Using Genetic Programming to Perform Time-Series Forecasting of Stock Prices

Anthony Hui
Computer Science Department
Stanford University
Stanford, California 94305
huics@cs.stanford.edu

ABSTRACT

The goal of this project is to explore the use of genetic programming to perform stock time-series analysis. Using the genetic programming capability provided by lilgp (and later on by the constrained genetic programming extension CGP lilgp 2.1;1.02), we built a genetic program that is capable of producing individuals which give a prediction of the stock price of the next trading period based on the stock prices (or other derived data) of the past *N* periods, where *N* is a user-defined variable (hereafter called the window period). We made different attempts to enhance the performance of the best-of-run individual produced by the genetic program, such as changing the data that it was learning on and experimenting with different window periods and population sizes. We found that with suitable parameters, genetic programming is capable of producing individuals which give reasonably close predictions both within the training set and in cross-validations.

1. Introduction

Unlike machine learning problems such as the 4-parity problem, the problem of stock price time-series forecasting is especially challenging since the data is often chaotic and volatile and does not necessarily exhibit obvious logical patterns. In this project, we used the adjusted closing stock prices of the IBM stock from 2 January 2001 to 12 Nov 2003 (from http://finance.yahoo.com) as the training data for the genetic program. The experiments that we ran ranged from using the primary raw data of the adjusted closing prices to using secondary derived data such as changes in the stock price from the previous trading period.

1.1 Information Specific to the Time Series

The time series that we used has a mean price of 91.157 and a standard deviation of 15.143. The maximum stock price in our time series is 123.74 and the minimum 54.55.

Adjusted closing stock prices refer to the closing prices of the stock after having been adjusted for all applicable splits and dividend distributions. http://finance.yahoo.com explains: "*Data is adjusted using appropriate split and dividend multipliers. Split multipliers are determined by the split ratio. For instance, in a 2 for 1 split, the pre-split data is multiplied by 0.5. Dividend multipliers are calculated based on dividend as a percentage of price, primarily to avoid negative historical pricing. For example, when a $0.08 cash dividend is distributed on Feb 19, and the Feb 18 closing price was 24.96, the pre-dividend data is multiplied by (1-0.08/24.96) = 0.9968.*"

2. Phase I – Learning on Raw Stock Prices

2.1. Basic Architecture

The architecture of the genetic program in phase I was relatively simple. In this first stage of the implementation, we used the adjusted closing prices of the IBM stock in our simulation period as the training data. The period of past stock prices

which the genetic programming had access to, which we called the window period *N*, was set to 5. We calculated the fitness using a simulation process over a period of *F* days, where *F* was the number of fitness cases set in advance. On each of these *F* days, an individual would look back on the past 5 days and give a prediction for the closing stock price of the next day. If we designate today as *day(t)*, on each simulation, the genetic program would have access to the closing prices of *day(t)* (today's closing price), *day(t-1)*, *day(t-2)*, *day(t-3)* and *day(t-4)* and give a prediction of the closing price on *day(t+1)* based on these prices. Each of these five prices was represented by a terminal in the function set, which simply returned the corresponding closing price from the stock price array created upon initialization of the genetic program. The variable *t* was incremented continuously throughout the fitness evaluation process so that the agent "moved" through the entire simulation period of *F* days, making a prediction on each day by looking back on the past 5 stock prices.

The numerical raw fitness of an individual was simply defined as the total squared error of all the predictions made by the individual in the simulation period.

2.2. Results and Problem Encountered

We set the number of fitness cases *F* to 200 and ran several trial runs. The population size *M* of each generation was set to 1000, with a maximum generation *G* of 200. The crossover percentage was 0.9, the reproduction 0.1 and the mutation 0.1. The function set consisted of addition, subtraction, multiplication and division, and the terminal set consisted of the five terminals returning the stock prices from the past five days as well as the ephemeral random constant.

The problem with this simple implementation was immediately apparent – the genetic program converged to the local minimum of always giving yesterday's stock price as its prediction. A closer look at the evolution process indicated that only after a few generations, the entire population was dominated by the individual which only returned yesterday's stock price as the prediction. The initial diversity of generation 0 was entirely lost and no more evolution took place.

In one of the runs, the genetic programming gave the following best-of-run individual in generation 2 with a raw fitness of 1194.4213:

```
(+ 0.27860
    (+ -0.20547 X1))
```

X1 was a terminal that returned today's closing stock price (the most recent stock price that the individual had access to).

The reason why the solution relied solely on the most recent stock price was that the change of the stock price from day to day was small enough to give a high enough adjusted fitness to out-compete all the other less accurate individuals. Thus the genetic program simply resorted to this local minimum as the solution.

3. Phase II – Training On Derived Data

3.1. Changes to the basic architecture

To overcome the local minimum problem encountered in the first phase, we changed the data that the agent was learning on. Instead of using the raw stock prices, we switched to using derived data – changes in the adjusted closing prices from the previous trading day. Since changes in stock prices are not likely to be close with each other on a day-to-day basis, the genetic program will be less likely to give an individual which relies too heavily on yesterday's data as the prediction. With a prediction of the change, one can derive the corresponding prediction of the stock price with the information of today's closing price (today's price + change = prediction of tomorrow's price). In addition, the profit made in stock trading is more dependent on the *change* of stock prices than on the actual prices themselves. Therefore our decision to switch over to learning on changes in stock prices rather than the raw prices is well justified.

We also made a second change in our implementation by generalizing our window period *N*. Instead of fixing *N* to 5, we generalized it so that it could be set to any value within the limits imposed by the stock price data we had. There were a total of 719 data points in our time series, and since we wanted 200 fitness cases, the maximum window period we could have is 719 – 200 = 519. However we also had to leave room for cross-validation, so the actual maximum window period that we could use was smaller than that (see figure 1). The maximum window period *N* that we experimented with was 450.

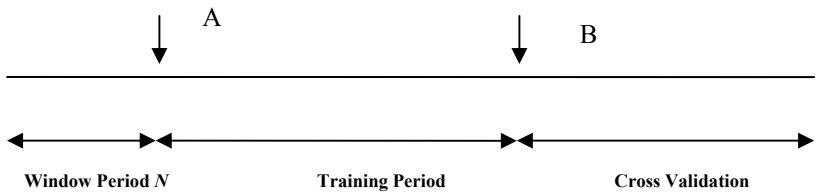

Figure 1: How the time series was divided for training and cross-validation. The cross-validation period followed the training period immediately.

Figure 1 shows how we divided the time series into different parts. When an individual was evaluated, it was moved through the training period, giving a prediction on each day by looking back on the past *N* prices (which was why we there are *N* periods before the first day of the training period). Each day in the training period was therefore a fitness case, and we kept track of the differences between the predictions given by the individual and the actual stock prices from the time series. The fitness evaluation process for an individual thus starts on the first day of the training period (arrow A in the Figure 1), and runs till the last day of the training period (arrow B).

Since we fixed the number of fitness cases *F* to 200, the training period consisted of 200 days in all our experiments.

The cross-validation period followed immediately after the training period and it was not used to evaluate the fitness of an individual. In our experiments, we fixed the cross validation period to be 60 days.

To generalize *N*, instead of creating a separate function for each stock data point that the agent had access to, we created a generic function *stockPrice* as part of the function set of the genetic program. The function *stockPrice* took in one numerical constant as its only argument as the offset and mapped it to a stock data point within the window period *N* and returned that piece of stock data (which in this case was the change in stock price from the previous trading day). This enabled the genetic program to have access to all the stock data points within the window period *N*.

To ensure efficient formation of genetic program trees we also imposed certain constraints to the genetic programming tree formation. With the help of the Constrained Genetic Programming extension developed by Cezary Z. Janikow and Scott DeWeese, we managed to add some semantic constraints to the function *stockPrice*. We allowed *stockPrice* to directly take in only an ephemeral random constant R as its argument. The other functions were not valid arguments.

Since our ephemeral random number generator generates a floating point constant uniformly in the range [0, 1), the agent would have equal probability of access to every single one of the *N* data points within the window period *N*.

This was the major genetic programming implementation that we did our experiments with. We thereby show the parameter tableau on the next page.

Table 1: The Genetic Programming Tableau

Objective	To find a stock predicting agent that performs stock price forecasting to reasonable accuracy based on data from a certain window period *N* from the past history of the stock price time series
Terminal Set	**The ephemeral random floating random constant atom R**
Function Set	**+, -, /, *, *stockPrice***
Fitness Cases	**Two hundred pairs of (x_i, y_i) (x_i, y_i) where x_i is a vector of size *N*:** $x_i = (x_{i0}, x_{i1}, x_{i2}, x_{i3} \ldots x_{i(N-1)})$ **where x_{ij} is stock data point in the time series on day(t-j) and t refers to the present day.** **y_i is the stock data point on day(t+1) (tomorrow's data), which is the target of the agent's prediction**
Raw Fitness	**Sum, taken over the two hundred cases, of the differences between the value returned by the S-expression and the actual value of y_i**
Standardized Fitness	Same as Raw Fitness
Hits	A fitness case for which the error is less than 10% of the target (in this case it is the change in stock price from today)
Wrapper	None
Rules of Construction	The function s*tockPrice* will only take in the ephemeral random constant R as the argument. The maximum tree depth is 20.
Parameters	*M* = 1000, *G* = 200
Termination Criteria	The GP has run for *G* generations or the individual scores 200 hits.
Result Designation	The best-of-run individual

3.2. Training on Changes of Stock Prices

We tried several runs of the new implementation, with all the parameters equal to the ones that we used in phase I except the changes that we documented above. We set the window period *N* to 300. Most of the runs took about 5 to 7 minutes to run, depending on the size of the population *G*.

In one of the runs, the genetic programming produced the following best-of-run individual, with a raw fitness of 822.0671:

```
(X 0.75018
(X 0.37786
   (X (+ (X 0.36341 0.55662)
         (- (stockPrice 0.83162)
            (/ 0.62236 0.65389))) 0.84362)))
```

(The "X" sign is the multiplication function)

With a larger window period and learning on the changes of stock prices instead of the actual stock prices, we managed to obtain an improvement in best-of-run raw fitness (which is also standardized fitness in this case) from 1194 to 822.

3.2.1. Best-of-Run Fitness vs. Population Size

We also fiddled with the genetic programming parameters and ran several more trials with larger population sizes and observed how the fitness of the best-of-run individual changed accordingly. As expected, the raw fitness improved as the population size increased, due to an increase in diversity of the population:

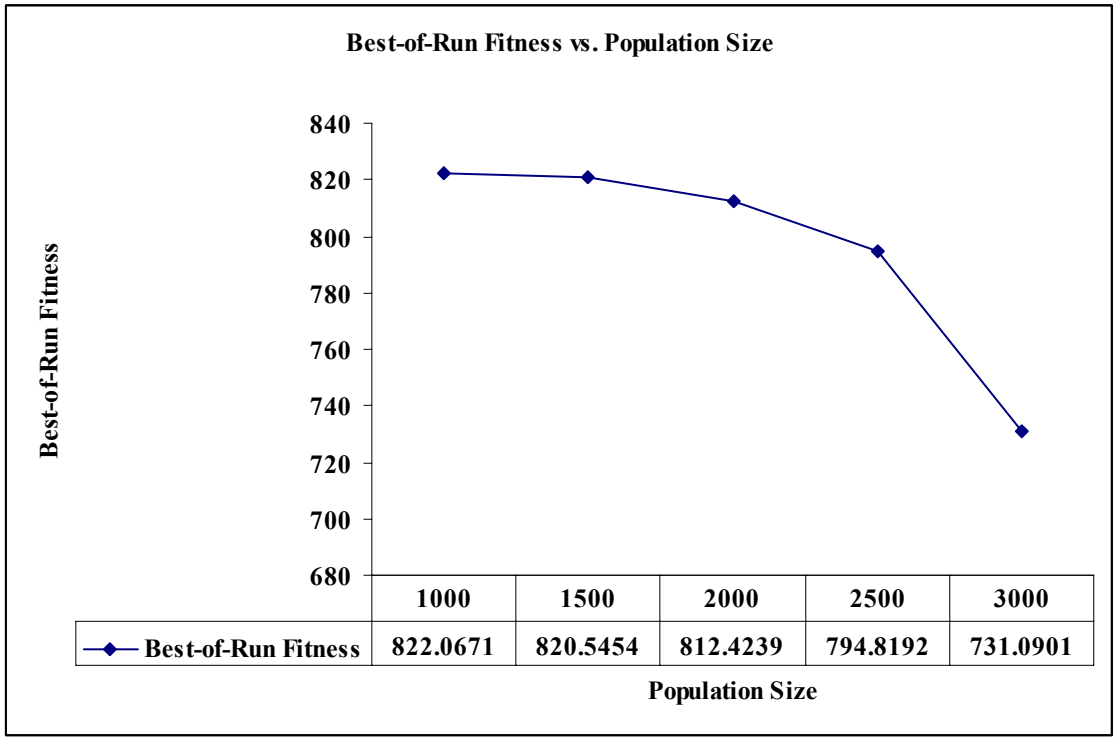

Figure 2: Best-of-Run Fitness vs. Population Size

The best-of-run individual given by the genetic programming when the population size $M = 3000$ was:

```
(/ (/ (stockPrice 0.88133)
      (/ (/ (+ (/ 0.50126 0.01564)
               (- 0.86811 0.05018))
            (- (- 0.05031 0.94362)
               (- 0.34612 0.47739)))
         (X (stockPrice 0.06920)
            (stockPrice 0.37098))))
   (+ (/ (+ (/ (- 0.35228 0.78184)
               (+ 0.92543 0.56005))
            (+ (stockPrice 0.47807)
               (X 0.18911 0.95514)))
         (stockPrice 0.49230)) 0.67947))
```

This individual has a raw fitness of 731.0901 and an average absolute error of 1.514626. (The average absolute error is the total absolute error between the stock price predictions given by the individual and the actual stock prices over the simulation period divided by the number of days in the simulation period, which in this case was 200.)

3.2.2. Window Period vs. Best-of-Run Fitness

Next we experimented with different window periods *N* and observed how the best-of-run fitness changed according to the window period *N*.

At a population size *M* of 3000, we obtain the following empirical relationship between the best-of-run fitness and the window period *N*:

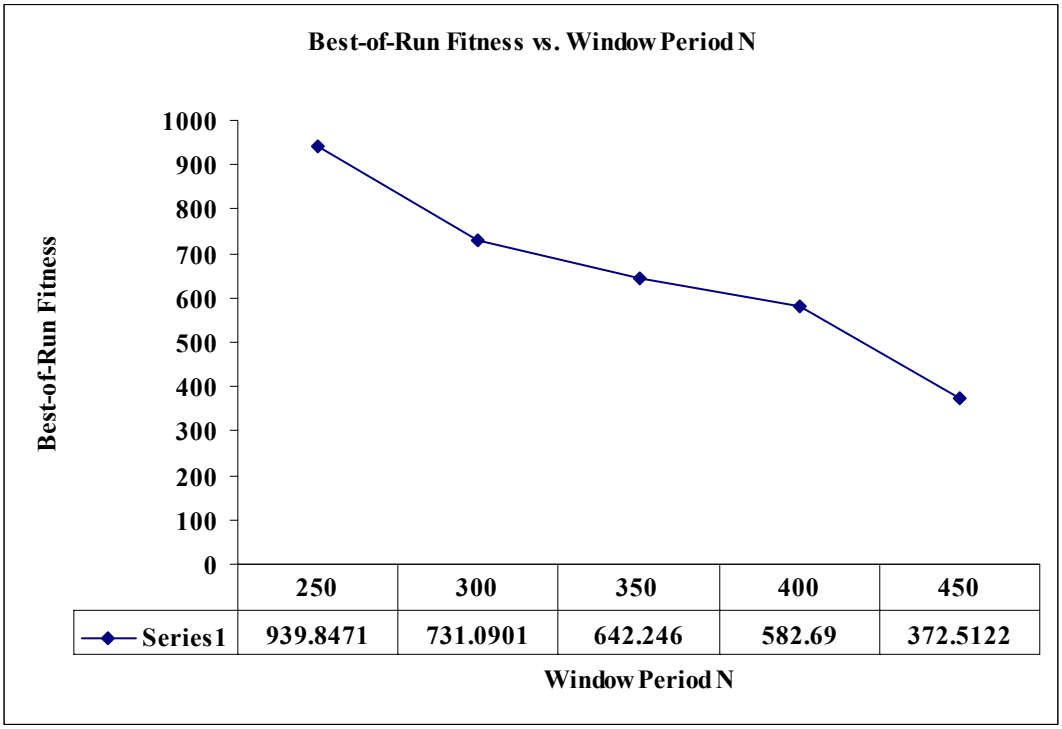

Figure 3: Best-of-Run Fitness vs. Window Period N

An increase in the window period *N* from 250 to 450 resulted in an improvement of the best-of-run fitness as shown in figure 3.

The best-of-run individual given by the genetic program with a window period *N* of 450 and a population size of 3000 was:

```
(- (X (+ (/ 0.09378 0.86352)
         (+ 0.33467 0.49968))
      (+ (X (+ 0.75014
               (X (- 0.88926 0.38868)
                  (X 0.27938 0.13717)))
            (X (X (/ 0.67247 0.53376)
                  (X (stockPrice 0.33351) 0.71524))
               (X (- (X 0.10458
                        (X 0.71833 0.82310))
                     (- (- (/ (- 0.68316 0.22442)
                              (/ 0.33076 0.72767))
                           (/ (+ (- 0.90421
                                    (X (+ (X 0.98134 0.42174) 0.00562) 0.00562))
                                 (- (X (stockPrice 0.43744) 0.07137) 0.12381))
                              (X 0.97391 0.39945)))
                        (stockPrice 0.22934))) 0.11735)))
         (X (- 0.20536 0.00646)
            (X 0.54977 0.06641))))
   (X (X 0.49705 0.02539)
      (X 0.51353
         (- 0.08233 0.44126))))
```

This individual had a raw fitness of 372.5122, an average absolute error of 1.083708 over the 200 cases in the training set.

The experiments in sections 3.2.1 and 3.2.2 also showed that fitter individuals usually referenced the *stockPrice* function more often than individuals with lower fitness. This observation was consistent with the intuition that past stock prices had substantial predictive values for future stock prices. By accessing the past stock prices more often, individuals managed to make predictions with higher accuracy and thus achieve better fitness.

3.3. Cross-Validation of Best-of-Run Individuals

To perform cross-validation on our best-of-run individuals, we performed a simulation on a time period which was not within our training set. In our implementation, the cross-validation simulation period consisted of 60 days, which immediately followed our training period. In other words, if the training period ends on t_e, then $day(t_e+1)$ to $day(t_e+60)$ will be the cross-validation period.

The cross-validation of the best-of-run individual produced by the genetic program with a window period N of 450 and population size M of 3000 showed that it had a total squared error of 72.973310 and an average absolute error of 0.765702 over the 60-day period. Compared to the total squared error of 372.5122 (the raw fitness) and the average absolute error of 1.083708 over the training period, we concluded that the solution given by the genetic program generalized reasonably well in cross-validation.

We also performed cross-validation of the best-of-run individual produced by the genetic program with a window period N of 300 and a population size M of 3000 (the one shown in section 3.2.1). The resulting total squared error was 185.891651 and the average error rate was 1.272369, compared with a total squared error of 731.0901 and an average error rate of 1.514626 in the training set. Again, the solution generalized reasonably well in cross-validation.

4. Conclusions

Given the volatile nature of real-life stock data, genetic programming seemed to do a reasonably good job in producing individuals which could give reasonably close predictions and generalize reasonably well during cross-validation. Learning on changes of stock prices gave much better results than learning on the raw stock prices, and having a longer window period and a larger population size also gave rise to best-of-run individuals with better fitness.

5. Future Work

There are many different approaches that future work can do, one of which is to experiment with the addition of a neural net architecture into the existing genetic programming implementation and observe how the fitness of the best-of-run individual would change accordingly. Plenty of research has been done on the use of neural net to perform stock time-series analysis, and it would be interesting to see how the combination of genetic programming techniques and the neural network architecture would perform in time series forecasting. One important advantage of using genetic programming is that the training time is significantly shorter than those of other machine learning techniques such as neural networks and support vector machines. Combining the neural network architecture and genetic programming might enable us to get the advantages of both and arrive at individuals which give more accurate predictions.

There are also other possible approaches:

Experimenting with a longer time series: A longer time series would allow for a bigger window period N, and based on our results in section 3.2.2, an increase in the window period N seemed to give rise to individuals with better fitness. It would also allow room for a longer cross-validation period, which would give us a better look at how well an individual fares outside of the training set.

Changing the frequencies at which functions are used in the genetic program: The Constrained Genetic Programming extension done by Cezary Z. Janikow and Scott DeWeese enables the probabilities of the functions and terminals being selected as arguments to other functions to be user-adjustable, enabling certain functions to be more frequently (or less frequently) selected. To encourage the genetic program to feed on the past stock prices more often rather than relying on random constants, one might want to change the weights so that the *stockPrice* function is used more often to build the genetic program tree.

Different measures of fitness: While total squared error is a very common way to measure fitness, other measures of fitness such as total absolute error or average absolute error are also worth experimenting with. One other possible way of measuring the fitness of a time-series forecasting agent is to simulate an actual trading environment and make the agent participate in buying and selling stocks, and then evaluate the fitness based on the profit or loss made in the training period. This would require changing the genetic programming implementation around and adding certain trading rules, but would certainly make for an interesting piece of research.

Acknowledgements

We would like to thank Professor John Koza for his useful input to this project and Ken Ashcroft for proofreading this paper.

Bibliography

Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press.

Saxena, Kushagra and Hui, Anthony 2003. *Agent Based Simulation of Option Trading with Heterogenous Agents in an Endogenous Pricing Setting.* Stanford University Computer Science Department and Management Science and Engineering Department.