University of Strathclyde
Department of Computer Science

M.Sc. Financial Technology
(2017/2018)

CS983 – Evolutionary Computation for Finance

Assessment

"Report of predictive power of the application of Genetic Programming"

Joshua Eick

February 23, 2018

Report of predictive power of the application of Genetic Programming

I. Exploring dataset

What is predictive power of Genetic Programming? In Finance the Genetic Programming it is widely used. In order to understand the predictive power of Genetic Programming it's decide to choose a real data set to evolve the application of Genetic Programming. The data set is the daily stock prices of the Bank of Well Fargo. According to Wikipedia (2007) Well Fargo is an "American international banking and financial services holding company headquartered in San Francisco, California. It is the world's second-largest bank by market capitalization and the third largest bank in the U.S. by assets". Having said that, the reason I choose this financial dataset is because this is the most stable bank in United States, which mean that the volatility (standard deviation) of the prices in Well Fargo bank is very small and the fluctuations of the prices of the stock move less than the stock market. In other words, I choose this bank because as you know that greater volatility represent more difficult to predict future values, which means less predictive power. Hence, choosing a data set with less volatility it will be possible to measure the predictive power of Genetic Programming. In fact, Well Fargo bank is so solid and stable that in the financial crisis of 2008 they don't need a bailout cause his risk profile in their loans was very conservative. Even though they do not need a bailout in the financial crisis the Federal Reserve obligates Well Fargo to take the funds that the Federal Reserve was providing in order to issue more loans and in that way move the economy.

On the other hand, in order to have more sense/understand the dataset it's explored the stock prices of Well Fargo through visualization. It's visualize in illustration 1 that the time series between 2007 and 2017 there an increase in volatility. Therefore, it's choose a subset of the time series of 100 daily stock prices of the Bank of Well Fargo from January 1, 2007 to April 10, 2017 because have less variability.

Illustration 1

On the perspective as an investor it's choose the Well Fargo closing stock prices because are the important stock prices to predict at the end of the day, it's visualize in illustration 2.

	WFC.Open	WFC.High	WFC.Low	WFC.Close	WFC.Volume	WFC.Adjusted
2007.000	35.93	36.01	35.37	35.74	12447600	26.52586
2007.033	35.74	36.02	35.54	35.80	11060500	26.57040
2007.067	35.79	35.83	35.51	35.60	10329200	26.42195
2007.100	35.61	35.89	35.39	35.50	11650400	26.34774

Illustration 2

Therefore, it's choose a subset of the time series of 100 daily stock prices of the Bank of Well Fargo from January 1, 2007 to April 10, 2017 because have less variability. It's visualized in illustration 3 in order to see how the stock price behave between this range of 100 days.

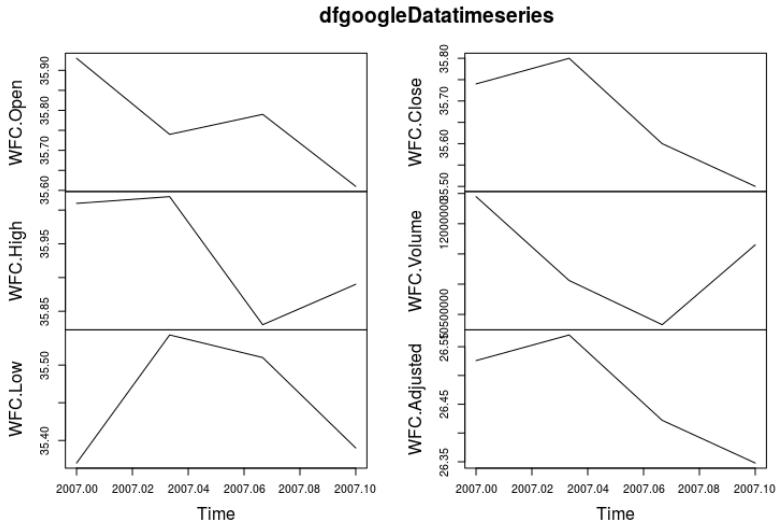
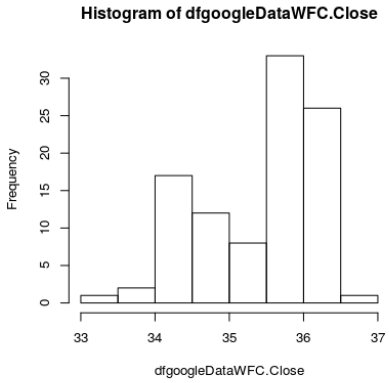
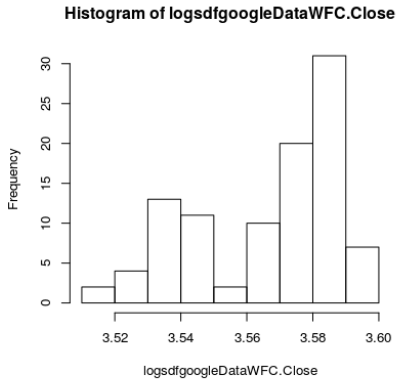


Illustration 3

Moreover, in order to analyze the distribution of my data set and to figure out it's normal distributed; it's plotted a histogram. As result of the histogram on Illustration 4, it's identify that the data set is skew to the right with tail to the left, which means it's not normal distributed and as well have extreme values. Hence, it's modified the data set to log and as result it's turn out to be more distributed data and the extreme values are mitigate. Indeed, been able to transformed the data results on decreasing significantly the standard deviation from 0.7655157 to 0.02178794, which it's the goal with log. Because one of the key measures to have higher predictive power which is the equivalent of high fitness value is the measure of volatility of your dataset. Thus, this represent we can go forward to predict future values with my data log in the application on Genetic Programming.






```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  33.47   34.63   35.68   35.40   36.01   36.56

Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.511   3.545   3.575   3.566   3.584   3.599
```

```
> sd(dfgoogleDataWFC.Close)
[1] 0.7655157
> sd(logsdfgoogleDataWFC.Close)
[1] 0.02178794
```

Illustration 4

II. Details of the process of GP

Evolving a genetic program for this data set in order to predict future values we follow a process which contain key parameters, which you can see the summary in Table 1.

First Scenario
Key parameters
Population size=100 daily stock prices of Well Fargo
Window size=5
Training data-(2/3*100)=66 daily stock prices
Test data-(1/3*100)=33daily stock prices
Iterations=100 seconds

Table 1

First, my population size is going to be 100 daily stock prices of 2007, which equals 3.3 months. I choose this population size of 3.3 months because the analysts in the industry of finance try to predict every quarter the stock prices cause public companies issue quarterly the earnings reports. Secondly, as my volatility is low I choose 5 windows because the next values predicted by the model it's not going to be influence by other values that happen a long time ago. In other works, because my data is financial data it's affected by external economic factor in the long run. That's why the window size is key

measure because represent how many values the Genetic Programming Process will take from the original data to predict the next value, in this case value number 6.

Third, I choose my training data to be 66 daily stock prices (100*2/3=66) and 33 daily stock prices (100*1/3=33). Because it's widely agreed that you training data have to be 2/3 of your dataset and test data 1/3 of you data set. This represent that my training dataset I choose is the original dataset I going to use to predict futures values. As well I choose test data, which is the original data I going to use to prove the predictive power of the model that generate with Genetic Programming. Using the method with training set and test set is well known as cross validation process and is widely use when you have a population size of 10 to 100.

Fourth, choose a iteration number of 100 seconds. Because usually more iterations equals better predicting power cause the root mean square error decrease.

Fifth, it's perform a fitness function on the training set to predict values from 5:66 based on values 1:5, 2:6 and soon on. The fitness function generates the best result for the predicting future values minimizing the variance of the predictive with sum mean square error. The sum mean square error is equal to the fitness value. It's the key measure to calculate the predictive power in Genetic Programming. According to The Analysis Factor (2018) the "Root Mean Square Error can be interpreted as the standard deviation of the unexplained variance".

Overall, it's selected the appropriateness of the choices on the parameters cause it have more interaction with my data set. Therefore, it's performed a fitness function and best result and is as follow in Illustration 5.

```
gpResult1 <- geneticProgramming(functionSet=functionSet1,
+                         inputVariables=inputVariables,
+                         constantSet=constantFactorySet1,
+                         fitnessFunction=fitnessFn,
+                         stopCondition=makeTimeStopCondition(100))
STARTING genetic programming evolution run (Age/Fitness/Complexity Pareto GP search-heuristic) ...
evolution step 100, fitness evaluations: 4950, best fitness: 0.364192, time elapsed: 3.43 seconds
evolution step 200, fitness evaluations: 9950, best fitness: 0.364192, time elapsed: 6.7 seconds
evolution step 300, fitness evaluations: 14950, best fitness: 0.363699, time elapsed: 9.92 seconds
evolution step 400, fitness evaluations: 19950, best fitness: 0.362879, time elapsed: 13.23 seconds
evolution step 500, fitness evaluations: 24950, best fitness: 0.362588, time elapsed: 16.69 seconds
evolution step 600, fitness evaluations: 29950, best fitness: 0.361869, time elapsed: 20.16 seconds
evolution step 700, fitness evaluations: 34950, best fitness: 0.361745, time elapsed: 23.54 seconds
evolution step 800, fitness evaluations: 39950, best fitness: 0.361745, time elapsed: 26.84 seconds
evolution step 900, fitness evaluations: 44950, best fitness: 0.361745, time elapsed: 30.13 seconds
evolution step 1000, fitness evaluations: 49950, best fitness: 0.359965, time elapsed: 33.5 seconds
evolution step 1100, fitness evaluations: 54950, best fitness: 0.359907, time elapsed: 36.85 seconds
evolution step 1200, fitness evaluations: 59950, best fitness: 0.357466, time elapsed: 40.21 seconds
evolution step 1300, fitness evaluations: 64950, best fitness: 0.357435, time elapsed: 43.54 seconds
evolution step 1400, fitness evaluations: 69950, best fitness: 0.357425, time elapsed: 46.97 seconds
evolution step 1500, fitness evaluations: 74950, best fitness: 0.357425, time elapsed: 50.36 seconds
evolution step 1600, fitness evaluations: 79950, best fitness: 0.357425, time elapsed: 53.7 seconds
evolution step 1700, fitness evaluations: 84950, best fitness: 0.357425, time elapsed: 57.09 seconds
evolution step 1800, fitness evaluations: 89950, best fitness: 0.357425, time elapsed: 1 minute, 0.6 seconds
evolution step 1900, fitness evaluations: 94950, best fitness: 0.357425, time elapsed: 1 minute, 4.14 seconds
evolution step 2000, fitness evaluations: 99950, best fitness: 0.357425, time elapsed: 1 minute, 7.69 seconds
evolution step 2100, fitness evaluations: 104950, best fitness: 0.357425, time elapsed: 1 minute, 11.21 seconds
evolution step 2200, fitness evaluations: 109950, best fitness: 0.357400, time elapsed: 1 minute, 14.77 seconds
evolution step 2300, fitness evaluations: 114950, best fitness: 0.357400, time elapsed: 1 minute, 18.4 seconds
```

```
evolution step 2400, fitness evaluations: 119950, best fitness: 0.357397, time elapsed: 1 minute, 21.81 seconds
evolution step 2500, fitness evaluations: 124950, best fitness: 0.352135, time elapsed: 1 minute, 25.31 seconds
evolution step 2600, fitness evaluations: 129950, best fitness: 0.349457, time elapsed: 1 minute, 28.76 seconds
evolution step 2700, fitness evaluations: 134950, best fitness: 0.349428, time elapsed: 1 minute, 32.39 seconds
evolution step 2800, fitness evaluations: 139950, best fitness: 0.349428, time elapsed: 1 minute, 35.92 seconds
evolution step 2900, fitness evaluations: 144950, best fitness: 0.349426, time elapsed: 1 minute, 39.4 seconds
Genetic programming evolution run FINISHED after 2916 evolution steps, 145750 fitness evaluations and 1 minute, 40.02 seconds.
>
>
> best1 <- gpResult1$population[[which.min(gpResult1$fitnessValues)]]
> best1
function (x1, x2, x3, x4, x5)
x5 - (x5 - x2)/((x5 - x3 + -0.326692152203612/x3) * (x4 + x5)) -
   (x5 - x2)/8.00140941837023 - 0.0184011285311718

> rmse(trainingResults,trainingdata[5:66])
[1] 0.09171655


> rmse(predictions[1:33],testdata[1:33])
[1] 1.465386
```

Illustration 5: Results of the first scenario

The best result represents the best formula to fit the model to predict in the dataset that generate Genetic Programming. The best result show as well how many variable/values is going to use to predict the next value.

III. 1-Analysis of the results of the GP (first scenario)

In order to measure the predict power of Genetic Programming and predictive model/best result is accurate analysis our overall fitness value. Thus, it's generated a new vector with our fitness function and compares it with the training data/original data. It's calculated the difference between this new vector predicted and the training set/original data to evaluate if the model is good. The root mean square error, which is our fitness value is 0.09171655(on the training set). These represent good predictive powers cause the lower or closer to 0 the higher the predictive power. In the Illustration 5 it's the output of root mean square error.

Now we know it's a good model to predict it's generate a function on the test dataset to predict values from 67 to 100 of the data set. The process start with last 5 training values from training data and predict future value.

In order to measure fitness function it's compare predictions from 67-100 daily stock prices against test dataset/original data 67-100 daily stock prices. Afterward, it's calculated the root mean square error, which equals 1.465386(on the test set), which you can see in the illustration 6. This represents a significant good fitness value/predictive power cause the value is small. Therefore, in order to understand better the predicting power of Genetic programming and have a clear picture its visualized in illustration 6. Certainly, with this visualization it's clearly that the model it's good cause the blue line and green line which are the lines that we predict with the model of application of GP

have a small gap compare to the black line, which is the original data. This gap represents the little deviation that has the model using Genetic Programming.

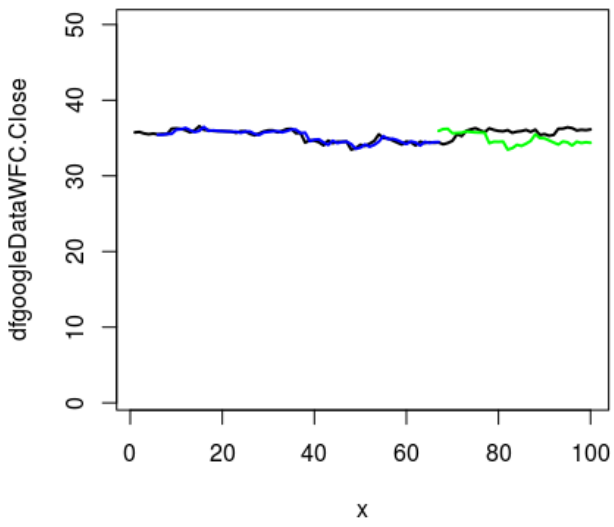


Illustration 6- Visualization of original data and predict data

III. 2-Analysis of the results of the GP with different scenarios

Indeed, in this case the predicting power of the application of genetic programing is good cause have lower root mean square error. However, perhaps the application of genetic programming can be sensible to his key parameters like windows and iterations.
It's performed different scenarios in order to evaluate how sensible the application of Genetic Programming is to his parameters with the overall fitness value.

One key parameter is iterations and can be sensible on the predict power/overall fitness value of the application of genetic programming. Thus, it's performed a second scenario with 200 seconds/iterations instead of 100 seconds. As result it's found that with 200 seconds compare with 100 seconds of iterations the root mean square error increase. Therefore, it's suggest according to this scenario to run the application of the genetic programming shorter than 200 seconds of iterations to have a better predictive model cause the root mean square error (on the test set) increase from 1.465386 to 1.53884 . This result is unusual cause it's view that if you run the iterations longer the smaller the root mean square error. Thus, after a range of iterations (it can be between 100seconds-200seconds) it reduces the root mean square error but after that range of 200 seconds root mean square error start it to increase by not that much and some of the cases stay constant. It might be happen because Genetic programming process already find the best solution and trying to find another better solution which not exit means the root mean square error increase. Thus, it's recommend to choose the number of iterations between that range. In the Illustrative 7 we can see the results of 200 seconds of iterations and how the root mean square error increase compare to 100 seconds of iterations.

Second Scenario
Key parameters
Population size=100 daily stock prices of Well Fargo
Window size=5
Training data-(2/3*100)=66 daily stock prices
Test data-(1/3*100)=33daily stock prices
Iterations=200 seconds

```
> gpResult1 <- geneticProgramming(functionSet=functionSet1,
+                         inputVariables=inputVariables,
+                         constantSet=constantFactorySet1,
+                         fitnessFunction=fitnessFn,
+                         stopCondition=makeTimeStopCondition(200))
STARTING genetic programming evolution run (Age/Fitness/Complexity Pareto GP search-heuristic) ...
evolution step 100, fitness evaluations: 4950, best fitness: 0.364192, time elapsed: 3.2 seconds
evolution step 200, fitness evaluations: 9950, best fitness: 0.364192, time elapsed: 6.27 seconds
evolution step 300, fitness evaluations: 14950, best fitness: 0.364192, time elapsed: 9.39 seconds
evolution step 400, fitness evaluations: 19950, best fitness: 0.364192, time elapsed: 12.46 seconds
evolution step 500, fitness evaluations: 24950, best fitness: 0.364192, time elapsed: 15.47 seconds
evolution step 600, fitness evaluations: 29950, best fitness: 0.364192, time elapsed: 18.59 seconds
evolution step 700, fitness evaluations: 34950, best fitness: 0.363709, time elapsed: 21.67 seconds
evolution step 800, fitness evaluations: 39950, best fitness: 0.363703, time elapsed: 24.78 seconds
evolution step 900, fitness evaluations: 44950, best fitness: 0.363342, time elapsed: 28.01 seconds
evolution step 1000, fitness evaluations: 49950, best fitness: 0.363141, time elapsed: 31.41 seconds
evolution step 1100, fitness evaluations: 54950, best fitness: 0.347983, time elapsed: 34.78 seconds
evolution step 1200, fitness evaluations: 59950, best fitness: 0.334442, time elapsed: 38.14 seconds
evolution step 1300, fitness evaluations: 64950, best fitness: 0.332920, time elapsed: 41.59 seconds
evolution step 1400, fitness evaluations: 69950, best fitness: 0.326777, time elapsed: 44.97 seconds
evolution step 1500, fitness evaluations: 74950, best fitness: 0.326541, time elapsed: 48.36 seconds
evolution step 1600, fitness evaluations: 79950, best fitness: 0.326541, time elapsed: 51.74 seconds
evolution step 1700, fitness evaluations: 84950, best fitness: 0.326515, time elapsed: 55.13 seconds
evolution step 1800, fitness evaluations: 89950, best fitness: 0.326515, time elapsed: 58.5 seconds
evolution step 1900, fitness evaluations: 94950, best fitness: 0.326515, time elapsed: 1 minute, 1.92 seconds
evolution step 2000, fitness evaluations: 99950, best fitness: 0.325191, time elapsed: 1 minute, 5.31 seconds
evolution step 2100, fitness evaluations: 104950, best fitness: 0.316096, time elapsed: 1 minute, 8.88 seconds
evolution step 2200, fitness evaluations: 109950, best fitness: 0.315194, time elapsed: 1 minute, 12.46 seconds
evolution step 2300, fitness evaluations: 114950, best fitness: 0.315194, time elapsed: 1 minute, 16.08 seconds
evolution step 2400, fitness evaluations: 119950, best fitness: 0.313742, time elapsed: 1 minute, 19.66 seconds
evolution step 2500, fitness evaluations: 124950, best fitness: 0.313742, time elapsed: 1 minute, 23.36 seconds
evolution step 2600, fitness evaluations: 129950, best fitness: 0.313556, time elapsed: 1 minute, 26.98 seconds
evolution step 2700, fitness evaluations: 134950, best fitness: 0.313408, time elapsed: 1 minute, 30.76 seconds
evolution step 2800, fitness evaluations: 139950, best fitness: 0.313408, time elapsed: 1 minute, 34.66 seconds
evolution step 2900, fitness evaluations: 144950, best fitness: 0.312939, time elapsed: 1 minute, 38.7 seconds
evolution step 3000, fitness evaluations: 149950, best fitness: 0.312939, time elapsed: 1 minute, 42.79 seconds
evolution step 3100, fitness evaluations: 154950, best fitness: 0.312917, time elapsed: 1 minute, 46.76 seconds
evolution step 3200, fitness evaluations: 159950, best fitness: 0.312917, time elapsed: 1 minute, 50.92 seconds
evolution step 3300, fitness evaluations: 164950, best fitness: 0.312873, time elapsed: 1 minute, 55.06 seconds
evolution step 3400, fitness evaluations: 169950, best fitness: 0.312873, time elapsed: 1 minute, 59.25 seconds
evolution step 3500, fitness evaluations: 174950, best fitness: 0.312873, time elapsed: 2 minutes, 3.46 seconds
evolution step 3600, fitness evaluations: 179950, best fitness: 0.312515, time elapsed: 2 minutes, 7.66 seconds
evolution step 3700, fitness evaluations: 184950, best fitness: 0.312515, time elapsed: 2 minutes, 11.97 seconds
evolution step 3800, fitness evaluations: 189950, best fitness: 0.312474, time elapsed: 2 minutes, 16.28 seconds
evolution step 3900, fitness evaluations: 194950, best fitness: 0.312474, time elapsed: 2 minutes, 20.71 seconds
evolution step 4000, fitness evaluations: 199950, best fitness: 0.312474, time elapsed: 2 minutes, 25.26 seconds
evolution step 4100, fitness evaluations: 204950, best fitness: 0.312348, time elapsed: 2 minutes, 29.87 seconds
evolution step 4200, fitness evaluations: 209950, best fitness: 0.312348, time elapsed: 2 minutes, 34.36 seconds
evolution step 4300, fitness evaluations: 214950, best fitness: 0.311869, time elapsed: 2 minutes, 38.67 seconds
evolution step 4400, fitness evaluations: 219950, best fitness: 0.311869, time elapsed: 2 minutes, 43.01 seconds
evolution step 4500, fitness evaluations: 224950, best fitness: 0.311504, time elapsed: 2 minutes, 47.64 seconds
evolution step 4600, fitness evaluations: 229950, best fitness: 0.311316, time elapsed: 2 minutes, 52.18 seconds
evolution step 4700, fitness evaluations: 234950, best fitness: 0.311316, time elapsed: 2 minutes, 56.74 seconds
evolution step 4800, fitness evaluations: 239950, best fitness: 0.311273, time elapsed: 3 minutes, 1.42 seconds
```

```
evolution step 4900, fitness evaluations: 244950, best fitness: 0.311273, time elapsed: 3 minutes, 6.12 seconds
evolution step 5000, fitness evaluations: 249950, best fitness: 0.311273, time elapsed: 3 minutes, 10.49 seconds
evolution step 5100, fitness evaluations: 254950, best fitness: 0.310630, time elapsed: 3 minutes, 14.87 seconds
evolution step 5200, fitness evaluations: 259950, best fitness: 0.310630, time elapsed: 3 minutes, 19.17 seconds
Genetic programming evolution run FINISHED after 5218 evolution steps, 260850 fitness evaluations and 3 minutes, 20.01 seconds.
>
>
> best1 <- gpResult1$population[[which.min(gpResult1$fitnessValues)]]
> best1
function (x1, x2, x3, x4, x5)
x4/x5 + ((x5/x4 - (x3 - x3) - x1)/((x3 + (x2 - x3))/x5)/x3 +
   ((x3 - x1)/x5 + (-0.0161841374629282 + ((x2 - x5)/(((x2 -
      x4)/(x5 * 0.372129224357104) + ((x4 - x5)/6.10709347678305 +
      x5)) * 0.246106001577739) + (1.5567789703085/(0.0160947883050009/-1.56459820347619 -
      (-1.19923635196425 * x4 - 0.188506318981083) + (x3 +
      (x1/(x4 - x1) + x5))) + x5)))))
>
```

```
> rmse(trainingResults,trainingdata[5:66])
[1] 0.1831591

> rmse(predictions[1:33],testdata[1:33])
[1] 1.53884
```

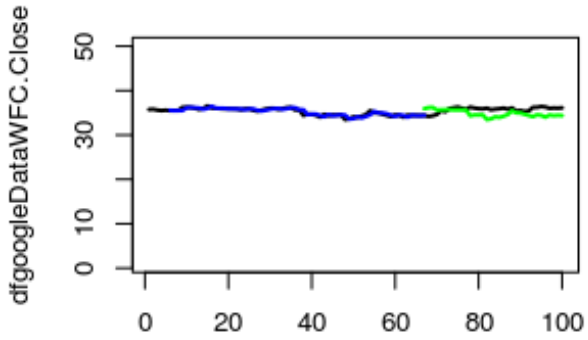


Illustration 7- All variables constants to the first scenario but 200 seconds iterations

In addition, other key parameter is the number of window it's chosen. For instance, its choose a third scenario of 10 windows in order to see how sensible is the application of genetic programming to this parameter leaving all other parameter constant. Evolving the process again with genetic programming it's found that with more 10 windows the predict power/overall fitness value increase because square root mean square error increase(on the test set) from 1.465386 to 1.501214. Therefore, the predict power it's sensible to the number of windows and is better to choose lower window size. In the Illustrative 8 we can see the results of 10 windows and how the root mean square error increase.

Third Scenario
Key parameters
Population size=100 daily stock prices of Well Fargo
Window size=10
Training data-(2/3*100)=66 daily stock prices
Test data-(1/3*100)=33daily stock prices
Iterations=100 seconds

```
gpResult1 <- geneticProgramming(functionSet=functionSet1,
+                         inputVariables=inputVariables,
+                         constantSet=constantFactorySet1,
+                         fitnessFunction=fitnessFn,
+                         stopCondition=makeTimeStopCondition(100))
STARTING genetic programming evolution run (Age/Fitness/Complexity Pareto GP search-heuristic) ...
evolution step 100, fitness evaluations: 4950, best fitness: 0.367282, time elapsed: 3.34 seconds
evolution step 200, fitness evaluations: 9950, best fitness: 0.367282, time elapsed: 6.53 seconds
evolution step 300, fitness evaluations: 14950, best fitness: 0.367282, time elapsed: 9.75 seconds
evolution step 400, fitness evaluations: 19950, best fitness: 0.367282, time elapsed: 12.98 seconds
evolution step 500, fitness evaluations: 24950, best fitness: 0.367282, time elapsed: 16.15 seconds
evolution step 600, fitness evaluations: 29950, best fitness: 0.367282, time elapsed: 19.27 seconds
evolution step 700, fitness evaluations: 34950, best fitness: 0.367282, time elapsed: 22.39 seconds
evolution step 800, fitness evaluations: 39950, best fitness: 0.367282, time elapsed: 25.49 seconds
evolution step 900, fitness evaluations: 44950, best fitness: 0.367282, time elapsed: 28.63 seconds
evolution step 1000, fitness evaluations: 49950, best fitness: 0.367282, time elapsed: 31.83 seconds
evolution step 1100, fitness evaluations: 54950, best fitness: 0.367282, time elapsed: 34.99 seconds
evolution step 1200, fitness evaluations: 59950, best fitness: 0.367282, time elapsed: 38.18 seconds
evolution step 1300, fitness evaluations: 64950, best fitness: 0.367282, time elapsed: 41.28 seconds
evolution step 1400, fitness evaluations: 69950, best fitness: 0.367282, time elapsed: 44.45 seconds
evolution step 1500, fitness evaluations: 74950, best fitness: 0.367282, time elapsed: 47.59 seconds
evolution step 1600, fitness evaluations: 79950, best fitness: 0.367282, time elapsed: 50.71 seconds
evolution step 1700, fitness evaluations: 84950, best fitness: 0.367282, time elapsed: 53.83 seconds
evolution step 1800, fitness evaluations: 89950, best fitness: 0.367282, time elapsed: 56.94 seconds
evolution step 1900, fitness evaluations: 94950, best fitness: 0.367282, time elapsed: 1 minute, 0.06 seconds
evolution step 2000, fitness evaluations: 99950, best fitness: 0.365691, time elapsed: 1 minute, 3.26 seconds
evolution step 2100, fitness evaluations: 104950, best fitness: 0.365590, time elapsed: 1 minute, 6.47 seconds
evolution step 2200, fitness evaluations: 109950, best fitness: 0.365556, time elapsed: 1 minute, 9.66 seconds
evolution step 2300, fitness evaluations: 114950, best fitness: 0.365487, time elapsed: 1 minute, 12.78 seconds
evolution step 2400, fitness evaluations: 119950, best fitness: 0.365480, time elapsed: 1 minute, 15.83 seconds
evolution step 2500, fitness evaluations: 124950, best fitness: 0.360529, time elapsed: 1 minute, 18.98 seconds
evolution step 2600, fitness evaluations: 129950, best fitness: 0.355016, time elapsed: 1 minute, 22.17 seconds
evolution step 2700, fitness evaluations: 134950, best fitness: 0.355016, time elapsed: 1 minute, 25.39 seconds
evolution step 2800, fitness evaluations: 139950, best fitness: 0.354917, time elapsed: 1 minute, 28.72 seconds
evolution step 2900, fitness evaluations: 144950, best fitness: 0.350985, time elapsed: 1 minute, 31.96 seconds
evolution step 3000, fitness evaluations: 149950, best fitness: 0.350985, time elapsed: 1 minute, 35.23 seconds
evolution step 3100, fitness evaluations: 154950, best fitness: 0.350154, time elapsed: 1 minute, 38.47 seconds
Genetic programming evolution run FINISHED after 3149 evolution steps, 157400 fitness evaluations and 1 minute, 40.03 seconds.
>
>
> best1 <- gpResult1$population[[which.min(gpResult1$fitnessValues)]]
> best1
function (x1, x2, x3, x4, x5, x6, x7, x8, x9, x10)
x10 + -1.9536182729668/x10 + (x2 - x10)/(0.204366860017875 *
    x4)
>



> rmse(trainingResults,trainingdata[10:66])
[1] 0.1072225

> rmse(predictions[1:33],testdata[1:33])
[1] 1.501214
```

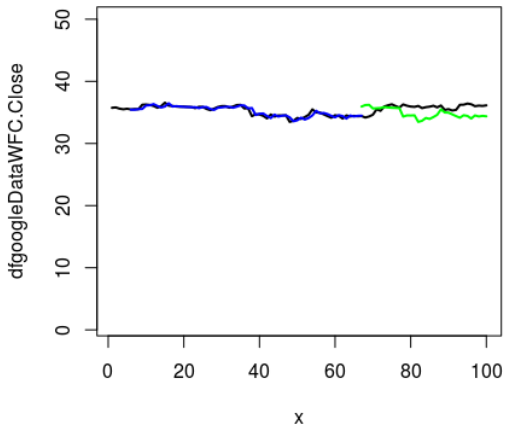
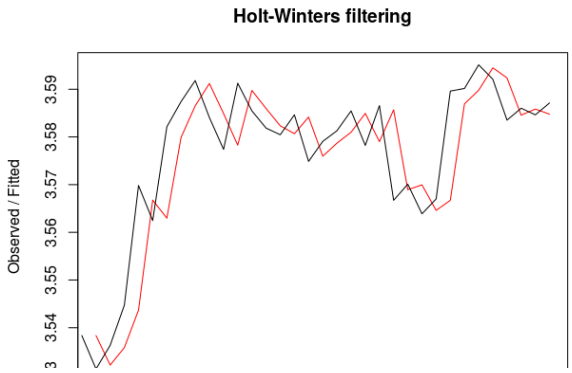


Illustration 8- All variables constant to the first scenario but 10 windows

In conclusion, the application of genetic programming is very sensible to his parameters especially with the window size and iterations. Therefore, if you want a good prediction model using Genetic Programming choose less widows size and a range of iterations between 100 and 200 seconds. As result, it's agree that the Genetic Programming application it's good model for forecasting if it's met the assumption of low volatility.

IV. Comparing predictive abilities of GP with other approaches

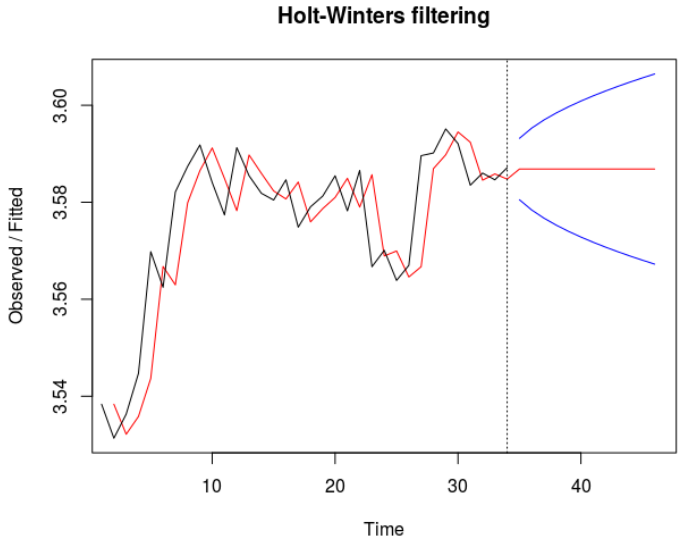
In conclusion, the predictive power of the application of GP it's turns to be powerful for predicting when the data set have low volatility. Hence, let's compare the predictive abilities of GP with other standard time series analysis techniques as Holt's Exponential Smoothing. It's choose Holt' exponential smoothing to compare it with Genetic Programming cause can it's used to make short term forecast for time series, which is this case, only 100 stock prices.
In order to decide if this model is appropriate we have evaluate if the fluctuations are constant overtime, I mean the seasonality. Thus, in the illustration 9, it's identified a constant fluctuation of the price of 3.5. Recall, the time of the stock prices is represent it on days, which is a total 33 days of the stock of 2007 because the model is performance over the test set. The addictive method, Holt's Exponential Smoothing is appropriate cause the data have a constant price fluctuating with log mean of 3.57. Finally, it's performed the log of this data as it's said it in the past; log the data is better for statistical analysis and prediction in order to be more normal distributed. Therefore, the output of the forecast using Holt's exponential smoothing on the test set is the following:



```
> mean(logtestdata)
[1] 3.576204

Call:
HoltWinters(x = logtestdata, beta = FALSE, gamma = FALSE)

Smoothing parameters:
 alpha: 0.8830074
 beta : FALSE
 gamma: FALSE

Coefficients:
      [,1]
a 3.586848

> logtestdataforecast$SSE
[1] 0.002914481
```

Illustration 9

As a result the .8830074 alpha means that the forecast are based more in the less recent observations. In order to analysis better the prediction it's plotted the comparison between my original data, which is the black line vs. forecast data as a red line. According to this plot we can imply that the forecast using Holt's Exponential Smoothing it has very small variation to the original data cause have small gap between the lines. Therefore, to be able to measure the accuracy of the forecast, it's calculate the sum of squared errors between this dataset, which is 0.002914481.




```
forecast <-predict(logtestdataforecast, n.ahead = 12, prediction.interval = T, level = 0.50)
plot(logtestdataforecast, forecast)
```

Illustration 10

Hence, in this case it does conclude that the application of Genetic Programing it's not better model for forecasting than Holt's Exponential Smoothing model. Because the root mean square error on Genetic Programming is between .09 and 1.5 and on Holt's exponential smoothing model have a sum square error of approximately 0.002914481. Thus, it's significantly more accurate model Holt's Exponential Smoothing because the sum square error is even smaller that the mean of the sum square error of Genetic Programming in the same dataset. This result indicates that the Holt's Exponential Smoothing is better approach for forecasting if the following assumptions are met: seasonality, low volatility and small data set (10-100 time series). This represents more predicting power on Holt's Exponential Smoothing approach on small financial data time series than Genetic Programming. Research by R. S. Soni and D. Srikanth (2017) reviews " the GP model simulated a time-sequence of item demand values on monthly basis for the period Feb 2005 to May 2015 as per the time-horizon of inventory dataset. Subsequently, Holt-Winters Exponential Smoothing Algorithm have been used to decompose the demand time-series into level, trend and seasonality components and compute item demand forecasts for a period of 24-months." In other words, this research suggests as well that for small data set it's used the Holt's Exponential Smoothing. Otherwise, if you do not meet these assumptions of seasonality and small data set it's suggest it a better approach Genetic Programming.

Furthermore, another advantage that it's better to choose Holt's exponential smoothing in forecasting short-term data series it's because you can identify a confidence interval (if meet the assumptions). For instance, in this case you can identify a range of the stock prices within a confidence interval. This is represent in Illustration 10 where Holt's exponential smoothing explain that on 50% range of the prediction model go right the stock prices is between from 3.56 to 3.60. Overall each method have their advantage and disadvantages. Therefore, your forecasting approach depends on the behavior of your data and if met the assumptions of each approach.

References:

Wells Fargo (2017) Available at: https://en.wikipedia.org/wiki/Wells_Fargo (Accessed: February 13, 2018)

The Analysis Factor (2018) Available at: https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models (Accessed: February 14, 2018)

R. S. Soni and D. Srikanth (2017) Inventory Forecasting Model Using Genetic Programming and Holt-Winter's Exponential Smoothing Method. Research. Defense Institute of Advanced Technology, DIAT (DU), Pune, India.

Appendix

```
#Assessment-Joshua Eick

#Instal packages
install.packages("emoa")
install.packages ("rgp")
install.packages("ggplot2")
install.packages("astsa")
install.packages("TTR")
install.packages('timeSeries')
library(emoa)
library(rgp)
library(ggplot2)
library(xts)
library(quantmod)
library(astsa)
library(forecast)
library(TTR)
library(timeSeries)

#Importing data:Well Fargo stock prices
install.packages("quantmod")
getSymbols("WFC",src="yahoo")
googleData <- WFC
dfgoogleData <- as.data.frame(googleData)
dfgoogleData
dfgoogleDataWFC.Close <- dfgoogleData[c(1:100),c(4)]
dfgoogleDataWFC.Close
dfgoogleDataWFC.Close



#Exploring the data

head(dfgoogleDataWFC.Close)
par(mfrow=c(2,2))
hist(dfgoogleDataWFC.Close)
hist(logsdfgoogleDataWFC.Close)
#log the data to reduce volatility
log10(dfgoogleDataWFC.Close)
logsdfgoogleDataWFC.Close
logsdfgoogleDataWFC.Close<-dfgoogleDataWFC.Close
summary(logsdfgoogleDataWFC.Close)
summary(dfgoogleDataWFC.Close)
sd(dfgoogleDataWFC.Close)
```

```
sd(logsdfgoogleDataWFC.Close)
chartSeries(WFC)

dfgoogleDatatimeseries <- ts(dfgoogleData, frequency=30, start=c(2007,1),end =
c(2007,4))
dfgoogleDatatimeseries
plot.ts(dfgoogleDatatimeseries)

#My data name
logsdfgoogleDataWFC.Close<-dfgoogleDataWFC.Close




#First Esenario 5 windows with  100 seconds of iterations

trainingdata<-dfgoogleDataWFC.Close[1:66]
testdata<-dfgoogleDataWFC.Close[67:100]
testdata
length(dfgoogleDataWFC.Close[62:66])
ws<-5 #weekly windowsize
inputVariables <- do.call(inputVariableSet, as.list(paste0("x",1:ws)))

# Slightly nicer way of doing this is to define a window size ws 30,40<- 4
# and then generate the input variables in the following way


functionSet1<-functionSet("+","*","-","/")
functionSet2<-functionSet("+","*","-","/","exp")
constantFactorySet1<- constantFactorySet(function()rnorm(1))

fitnessFn<-function(f){
  fn.result = NULL
  for(i in 1:61) {fn.result[i] = do.call(f,as.list(dfgoogleDataWFC.Close[i:(i+4)]))}
  fitness = rmse(fn.result, trainingdata[6:66])
  return (if (is.na(fitness)) Inf else fitness)}


gpResult1 <- geneticProgramming(functionSet=functionSet1,
                                inputVariables=inputVariables,
                                constantSet=constantFactorySet1,
                                fitnessFunction=fitnessFn,
                                stopCondition=makeTimeStopCondition(100))
```

```
best1 <- gpResult1$population[[which.min(gpResult1$fitnessValues)]]
best1

trainingResults<-NULL
for(i in 1:62) {
  trainingResults[i]<-do.call(best1,as.list(trainingdata[i:(i+4)]))}




# Look at the training results
trainingResults
trainingdata
# Can then compare trainingResults against trainingdata using rmse etc...
# or just look at the difference:
trainingResults - trainingdata[5:66]
rmse(trainingResults,trainingdata[5:66])

#to make sure the same amount of values
length(trainingResults)

length(trainingdata[5:66])


#Next, let's see what's it like as a predictor
# and then use predictions to make future predictions
# i.e. add the results of the prediction into the array
# so it is input for the next iteration of the loop
predictions<-trainingdata[33:66]
for(i in 1:6){
  predictions[i+5]<-do.call(best1,as.list(predictions[i:(i+4)]))}

predictions

# Can then compare predictionsagainst testdata
predictions[1:33]
testdata[1:33]

predictions[1:33] - testdata[1:33]

rmse(predictions[1:33],testdata[1:33])


# Now let's plot this...
# want to plot the original data, the training results and the predictions
```

```
# set up the x-axis based on the length of the data set
x<-1:100
# plot the original values

# of the initial plot then they will not be seen
par(mfrow=c(2,2))
plot(x, dfgoogleDataWFC.Close, ylim=c(1,50), type="l",  col = "black", lwd = 2)


# Now add in the training results
# we'll make them blue...

lines(x[6:67], trainingResults, col = "blue", lwd = 2)

# And finally the predictions in green
lines(x[67:100], predictions[1:34], col = "green", lwd = 2)
```

#Second Esenario 5 windows but 200 seconds of iterations

```
trainingdata<-dfgoogleDataWFC.Close[1:66]
testdata<-dfgoogleDataWFC.Close[67:100]
testdata
length(dfgoogleDataWFC.Close[62:66])

ws<-5 #weekly windowsize
inputVariables <- do.call(inputVariableSet, as.list(paste0("x",1:ws)))


# and then generate the input variables in the following way
# inputVariables <- do.call(inputVariableSet, as.list(paste0("x",1:ws)))
# starting with the past function and working outwards.

functionSet1<-functionSet("+","*","-","/")
functionSet2<-functionSet("+","*","-","/","exp")
constantFactorySet1<- constantFactorySet(function()rnorm(1))

fitnessFn<-function(f){
  fn.result = NULL
  for(i in 1:61) {fn.result[i] = do.call(f,as.list(dfgoogleDataWFC.Close[i:(i+4)]))}
  fitness = rmse(fn.result, trainingdata[6:66])
  return (if (is.na(fitness)) Inf else fitness)}


gpResult1 <- geneticProgramming(functionSet=functionSet1,
                          inputVariables=inputVariables,
                          constantSet=constantFactorySet1,
                          fitnessFunction=fitnessFn,
                          stopCondition=makeTimeStopCondition(200))


best1 <- gpResult1$population[[which.min(gpResult1$fitnessValues)]]
best1

trainingResults<-NULL
for(i in 1:62) {
  trainingResults[i]<-do.call(best1,as.list(trainingdata[i:(i+4)]))}




# Look at the training results
trainingResults
trainingdata
# Can then compare trainingResults against trainingdatausing rmse etc...
```

```
# or just look at the difference:
trainingResults - trainingdata[5:66]
rmse(trainingResults,trainingdata[5:66])

#to make sure the same amount of values
length(trainingResults)

length(trainingdata[5:66])


#Next, let's see what's it like as a predictor
# and then use predictions to make future predictions
# i.e. add the results of the prediction into the array
# so it is input for the next iteration of the loop
predictions<-trainingdata[33:66]
for(i in 1:6){
  predictions[i+5]<-do.call(best1,as.list(predictions[i:(i+4)]))}

predictions

# Can then compare predictions against testdata
predictions[1:33]
testdata[1:33]

predictions[1:33] - testdata[1:33]

rmse(predictions[1:33],testdata[1:33])


# Now let's plot this...

x<-1:100
# plot the original values

par(mfrow=c(2,2))
plot(x, dfgoogleDataWFC.Close, ylim=c(1,50), type="l",  col = "black", lwd = 2)


# Now add in the training results
# we'll make them blue...

lines(x[6:67], trainingResults, col = "blue", lwd = 2)
# And finally the predictions in green
lines(x[67:100], predictions[1:34], col = "green", lwd = 2)
```

```
#Third Esenario 10 windows but 100 seconds of iterations

trainingdata<-dfgoogleDataWFC.Close[1:66]
testdata<-dfgoogleDataWFC.Close[67:100]
testdata
length(dfgoogleDataWFC.Close[62:66])

ws<-10 #weekly windowsize
inputVariables <- do.call(inputVariableSet, as.list(paste0("x",1:ws)))



# inputVariables <- do.call(inputVariableSet, as.list(paste0("x",1:ws)))


functionSet1<-functionSet("+","*","-","/")
functionSet2<-functionSet("+","*","-","/","exp")
constantFactorySet1<- constantFactorySet(function()rnorm(1))

fitnessFn<-function(f){
  fn.result = NULL
  for(i in 1:56) {fn.result[i] = do.call(f,as.list(dfgoogleDataWFC.Close[i:(i+9)]))}
  fitness = rmse(fn.result, trainingdata[11:66])
  return (if (is.na(fitness)) Inf else fitness)}


gpResult1 <- geneticProgramming(functionSet=functionSet1,
                                inputVariables=inputVariables,
                                constantSet=constantFactorySet1,
                                fitnessFunction=fitnessFn,
                                stopCondition=makeTimeStopCondition(100))


best1 <- gpResult1$population[[which.min(gpResult1$fitnessValues)]]
best1

trainingResults<-NULL
for(i in 1:57) {
  trainingResults[i]<-do.call(best1,as.list(trainingdata[i:(i+9)]))}
```

```
# Look at the training results
trainingResults
trainingdata
# Can then compare trainingResults against trainingdata using rmse etc...
# or just look at the difference:
trainingResults - trainingdata[10:66]
rmse(trainingResults,trainingdata[10:66])

#to make sure the same amount of values
length(trainingResults)

length(trainingdata[10:66])


#Next, let's see what's it like as a predictor

predictions<-trainingdata[33:66]
for(i in 1:11){
  predictions[i+10]<-do.call(best1,as.list(predictions[i:(i+9)]))}

predictions

# Can then compare predictions against testdata
predictions[1:33]
testdata[1:33]

predictions[1:33] - testdata[1:33]

rmse(predictions[1:33],testdata[1:33])


# Now let's plot this...

x<-1:100
# plot the original values

par(mfrow=c(2,2))
plot(x, dfgoogleDataWFC.Close, ylim=c(1,50), type="l",  col = "black", lwd = 2)


# Now add in the training results -
# we'll make them blue...
```

```
lines(x[11:67], trainingResults, col = "blue", lwd = 2)
length(1:33)
# And finally the predictions in green
lines(x[67:100], predictions[1:34], col = "green", lwd = 2)
```

Comparing with another approaches
#Holtzwinters predictive model

```
install.packages("quantmod")
install.packages("timeSeries")
install.packages('emoa')
install.packages('TTR')
install.packages('smooth')
install.packages('Mcomp')
install.packages('fGarch')
install.packages('astsa')
install.packages('tidyquant')
install.packages('rmeta')
library(emoa)
library(timeSeries)
library(quantmod)
library(rgp)
require(graphics)
library(TTR)
library(smooth)
library(Mcomp)
library(fGarch)
library(astsa)
library(tidyquant)
library(ggplot2)
library(forecast)
```

```
library(rmeta)

testdata<-dfgoogleDataWFC.Close[67:100]
mean(testdata)
plot.ts(testdata)
log(testdata)
logtestdata<-log(testdata)
plot.ts(testdata)
plot.ts(logtestdata)
mean(logtestdata)
logtestdataforecast<-HoltWinters(logtestdata,beta = FALSE,gamma = FALSE)
logtestdataforecast
plot.ts(testdata)
plot(logtestdataforecast)



forecast <- predict(logtestdataforecast, n.ahead = 12, prediction.interval = T, level = 0.50)
plot(logtestdataforecast)



#Analysis of the Hots approach
logtestdataforecast$SSE
logtestdataforecast$fitted
logtestdataforecast$mean
```