Asn 2: Building the Black Box

Ethan Heavey
St. Francis Xavier University
Department of Computer Science
Antigonish, Nova Scotia, Canada
x2018zbf@stfx.ca

***Abstract*—The symbolic regression is widely viewed as a "toy" problem for genetic programming. In this report, it will be explored how to use python's DEAP library to implement both a typed and non-typed genetic programming system. Plotting the results, it will be shown how the system is relatively competent at discovering functions for low-dimensionality data, but once the number of inputs increases it becomes increasingly difficult to search for an accurate solution. A dimensionality reduction method, TSNE, was also employed in an attempt to gain a better understanding of the high-dimensionality data solutions.**

***Index Terms*—Evolutionary Computation; Symbolic Regression; Genetic Algorithm; Genetic Programming;**

I. Introduction

A type of regression analysis, symbolic regression searches the space of possible mathematical expressions in order to determine the optimal solution to a given dataset. Using decision trees, a genetic programming (GP) system implemented using Distributed Evolutionary Algorithms in Python (DEAP)[2] has been developed to take the input of a .csv file with an arbitrary number of columns, and return an interpretable string representing the function the model found to best fit all the data provided, or x in the function below. The final data column represents y. Symbolic regression finds the best f for x and y.

$$f(x) = y$$

A major difference between symbolic and linear regression is that symbolic regression avoids any bias and assumptions about the model, whereas linear (and other classical forms of regression) seek to optimize the parameters for a pre-specified model structure. However, a major disadvantage of this is that there is a significantly larger search space, as there is not only an infinitely large search space, there are an infinite number of models than can accurately fit to finite data. The advantage of this, despite symbolic regression theoretically taking longer to come to a conclusion for a particular dataset, is that the solution provided will be the best of all worlds, in that it attempts to discover the best model structure *and* parameters for a finite dataset.

A typed GP system is designed to be a more generalized version of a regular GP system, with specific types associated with the input and output variables of all aspects within the system.

Ethan Heavey $00.00 ©2019

In the past, performing linear regression on the arbitrary outputs of symbolic regression has empirically been found to provide significant benefits[3]. The reference method does not introduce additional parameters to the symbolic regression run, is guaranteed to improve results on most problems, is not worse on others, and has a well-defined upper bound on the error, the scaled squared error spoken about in the article becomes an ideal candidate for the standard error measure for symbolic regression.

II. Data

The data used for this model was provided by Dr. James Hughes in our assignment handout package. All data is in a .csv file format, containing a number of columns where, all columns except the final one are the inputs values for a given function *f*. The final column represents the output from *f(x)*, or *y*.

For the typed GP, data was acquired from the UCI Machine Learning Repository[1][1]. The csv files used in the typed GP problem contain records of different types of red wine. The system will attempt to derive a function to predict the quality of a given wine, as an integer, from ten floating point values. In this scenario, the ten inputs are different measurements taken from a given wine sample[2], represented as x in the equation referenced in I. The integer value "quality" represents the y value in the function previously mentioned. The dataset contains eleven variables and 1599 records.

A. The Problem

Symbolic Regression is a widely used toy problem for GP systems. There is no starting point provided for the algorithm, which leads to some very interesting and unique results from the system. The challenge presented is to develop a GP system that can generalize across a multitude of input variables, and return an accurate-as-possible function. Given that each data file was generated by a specific function, it will be interesting to see if the system can develop solutions akin to the original functions.

III. Algorithm

The implemented GP system was built using the DEAP python library, and tested used the data outlined in II.

[1]http://archive.ics.uci.edu/ml/index.php

[2]fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol

A. *Parameters*

In developing a GP system, there are a few parameters that will be necessary to set and tweak to yield an ideal function. First things first, all parameters are significant, and fine-tuning a single parameter will be a tedious task, as once the value of one parameter changes, the optimal value of all other parameters shifts to create a new optimal permutation of values. The one parameter that has minor leeway regarding this permutation change is the *MAX_GENERATIONS* parameter. Once this parameter is set to an sufficiently large value, and the genetic algorithm has converged on a point in the problems state space, this parameter has minimal effect. However, too small of a value for *MAX_GENERATIONS* could result in premature convergence, or no convergence at all. Due to hardware limitations this value has been set to s static 500. There are once again the mutation rate (*MR*) and crossover rate (*CR*) parameters, similar to those in a genetic algorithm implementation, that will have their affect on the results of the GP system tested.

The effect of these parameters is difficult to visualize outside of simply showing the functions generated. For a *CR* of 0.8, a *MR* of 0.1, and a *MAX_GENERATIONS* of 500, the GP presented $4.4991079590881558 * a + 7.2658365037360899 - 0.49910795908815575 * sin(a)/a + sin(2 * a^2 - 0.49910795908815575)/a + 0.49821750965028831/a$ as a solution. Clearly this is a linear line, and the plot of the function Fig 1 shows a near-perfect fit to the given data (represented as blue points), but this function can be vastly simplified. In Fig 2, it can be seen that even though the line is almost identical to the previous, over complicated function, $4.5045452315020379 * a + 7.238608053429207$ is a *lot* easier on the eyes.

The parameters for the typed GP problem were almost identical to the non-typed version, with one significant change. The *NUM_GEN* was decreased from 500 to 100 due to the size of the dataset.

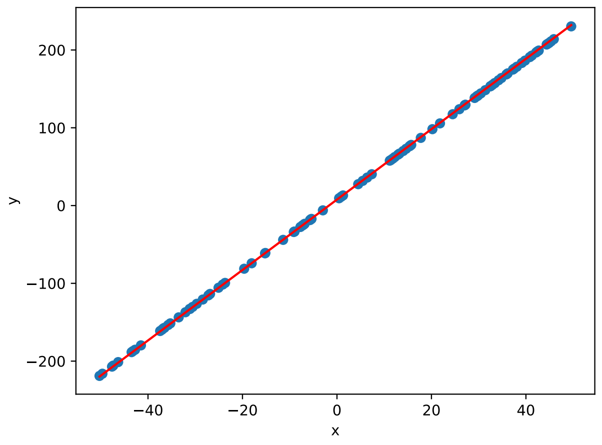

Fig. 1: D1 Data - Non-simplified Solution

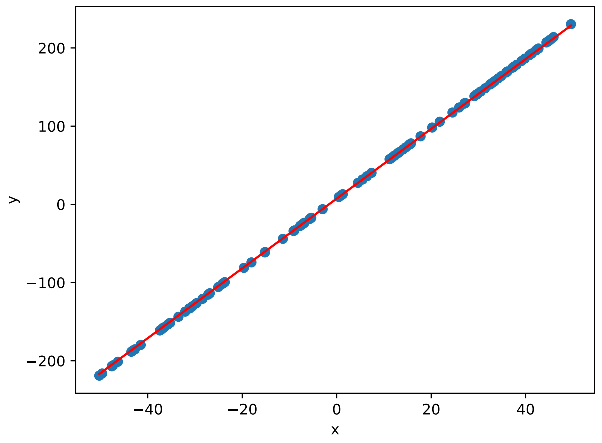

Fig. 2: D1 Data - Simplified Solution

B. *Structure*

The GP system was designed used the interface provided by DEAP, but still follows the general structure outlined by Dr. Hughes in lecture. The chromosomes are encoded as decision trees, with the genes represented by the nodes in the decision trees. The genes can range from real numbers to mathematical operators which culminate in a mathematical expression. Using python's Sympy[3] library, and a helpful Stack Overflow page[4], the final population's best individual was selected, and converted into a readable string. This string was then coupled with python's *eval* method in order to evaluate the accuracy of the individual in comparison to the original data.

[3]https://www.sympy.org/en/index.html

[4]https://stackoverflow.com/questions/44127475/human-readable-output-with-deap

C. *Modifications*

A modification added to the GP system was the removal of floating point values in favour of only using integer values in each solution determined by the system. For example, using the data file *d1.csv*, an alternate solution to the function presented above is $9*x/2+7$, where $9*x/2$ can be simplified to $9/2*x$, and simplified even further to $4.5x$. GP systems are smart enough to identify that, even with constrains on the type of numbers that can be used, integers can make floats by dividing one with another.

A second modification was the implementation of an elitism and tournament based selection approach to choosing a new population. The e individuals with the lowest mean-squared error will be carried over into the next population, with the rest of the population entered into a tournament selection, where *SELECTION_SIZE*$-e$ individuals will be selected. This process will repeat until the population is full.

The third and final modification to the GP system was the algorithm used to perform the regression. By default, DEAP uses the *eaSimple* algorithm to perform the evolution. In comparing the two evolutionary algorithms, eaSimple was observed to have a lower mean mean-squared error (MSE) by the best individual after the evolution had concluded, as can be seen in table I. I believe the reason for this is due to the method in which each function selects a new population, eaSimple selects offspring and does a generational update to the population, whereas Mu + Lambda selects offspring, and then selects a new population from the old population and the new offspring. This could lead to a large amount of members from the old population being carried forward and having little effect on the new population.

TABLE I: This table depicts the mean mean-squared error of the Simple and Mu+Lambda evolutionary algorithms.

Algorithm	*MR*	*CR*	Max Generations	Mean MSE
Simple	0.1	0.8	500	1008.88
Mu + Lambda	0.1	0.8	500	6758.87

IV. Analysis Methodology

In the analysis of the performance of the GP system, visualizations were developed to view the proposed function for a given dataset. As referenced in III-C, by finding the mean of the MSE values over thirty runs of each algorithm, it was determined that results from both algorithms are statistically significant. The reasoning behind the use of MSE to evaluate the proverbial fitness of the chromosomes within the GP system was to punish the incorrect values with a larger difference from the true value more than those who are closer to the true value.

V. Results

As expected, the GP system successfully performed symbolic regression on all presented data. Before we get into the visualizations, the following plots can be read as follows:

- Dots: Data points from the original data file
- Line/Plane: the function generated through symbolic regression

Initial plots before tweaking the GP parameters were subpar at best, finding functions that barely fit to the data (Fig 3). As parameters were tweaked and fine tuned, along with the addition of the Mu + Lambda evolutionary algorithm, the results were beginning to look more promising. As mentioned in III-A, complex functions can be simplified greatly for what appears to be near-identical results. It was incredible to see how the simple editing of parameters had such a drastic effect on the plots. As seen in Figs 4 and 5, the complex and simplified solutions still provide incredible similarity in their plots.

Other interesting figures were that of files *d3* and *d5*. As can be seen in Fig 6, the plot is approximately equivalent to the function provided in the caption, with a simple, linear plane across the space. In contrast, Fig 7's plot appears to be

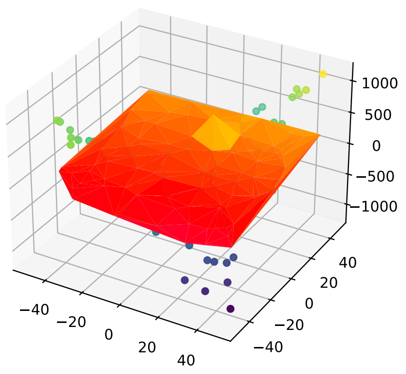

Fig. 3: D4 Data - Inaccurate Solution

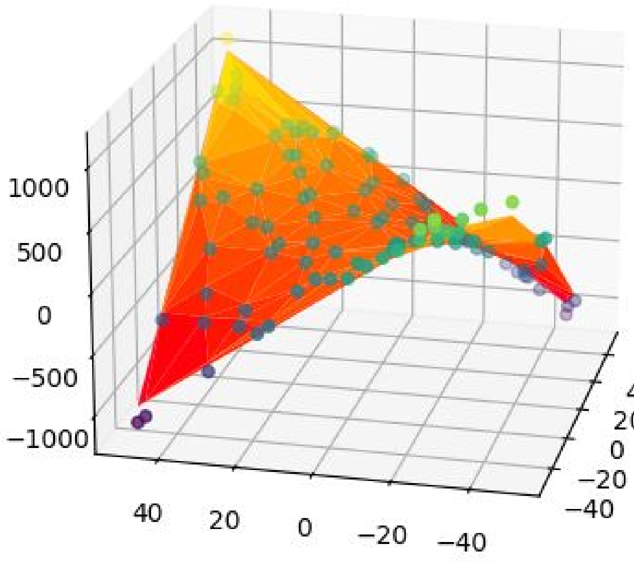

Fig. 4: D4 Data - Non-simplified solution

wild and noisy, taking on the function $sin(a)$ as shown in the caption. These results are interesting as even though the system was presented with three-dimensional data, it still chose to omit one of the variables as it was deemed irrelevant to the final product. Whether these omitted variables were noise or not, it was a unique result either way.

In an attempt to plot the thirty-dimensional *d7* data, sklearn's T-distributed Stochastic Neighbor Embedding (TSNE[5]) dimensionality reduction method was employed. Each row of the csv file (save the final, "output" row) was

[5]https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

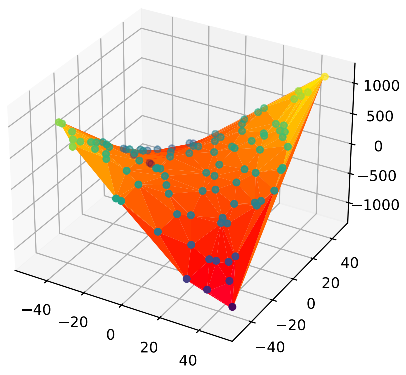

Fig. 5: D4 Data - Simplified Solution

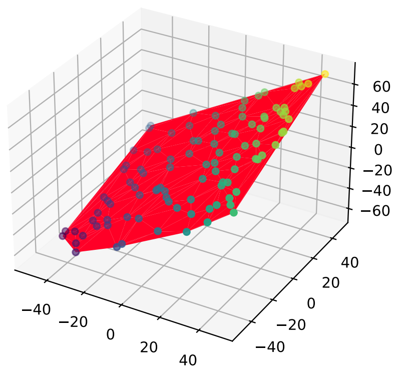

Fig. 6: D3 Data - $1.1011480278899446 * a^2$

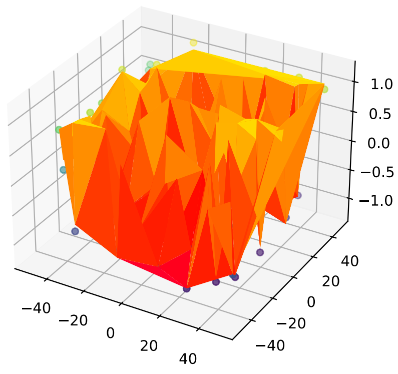

Fig. 7: D5 Data - $sin(a)$

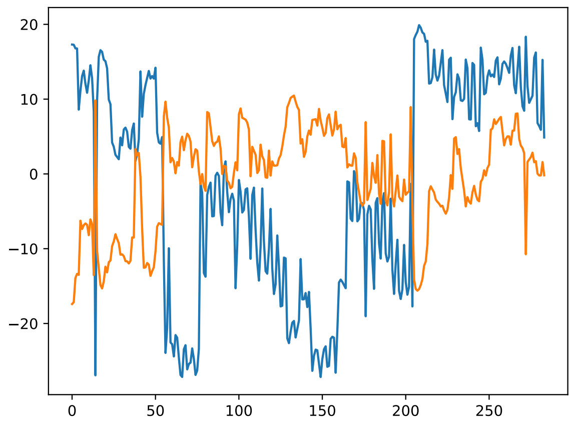

Fig. 8: D7 Data - TSNE Plot

evaluated in the (very lengthy) expression[6] presented by the GP system. There are two interesting points of observation, with the first being the amount of repeated terms. Secondly, it only uses *ARG10*, *ARG11*, *ARG12*, *ARG14*, *ARG16*, and *ARG27*, which is a mere six of the *twenty-nine* initial variables! In Fig 8 it can be seen that, even though the function is far from perfect, in certain ranges, it follows a similar trend to the original data. Another unique aspect of the plot is how for certain values on the x axis, the values on the y axis appear to mirror each other, before converging once more.

[6]Yeah this is a long one... $ARG27 - (ARG16 - -(ARG27 - -(ARG16 - -(ARG16 - -(ARG16 - -(ARG14)) - -(ARG16 - -(ARG16)--(ARG27))))--(ARG27--(ARG16--(ARG16)-2* -(ARG27)))--(ARG16-3*-(ARG16)--(ARG16--(ARG16))- -(ARG16--(ARG16--(ARG16)--(ARG16--(ARG27))))- -(ARG16--(ARG16)--(ARG27))))--(ARG27--(ARG16- -(ARG27--(ARG16)--(ARG27)))--(ARG16--(ARG16- -(ARG16)--(ARG16-2*-(ARG16)))--(ARG16--(ARG16)- -(ARG16--(ARG16))--(ARG16--(ARG16)--(ARG27))))- -(ARG27--(ARG10--(ARG16--(ARG16)))--(ARG16- -(ARG16--(ARG16)))--(ARG16--(ARG27)--(ARG16- -(ARG14)))--(ARG27--(ARG27--(ARG27--(ARG16)))- -(ARG16 - 2 * -(ARG16) - -(ARG16 - -(ARG16)))))) - -(ARG16--(ARG16)--(ARG16--(ARG16--(ARG16)- -(ARG16--(ARG27))))--(ARG27--(ARG16--(ARG16)))- -(ARG16-2*-(ARG16)--(ARG27)))--(ARG16--(ARG16)- -(ARG16--(ARG16))--(ARG16--(ARG16--(ARG16)))- -(ARG16--(ARG27-2*-(ARG16)))--(ARG27-2*-(ARG27- -(ARG16))--(ARG16--(ARG16)--(ARG27--(ARG16))))- -(ARG12 + ARG5 - -(ARG16 - -(ARG16) - -(ARG27)) - cos(ARG26)))--(ARG16--(ARG16)--(ARG14--(ARG16- -(ARG10)--(ARG16)))--(ARG16--(ARG16--(ARG16)))- -(ARG27--(ARG27--(ARG27--(ARG16))--(ARG27- -(ARG16--(ARG27)--(ARG16--(ARG27))))))--(ARG16- -(ARG27)--(ARG27--(ARG16)))--(ARG16--(ARG16- -(ARG27--(ARG16)))--(ARG16--(ARG27)--(ARG16- -(ARG14))--(ARG16--(ARG16))))--(ARG27--(ARG16)- -(ARG16--(ARG16)--(ARG27))--(ARG16--(ARG16)- -(ARG27)--(ARG16--(ARG16))))--(ARG27--(ARG16- -(ARG16--(ARG16--(ARG16))))--(ARG27--(ARG16))- -(ARG27--(ARG16--(ARG16)))--(ARG16--(ARG16)- -(ARG27))--(ARG16--(ARG11)--(ARG16)--(ARG16- -(ARG27--(ARG16)))))))^{(}1/4)$

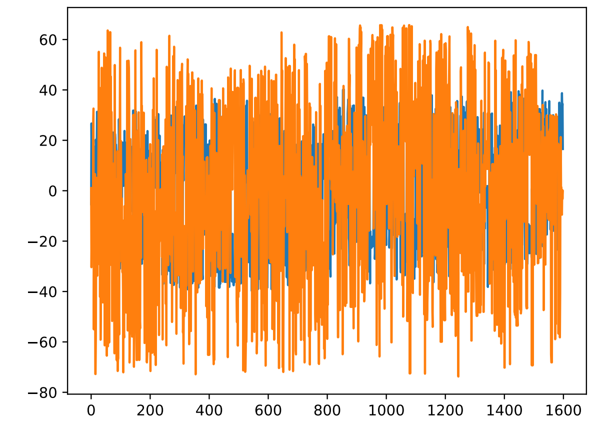

Fig. 9: Wine Data - Line Plot

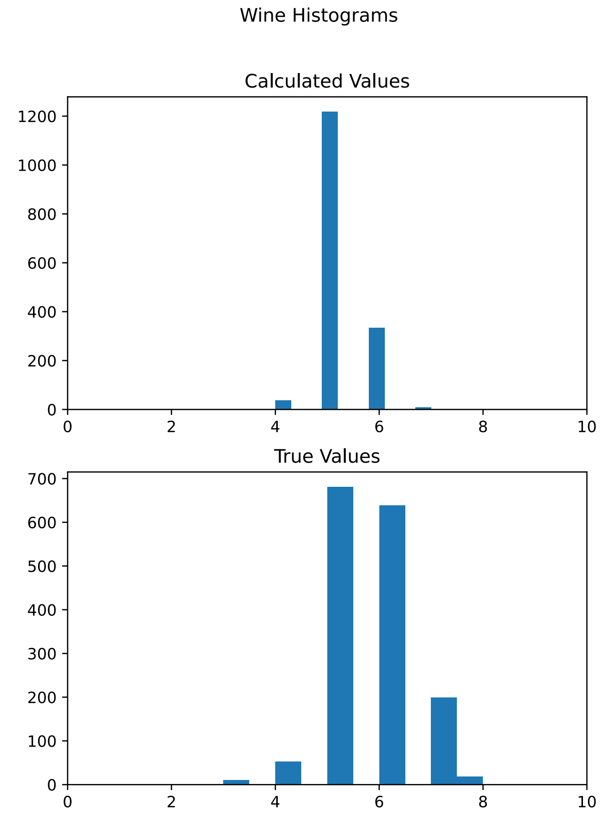

Fig. 10: Wine Data - Histogram - $(ARG10 + 2 * ARG8 + ARG9/(ARG8 + ARG2/ARG1^3) + ARG9/ARG1)/ARG8$

Moving onto the typed results, once again using TSNE to perform dimensionality reduction, visible in Fig 9 (orange are the true values, with blue being the calculated ones based on the function $(ARG10 + 2 * ARG8 + ARG9/(ARG8 + ARG2/ARG1^3) + ARG9/ARG1)/ARG8$), it is clearly difficult to visualize and distinguish the data in this space. Attempting a different form of visualization, histograms were used to collect the samples into bins. As can be seen in Fig 10, the GP system appears to cluster the majority of the ”quality” value of 5 and 6 into the bin labeled ”5”, with a small portion of the records moving into bin ”6”. The function can be seen to distinguish between quality values 4 and those greater than 4, but is not as adept at distinguishing between the values of 6, 7, and 8, as the majority of those values appear to be in the ”6” bin. Once again, it can be observed that not all the input variables are used.

VI. Conclusions and Future Work

As a whole, the GP system performed adequately well on lower dimensional data, and was unable to generate a more accurate function when high dimensional data was passed into the system. An interesting observation mentioned multiple times previously was how some input variables were deemed ”unimportant” by the system, which neglected to use said variables in a final solution. An important notion to be aware of is the affect that TSNE could have on the plotting of the results, whic may result in not painting an entirely accurate picture.

References

[1] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[2] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[3] Maarten Keijzer. Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269, 2004.