Using Genetic Programming (GP) to Implement Symbolic Regression

Bryce Anthony and **Joseph Richards**
{branthony,jorichards}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

This paper explores effectiveness of using genetic programming for the task of performing a symbolic regression on a data set. We create a test-train split of each data set to combat overfitting and our genetic program implements tournament selection to create the next generation of trees. We test our genetic program on two data sets with $25,000$ data points; one with only one independent variable and the other with three independent variables. Although our program was pretty effective at producing the equation for the simpler data set, it was significantly less accurate on the more complicated, 3 independent variable data set.

1 Introduction

Performing a regression is one of the most common supervised learning tasks in the field of machine learning. Regressions allow models to make predictions about a specific data point by using a function obtained from fitting a data set. This process of fitting a function to a data set involves minimizing the error between the dependent variable and the output of the function for each data point in the data set. Regressions are done with the assumption that the regression function will generalize well to similar data, but obtaining a function that generalizes well to other data is one of the most difficult parts of machine learning. Additionaly, the ability of a model to generalize well to other data is one of the main sources of variation between models built for the same task. A Regression function that perfectly fits the data your model is trained on may yield a model that's not very applicable to new data, but a function that does not fit the training data well enough would just create a bad model.

Genetic Programming is a subset of machine learning, that uses a process comparable to biological evolution to additively build a function to fit some data. Unlike traditional machine learning algorithms, genetic programming performs symbolic regression which defines the relationships between variables without making assumptions about the data. Because the function produced from genetic programming is solely based on the data used by the genetic programming algorithm, theoretically, genetic programming could be a uniquely effective method of creating a predictive model.

In this paper we use genetic programming to perform symbolic regressions on 2 data sets. Our genetic programming algorithm implements a tournament selection algorithm from (Fang and Li 2010) to create subsequent generations and we use a test train split to choose our best regression for each dataset. One dataset had 25,000 observations and one dependent variable and the other had 25,000 observations and 3 independent variables. The setup for our genetic programming algorithm is explored more in the abstract and our testing and training processes are explained in the experiments and results section of the paper. Finally, in the conclusion of this paper, we discuss the broader impacts of this work incorporating the work of (Ferreira, Guimarães, and Silva 2020) to discuss genetic programming and its potential to improve the interpretability of machine learning models.

2 Background

Our genetic program was implemented based on the information provided on `https://www.genetic-programming.com` and we incorporated the tournament selection method from (Fang and Li 2010). This method uses a subset of the current generation of trees and picks the tree with the best fitness to be mutated, reproduced, or for a pair of trees to be "crossed over" into a new tree. Each newly created tree was then saved into a new array so that these newly generated trees wouldn't affect the "evolution" of the previous generation. The fitness of each tree was determined by calculating the mean squared error of the tree's equation on some data. To combat overfitting we created both a test and train portion of each dataset. the training portion of each dataset was created by reserving $\frac{2}{3}$'s of the data in each dataset and the testing portion of each dataset was created reserving the other $\frac{1}{3}$ of each dataset. while, for both of the datasets, the training portion of was used to create several regression functions, after these functions had all been created, the testing portions of the datasets were used to combat overfitting by choosing the regression function that was most representative of each of the datasets.

3 Experiments

The experimental setup was the same for the two datasets we performed regressions on. Using the $\frac{2}{3}$'s of the dataset reserved for testing, we ran our genetic program 10 times to obtain 10 different regression functions. Each function was represented as a tree where the parent nodes were one of the 4 basic mathematical operators (+, -, ÷, x) and the leaf nodes were each an "X" and constant being either (-5, -4, -2.3, -3, -2, -1, 1, 1.4, 2, 3, 3.7, 4, 5). These trees varied in size and the equations of each tree could be obtained by starting at the parents of the leaf nodes and creating expressions using the child nodes and parent node as the operator. (See Figure 1)

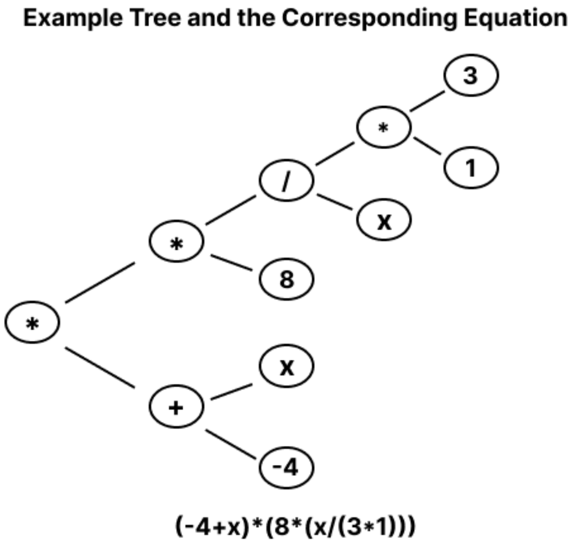

Figure 1: On the reading of equations from trees.

For each of the 10 runs of our genetic program, our initial population consisted of 30, depth 2, trees that were randomly generated from our specified constants and operators. The fitness of each of the trees was evaluated, and the next generation of trees was created using the tournament selection method from (Fang and Li 2010). For our implementation of the tournament selection method, we chose random subsets of 4 trees from the current generation; selecting the function with the lowest fitness value for mutation 2 % of the time, the one with the best fitness value for reproduction 90% of the time, and the two trees with the lowest fitness values were selected for crossover the other 8% of the time. This creation of a new generation or an "evolution" was done until there was an equation with a fitness that was less than 6, to help avoid overfitting, or after 30 evolution's had been performed.

After 10 trees were created from the training process, we chose or most representative tree by evaluating the fitness of each of the 10 trees using the test dataset and the tree with the lowest fitness value was output by our program as it was the most representative of the entire dataset.

4 Results

For the Data Set 1, which only had one independent variable, our best equation had a mean squared error of 1.0 on the test dataset. This was much better than most of the equations produced from the training procedure as the worst performing equation had a fitness value of 65.17 and the average fitness value of the 10 equations was 34.73. In terms of the complexity of the functions or or depth of trees our average depth was 5.4 and the depth of our best performing model was only 4 meaning it wasn't the most complex of all the equations from the training procedure. The equation that our best tree had was $x^2 - 6x + 13$ which is very similar to the equation that google sheets gaves us, $x^2 - 6x + 14$, when we inputted our data and created a trendline.

Table 1: Data Set Summary

Metrics	**Data Set 1**	**Data Set 2**
Max Depth	8	19
Min Depth	4	11
Avg Depth	5.4	14.67
Depth of Best Tree	4	14
Worst Fitness	65.17	1.58×10^{17}
Best Fitness	1.00	3.92×10^{12}
Avg Fitness	34.73	1.27×10^{14}

For Data Set 2, which had 3 independent variables, our best performing model had a mean squared error of 3,918,850,481,498.528. The average error of the 10 equations from the training process had an average error of $1.27294272 * 10^{14}$, and the equation with the worst error was 158,194,148,155,203.84. In terms of the complexity of the equations produced for the second dataset, the smallest depth of the trees from the training process was 11, the max depth was 19, and the averaged depth was 14.66. Unlike the first dataset, the best performing equation tree wasn't the one with the smallest depth but rather an equation tree with a depth of 14 which was closer to the average of the trees produced from the training procedure.

Ultimately our experiments highlight the effectiveness of our overfitting procedure described in the HW2 handout as the testing process described in the handout allowed us to pick from several equation trees produced from the training procedure rather than simply using the first equation tree produced. This is important because of the differences in the fitness/error of the 10 equations produced from the training process. The overfitting procedure can be attributed to the selection of the tree with an error of 1 rather then the tree with an error of 65.17 for the first dataset and for the selection of a tree with an error of 3,918,850,481,498.528 rather than 158,194,148,155,203.84 for the second dataset. Another area of our program that likely affected our final results was our implementation of the tournament selection algorithm. One shortcoming of the Tournament Selection Implementation is that the algorithm could pick a tree more than once since they are randomly selected. This means some trees are competing with themselves and other trees are not selected at all which does not lend itself to a diverse tournament. Ultimately this flaw in tournament selection can produce less diverse trees as the entire population isn't being considered in the production of new trees, the potential of trees to not be selected is referred to as the the Not

Sampled Issue in (Fang and Li 2010). Another area for further research would be the implementation of bloat control to ensure that our trees don't increase in size too much (or functions don't get too complex) This may have improved the results of data set 2 since we have fairly deep trees that also have large fitness values.

Despite the inadequacy of our performance on the second data set, the interpretability of genetic programming is on of its strengths. Due to the additive and simplistic way genetic programming performs a symbolic regression, extensions of genetic programming have been used to help reason about the decision making done by blackbox AI algorithms (Ferreira, Guimarães, and Silva 2020). propose a genetic programming explainer (GPX) algorithm that can explain the local behavior of more complex models for regression and classification problems. They found that their GPX algorithm was able to more accurately explain the decisions made by these black box models more accurately than decision trees and the LIME algorithm from (Ribeiro, Singh, and Guestrin 2016).

5 Conclusion

This paper explored the effectiveness of symbolic regression produced from genetic programs. We performed a test-train split on our dataset and used a tournament selection algorithm to create subsequent generations within our genetic programming algorithm. This method proved to be extremely effective for the first dataset which only had one independent variable but significanlty worse for our more complicated 3 independent variable dataset. We found that the test train split played a significant role in reducing the error of our model but that a method similar to bloating should be implemented to help reduce the complexity of our equations and reduce error. We found genetic programming for symbolic regression to be a promising method for performing a regression but conclude that the real potential of genetic programming is that it can be used to increase the comprehensibility, explainability, transparency, and functionality of blackbox AI models (Ferreira, Guimarães, and Silva 2020).

6 Contributions

J.R wrote the code for functions: bloatcontrol, runTournament, insertx, and collected experiment data for both datasets along with the abstract section. J.R and B.A wrote the code for functions: evolution, mutation, copy, both evaluates, maketree, constructor, setuptreegen, both fitness functions, and completed the experiment section together. B.A wrote the code for functions: printInOrder and crossover, and completed the introduction, background, results, and conclusion sections.

References

Fang, Y., and Li, J. 2010. A review of tournament selection in genetic programming. In Cai, Z.; Hu, C.; Kang, Z.; and Liu, Y., eds., *ISICA 2010*, volume 6382 of *Lecture Notes in Computer Science*, 181–192. Springer.

Ferreira, L. A.; Guimarães, F. G.; and Silva, R. 2020. Applying genetic programming to improve interpretability in machine learning models.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, 1135–1144. New York, NY, USA: Association for Computing Machinery.