Genetic Programming for Symbolic Regression

Chi Zhang
Department of Electrical Engineering and Computer Science,
University of Tennessee, Knoxville, TN 37996, USA
Email: czhang24@utk.edu

***Abstract*—Genetic programming (GP) is a supervised learning method motivated by an analogy to biological evolution. GP creates successor hypotheses by repeatedly mutating and crossovering parts of the current best hypotheses, with expectation to find a good solution in the evolution process. In this report, the task to be performed was a symbolic regression problem, which is to find the symbolic function that matches a given set of data as closely as possible.**

By training the genetic programming algorithm with the given data set, the relationship between the input and output was represented by functions generated in the training process. If the error rate reached a certain threshold, the training can be stopped and the testing can be applied to verify the effectiveness of the best function. Observations I made during the experiments are based on two aspects of the GP: search space and variety of generations. Firstly, the function and terminal sets had a powerful impact on the performance. Too many functions and terminals increased the search space exponentially, resulting in a long training times and more local minimums, so that the algorithm was more likely to be trapped by some suboptimal solutions. On the other hand, too few functions failed in thoughtfully representing the relationship of given dat, so that failed in finding a fitting function. Secondly, the depth of trees can influent the search space. Constraint of the maximum depth of trees in the training helped find good functions more quickly. Thirdly, too low variety made the solution converge to some local minimums. The size of population, the selection method and selection percentage are main factors affect the variety of generations. With finely tuned parameters, the genetic programming applied in this project can find a best function with error rate of 0.143 within 60 generations.

I. Introduction

Genetic programming (GP) is an evolutionary computation method to find computer programs that can perform a task[1]. Programs manipulated by a GP are usually represented by trees; a node in the tree represents a function call and a leaf in the tree represent a value to be plugged in functions[2].

In this project, a genetic programming algorithm was designed to perform the task of a symbolic regression problem. The given data had 101 input-output pairs, and the object was finding a function to fit those input-output data as close as possible. To solve this problem, an initial generation with 500 individuals are generated randomly, then crossover and mutation were applied to the individuals to generate offsprings for the next generation. Selection was based on a method I called tournament selection with windows.

II. Genetic Programming Algorithm Design

A. Formulation of the learning task

The genetic programming algorithm used in this project took a given real value in the range of $[-3.0, 7.0]$ as an input, then plugged the input to a function to get an output. Then the output was compared to the target output to compute an error, which was also regarded as the raw fitness of the function. All the functions were labeled with its fitness and were selected by fitness according to some selection methods. Mutation, reproduction and crossover were applied to individuals to generate their offsprings. At last, the offsprings replaced the individuals to form a new generation. By this way, individuals with better fitness were expected to survive and evolve in the training process.

1) Terminal set: The terminal set used in this project was composed by only one input x, while x values were defined in the given data. I also tried with some constants of 1, 2, 3, 5 to explore the effect of different terminal sets. More details are given in the later section.

2) Primitive function set: Primitive function set was composed by the simple mathematical operations, e.g., $+$, $-$, $\times$, $\div$, cos, sin, exp, log. In this project, I tried several different sets of functions and found that $+$, $-$, $\times$, $\div$, cos, sin were enough to represent the given data. The number of operands for $+$, $-$, $\times$, $\div$ are all 2. The functions of cos, sin have one operand. To protect the case of "divided by 0", the output of any number divided by 0 was defined as 1.

3) Method of probabilistic selection: In the mutation, reproduction, and crossover steps, one or two individuals would be selected from the current generation as the parent(s) to generate the offsprings for the next generation. The method of selection had a big effect on both the performance and the training time. In this project, four types of selection methods were tested: the proportional selection, the ranking selection, the tournament selection, and the tournament selection with windows. The final results showed that the tournament selection with window size of 30 achieved the best performance, while the other selection methods all failed in finding a good solution.

Moreover, due to the specific relationship of the input-output data pairs, in the crossover step, it was better to select one parent based on the tournament selection method, and another parent selected randomly, to introduce more variety for generations.

Algorithm 1 Genetic Programming Pseudo Code for the symbolic regression problem

Initialize population *P:*

- Generate p trees at random, for each tree:
  - Randomly generating a rooted tree with ordered branches
  - Randomly select functions from function set to be root
  - Create $Z(f)$ children; each function has $Z(f)$ arguments
  - Repeat recursively until tree is completely labeled with terminal as leaves

Evaluate: For each h in P, compute $Fitness(h)$
While $[maxFitness(h)] < Fitness\,threshold$ do

- Create a new generation Ps:
  - Select: Probabilistically select $(1 - r)p$ member of P to add to P_s. The probability $Pr(h_i)$ of selecting hypothesis h_i from P is given by

$$Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

  - Crossover: Probabilistically select $\frac{rp}{2}$ pair of hypotheses from P, according to $Pr(h_i)$ given above. For each pair, $\langle h_1, h_2 \rangle$, produce two offspring by applying the Crossover operator. Add all offspring to Ps
  - Mutate: Choose m percent of the members of Ps with uniform probability. For each, replace a subtree by a randomly generated new tree.
  - Update P with Ps
  - Evaluate: for each h in P, compute $Fitness(h)$

Return the hypothesis from P that has the highest fitness

4) Fitness function: In this project, the raw fitness was defined as the sum of squared error between the actual output and the target output. To make things simple, errors mentioned in the rest of this report are the sum of squared error. Since the output of each function needs to be normalized to [0, 1], the raw fitness was normalized by the sum of all fitness when doing selection.

B. Genetic programming algorithm

After outlining the structure of the genetic programming, pseudo code of the algorithm (Algorithm 1) is shown as below.

1) Initialize generation with ramped half-and-half method: The method I used to create the initial trees were "ramped half-and-half", which means it created an equal number of trees using a depth parameter that ranges between 2 and the maximum specified depth. In this project, the maximum depth was 12. Because the duplicates of trees were required to be eliminated from the original population and the functions in the function set are limited, trees with depth of 2 are less than the others. For example, for the function set of $+$, $-$, $\times$, $\div$, cos, sin, there were only 6 different trees with depth of 2.

2) Copy, crossover and mutation: When all the individuals were initialized, they were probabilistically selected and copied to the new generation by crossover, reproduction and mutation. Then crossover was applied by selecting two individuals as parents according to the selection method, and then one random node of each tree was selected as the crossover point. The two parents exchanged their subtrees at the crossover point, to generate two new offsprings for the next generation. In the mutation step, one individual was probabilistically selected as the parent, and one node in the parent tree was selected as the mutation point. Then a newly generated tress replaced the old subtree in the mutation point.

To avoid continues growing of the tree, a maximum depth was utilized in the algorithm to control the size of trees. In this project, the maximum depth of tree was around 15.

III. Experiments

A. The best function

After doing experiments with lots of different parameters, the best functions I found was as following:

$cos(x) + (cos(cos((cos(x) * ((cos(cos(cos(2 * x))) - cos(cos(cos(x)))) * cos(x))))) - cos((((cos(x) + (((cos(x) * cos(x)) - cos(cos(x))) * cos(x))) * cos(x)) * ((cos(x) * cos(x)) * cos(x)))))) * (cos(x) + (cos(cos(x)) - cos(((cos(x) + (((cos(x) * cos(x)) - cos(cos(x))) * cos(x))) + (cos(x) * cos(x) - cos(cos(x))) * cos(x)) * cos(x))) * cos(x))$

The most fit function achieved a sum of squared error as low as 0.143, with only 60 generations. Figure 1 shows results of the data points of the given data, along with the data values generated by the best function the genetic program learns. The found best function proved that with proper settings, the GP algorithm was able to solve the symbolic regression problem within an acceptable number of training iterations.

B. Fitness in training

Before plotting readable graphs, the raw fitness need to be converted to the adjusted fitness by Eq 1. The raw fitness is the sum of squared error between the actual output and the target output. In the implementation of the GP, all the selections were actually based on such adjusted fitness instead of the raw fitness, because the adjusted fitness ranges from 0 to 1, which was more suitable for calculation.

$$Adj\ Fitness = \frac{1}{1 + Raw\ Fitness} \tag{1}$$

In Figure 2, the fitness of the best individual of the population was displayed as a function of the generation number. The fitness of the best individual increased quickly after 30 generations. In the 60$^{\text{th}}$ generation, the error rate was lower than 0.25, which was the threshold for the training termination.

The average fitness of population was plotted in Figure 3. Before 40 generations, the average fitness was quite low, which was consistent with the trending of the best individual fitness in Figure 2. It indicated that most functions in the current generation poorly fit the given data. Because mutation and crossover can introduce new parts to the individuals, it

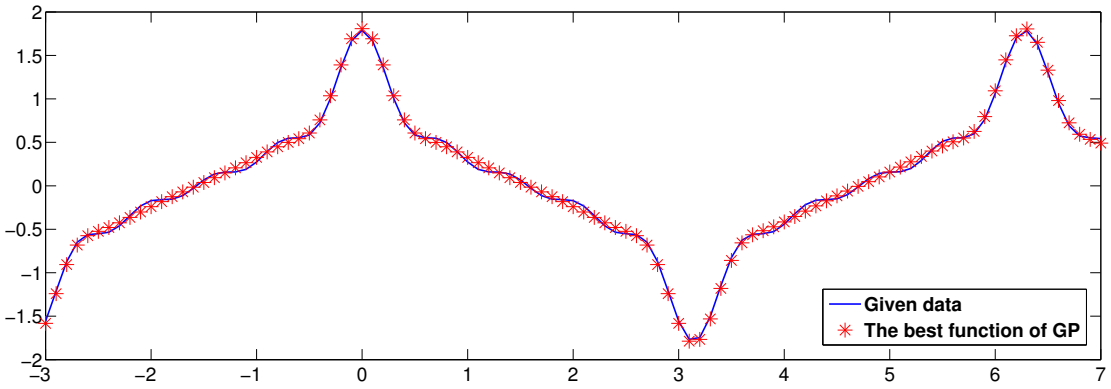

Figure 1. Error versus weight updates(epoch)

was reasonable to observe large fluctuation of the average fitness. As Figure 3 shows, after 40 generations, the average fitness began to increase. At the termination point of the 60th generation, the average fitness was 0.025, which means the average error was 59.82.

C. Variety of population

The variety of population indicates how many individuals are different in generations. In the experiments, I found that if the variety was alway low, the GP was likely to be trapped in some local minimums. Therefore, to avoid the low variety, multiple factors need to be carefully designed. Figure 4 shows that the variety decreased to a quite low level at the early generations, which means the GP might be trapped by some local minimums, but fortunately, after generation 20, the variety began to increase and maintained in the level of 60% ~80% until the training termination. The next section gives more details on it.

IV. Discussion

A. Effect of function and terminal sets

At the beginning of the experiments, I had no idea what kind of function sets and terminal sets were necessary for solving this symbolic regression problem, so that I used functions as many as possible. The initial function set I used contained: $+$, $-$, $\times$, $\div$, cos, sin, exp, log, and $\sqrt{}$. The terminal set was: $1, 2, 3, 5$ and x. However, by doing experiments with these functions and terminals, I found that the GP was always easily trapped by the local minimum of function $cos(x)$, and the average fitness was always high. Meanwhile, I fount that the constant elements in the terminal set disappeared quickly in training.

A typical case was that at the generation number of 300, the error of the best individual was still 5.18. The function of the root node was cos, which was obviously impossible to achieve an error lower than 0.25, since $cos(\ldots) \in [-1, 1]$. In the later generations, the best individual was trapped in cos and could not jumped out of this local minimum. The reason might be that such a big function set and terminal set resulted in a very large search space with more local minimums, so that the GP was more likely to be trapped by the local minimums and failed in find a fitting function.

Based on the above experiments, the final terminal and function sets that can work is as following:

$$Termianal\ Set = \{x\}$$

$$Function\ Set = \{+, -, \times, \div, cos, sin\}$$

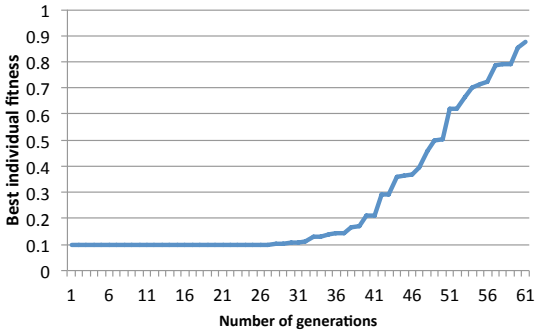

Figure 2. The best individual fitness in the population

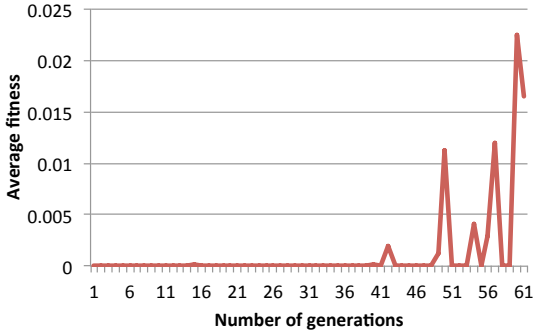

Figure 3. The average fitness of the population

B. Effect of the maximum depth of tree

The maximum depth of tree had a big impact on the search space in this project. At the beginning of experiments I didn't constrain the depth of trees and found that the trees would grow rapidly to a very large size and the GP was easily trapped by some local minimums. The reason was that if the trees were too large, nodes close to the root were less likely to be visited as a crossover or mutation point, so that it was more likely to be trapped in a tree with a fixed function labeled in the root node. On the other hand, if the trees were too small, the GP can not thoroughly represent the given data. Therefore, choosing a proper maximum depth was important for this symbolic regression problem. In the project, 15 and 16 were good candidates for the maximum depth of trees. Results are showed in Figure 5.

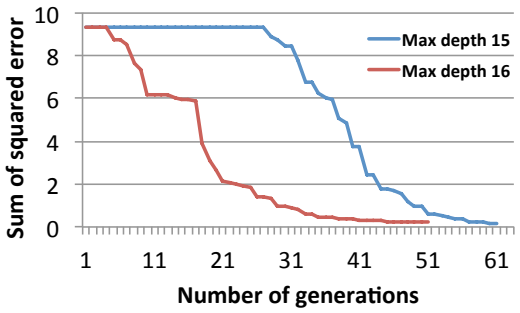

Figure 5. Effect of learning rate in terms of squared error

C. Initial population

1) Variety of initial population: The initial population affects the performance of genetic programming in the way of variety. Experiments without eliminating depilates in the initial generation always resulted in being trapped by $cos()$.

In the experiments, an interesting observation was that the way of generating random number can affect the algorithm's performance greatly. The random numbers in the GP relied on the system random number generator, which is the "rand()" function call. I utilized the current system time as seed, instead of a fixed system time, to introduce more random degree into the algorithm. It has been proved to perform better, as Figure 6 shows. The total generations required by the unfixed random seed approach is lower than the number of generations required by the fixed random seed.

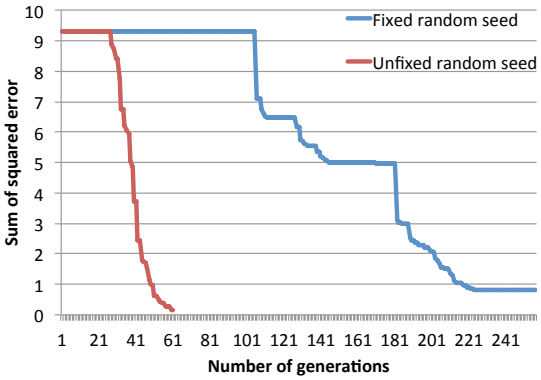

Figure 6. Effect of hidden units number in terms of squared error

D. Effect of population size

To explore the effect of population size, I did experiments with population size from 350 to 600. As Figure 7 shows, if population size was small, e.g., 350, the GP will take more generations to reach a good solution. The number of generations before convergence for population size of 400 and 500 are very close, but the error of population 400 was a little bit higher than population size of 500. Note that although population size of 600 had the smallest number of generations to get a good solution, its actual training time was longer than population 500, because it took much longer time for updating generations.

From this experiment, I draw the conclusion that the large size of population resulted in the more powerful representation ability to fit the training data, but also took longer training time for generating a new generation due to the large number of individuals.

E. Effect of percentage for selection and selection method

1) Method of selection: The selection method used in this project was tournament selection with window size of 30. This method extends the tournament selection approach, in the way that after one individual was selected from a tournament pair, it was put into an array of size 30. The final individual was the best individual in the array. This method was the

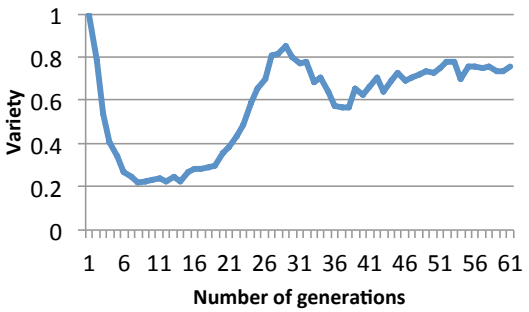

Figure 4. The variety of the population

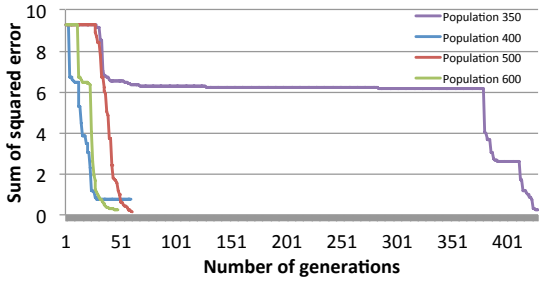

Figure 7. Effect of different input generalizations

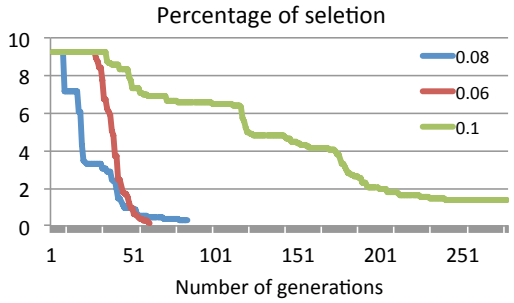

Figure 8. Effect of learning rate in terms of squared error

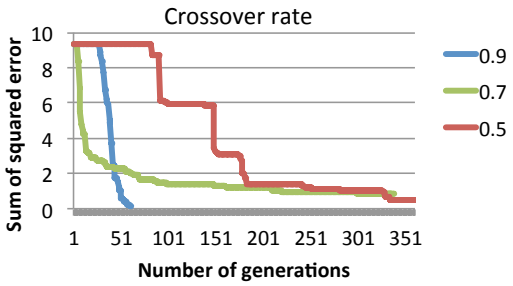

Figure 9. Classification rate and training time of k-fold and regular approach

only one can work in this project. The proportional selection, ranking selection and tournament selection all failed in finding a solution. It might because none of them can achieve good variety of the generation, which means the possible good individuals were alway abandoned.

Moreover, to increase the variety, one parent in the crossover was selected randomly, while another one was selected by the selection method.

2) Percentage of selection: Since tournament selection with window was proved to be effective in learning a good solution, I did experiments with different percentage of selection. The GP algorithm was quite sensitive to the percentage of selection. The percentage of 0.06 was the best values I can find, when most of the other values failed in learning a good solution. During the experiments, I found that if the selection rate was too small, the algorithm would behavior similarly to the classic tournament selection method, which has been proved to have a poor performance. On the other hand, if the percentage of selection was too high, the current known good individuals were more likely to be selected, which reduced the chance to find the actually good solutions over the entire evolution. Figure 8 shows the performance of different selection rates. The percentage of selection lower than 0.06 was not displayed here because they failed in finding a learning function and the GP was always trapped by $cos()$.

3) Threshold of tournament selection: The tournament selection thresholds from 0.5 to 1.0 were also explored with the experiments. The final results showed that there is no much difference of regression performance and training time for different tournament selection thresholds. It is because that the individuals selected by the tournament selection also need to compete with other good individuals in a pool. This step decreased the probability of selecting individual with poor fitness, so that in spite of the value of threshold, the final performance were very similar.

F. Effect of crossover rate and mutation rate

1) Crossover rate: Crossover rate decides how many individuals are generated by crossover in the next generation. Because the error space for the GP is not continues, and multiple factors affect the algorithm interactively, it is reasonable to observe results with high fluctuations. Therefore, the effect of different crossover rates were found unpredictable. In this experiment, the percentage of 0.5, 0.7 and 0.9 were tested. According to the results showed in Figure 9, I draw the conclusion that with low percentage of crossover, the convergence was slower than high percentage of crossover. Note that in the case of 0.7, the tournament selection rate was tuned to be 0.09, otherwise it cannot find a good solution.

Crossover rate	Error of the best function	# of generations
0.5	0.456	359
07	0.832	337
0.9	0.143	60

Table 1
REGRESSION ERROR AND LEARNING GENERATIONS

Table 1 showed the number of generations before finding a good solution, and the best functions for each case. It is obvious that with 0.9 crossover rate, the GP had the best performance.

2) Mutation rate: The best mutation rate found during the experiments was 0.1. If the percentage was higher than 0.1, the GP failed in finding a good solution. This was because large mutation rate introduced more random degree to the GP, and the good solutions would be replaced by new subtree in the mutation process. If the rate was lower than 0.1, the GP would took a relatively longer training time to converge. Figure 2 shows that with small mutation rate, the total generations required by training were more than 100. It was because at each training epoch, only a small number of individuals were selected to do mutation, so that only a small number of new parts of trees were generated to the generation, which resulted in more epochs for training.

Table 2 shows that with mutation rate of 0.1, the GP can converge to a good function within 60 generations.

Mutation rate	Error of the best function	# of generations
0.02	0.239	137
0.05	0.787	150
0.1	0.143	60

Table 2
REGRESSION ERROR AND LEARNING GENERATIONS

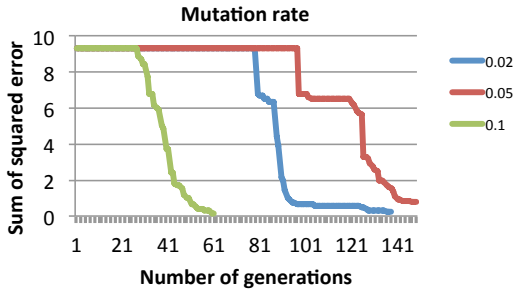

Figure 10. Classification rate and training time of k-fold and regular approach

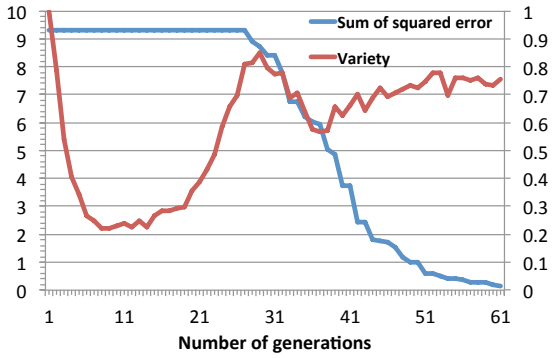

Figure 11. Classification rate and training time of k-fold and regular approach

G. Variety and fitness

For all of the above experiments, I collected data of variety and fitness and plotted two of them in Figure 11 and Figure 12. The results showed that the variety was relevant to the best fitness value. In both of the figures, when the variety began to increase, the error of the best function of generation began to decrease, which mean the variety had reversed trending with the error of the best function. This phenomenon illustrated that when variety began to increase, more differential functions were generated, consequently, a good solution appeared in the generation with high possibility.

V. Conclusion

In this project, a genetic programming algorithm was implemented to solve a symbolic regression problem. The object is to find a function fitting the given input-ouput data as close as possible.

I made some observations during this projects. Firstly, the genetic programming algorithm didn't follow a continues error space, which means the good solutions were discretely distributed among the search space. This explained lots of phenomenons in the experiments, e.g., values of the crossover rate and the mutation rate didn't affect the performance in a continues fashion, which means while some values led to good solutions, other values very close to the effective value would fail in finding a fitting function for the given data.

Secondly, because of the uncontinuous error space and interactional effect from various factors, the algorithm need

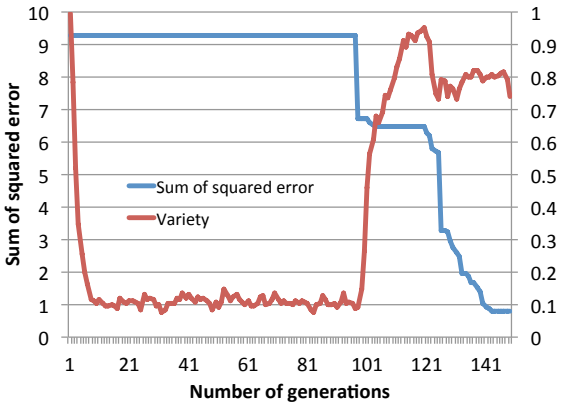

Figure 12. Classification rate and training time of k-fold and regular approach

to be carefully tuned to find a good configuration for the data of this problem. An interesting observation was that if the given data had a simple relationship, the more general configuration can guarantee to find a solution with good quality. However, when the training data changed, the original algorithm can easily fail. Therefore, the parameters of the genetic programming algorithm need to be finely tuned for the new data.

Thirdly, the variety of generations was critical for finding a learning function. If the variety is too low, all the individuals have similar functions, and the algorithm would be trapped by some local minimums. The factors relevant to the variety are the mutation rate, the crossover rate and the selection method.

Lastly, the size of search space was another big issue in the GP. If the search space is too large, it will take a long training time to find good solutions. Moreover, in a large search space, there were more local minimums, which means the GP was more likely to be trapped and failed in finding a solution. In this project, I controlled the search space by selectting function set and terminal set as small as possible, and limiting the maximum depth of trees in generations. These efforts were proved very effective in finding a high quality solution with an acceptable generation number.

In general, the best function I found for this problem can achieve an error rate as low as 0.143, after being trained for 60 generations.

References

[1] Tom Mitchell, McGraw Hill, *Machine Learning*. Mcgraw-Hill International Edit, 1997

[2] Koze J. R., *Genetic Programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.