of an experiment with population of size 200 and 60 generations. The numbers have been rounded for better readability.

While the fitness values might seem high, we don't consider the experiment unsuccessful. Our program started with fitness values around $1.0335047979400997e+18$ and ended up with values as low as 33442193643775.445 over 60 generations. In other words, throughout the random mutations and crossovers in every generation, we managed to achieve a fitness value lower by $1.0334991e+18$ overall.

$$F(x1,x2,x3): \frac{301.249619*x3+15.062481*x1}{x2}$$

Function 2: This is the simplified version of the best-fit function for Dataset 2.

5 Conclusions

In this paper, we presented our approach to a symbolic regression problem. Our experiment aimed to find functions that emulate the polynomial function given by each of our datasets. We successfully managed to use genetic programming to identify the function for Dataset 1 and our model managed to approach a solution for Dataset 2. While the resulting function for Dataset 2 does not emulate the "mystery function" in detail, it improves as the number of generations increases. We believe that given more generations, our algorithm could have made even more progress.

We also noticed that the success of such an algorithm doesn't necessarily rely on increasing the population size and the number of iterations. While increasing those variables up to a point is more likely helpful, using high values for them could make the program much slower without guaranteeing that the result will be of better quality. Due to the probabilistic nature of the algorithm, it is likely that we will get an equally fit result in iteration 15 to the one we would get if we had a few more iterations. This is illustrated by our results that vary in values for those two variables. Thus, we conclude that it is important to find a balance between the accuracy needed and the time consumption of the program to achieve satisfactory results in the least time possible. It is also important to state that throughout our experiments we noticed that a balance between the population size and the tournament size is essential. The bigger the tournament size is, the more likely is that the best values will be picked which can result in "eliminating" the probabilistic aspect of our approach which is crucial for the effectiveness of the algorithm.

6 Contributions

Jenny and Dimitrios collaborated on the entire project. We equally contributed on the planning and implementation of the project, either by meeting via Zoom and sharing ideas or by working individually and then sharing parts of the code with each other. The paper was written and revised by both of us and we believe that we contributed equally to the whole project. Overall, we collaborated throughout the whole process and we worked well together.

7 Acknowledgements

We want to acknowledge Dr. John Koza, computer scientist and former adjunct professor at Stanford University for his tutorial "Example of a Run of Genetic Programming (Symbolic Regression of a Quadratic Polynomial)" which

was the inspiration for our implementation. We also want to acknowledge Vipul K. Dabhi for his paper on overfitting "A survey on techniques of improving generalization ability of genetic programming solutions" which provided us with possible solutions on our problem. Lastly, we want to thank Desmos for their free online graphing software that helped us picture our functions and present them on this paper.

References

Koza, J. R. 2003. *Example of a Run of Genetic Programming (Symbolic Regression of a Quadratic Polynomial)* . Pearson Education.

Vipul K. Dabhi, S. C. 2012. A survey on techniques of improving generalization ability of genetic programming solutions. *Information Technology Department, Dharmsinh Desai University, Nadiad, INDIA* 2.