Brandon Toppins

Genetic Algorithm: Finding a Stock Market Rule

Introduction:

The project tasked Computer Scientists to create an algorithm to find a single rule to be used in the stock market. The rule is calculated from a genetic algorithm approach and the rule is considered the individual during the process. The rules are created and evaluated over a daily time span. For the genetic algorithm, I used .csv ranging from 1980 to 2018. This allowed me to experiment with a wide variety of dates until I finished creating the genetic algorithm. The experiment part required 30 historical pieces for comparison against the rule generated from the genetic algorithm. I chose to limit this timespan from 2005 to 2018, so that more companies were available for comparison. Keeping companies consistent in time across the 30 historical pieces is important, because stock market changes can affect all the companies in similar or widely different ways. So, keeping a constant in place was important.

The project created strict parameters for the programming of the genetic algorithm. The genetic algorithm followed a sequence of generating 200 individuals and having an initial population of 20. The 20 was created using Java's random number generator(RNG), as well as all probabilities and chance decisions in the actual algorithm use the same RNG. The Java RNG is not perfect, but it allowed for enough randomness to create an environment to run the genetic algorithm in. Aspects of the process like the luck variable for crossover and the chance of mutation were set in the project description.

The project is comprised of two major portions. Generating a rule and then testing that rule. The generating a rule is done in the genetic algorithm and the testing is done during the experiment.

Genetic Algorithm:

The genetic algorithm is comprised of five java files. The java files are: Main.java, CSVReader.java, Individual.java, Company.java, and Genetic.java.

The Main file only calls the genetic algorithm to start by creating the object and passing in five .csv files with historical data and then calling a function call startTrading().

CSVReader.java is a generic .csv reading class. It uses a scanner to read a path of a file and then appends that data to a data ArrayList. Since, the only two pieces of data needed for calculating the rules is closing price and data, two functions for getting an ArrayList of closing prices and dates was made.

Individual.java houses all the information for a single individual in the population. The class holds the information for what the actual rule is and the fitness of the rule. The class is also setup to calculate a buy or sell situation. So, the class will evaluate the rule and return whether or not, on a given day and closing price, if the individual should buy or sell. Buying and selling is set up as booleans in the class and if the function checkBuy() evaluates to true, then a buy call is sent, else a sell call is sent.

The Company.java class houses the information for the company and I also chose to house the money spent and gained in this class. So the company file instantiates two accounts of money. One account is the money pool which always has $20,000 or less in it, while there is also a gained account which stores any extra money made over the initial 20,000. Everytime a stock is bought, a stock counter goes up and for consistency only one stock a day is bought. The class handles all the actual transactions based upon the call from the Individual rule calculation and checkBuy() function.

The Genetic.java is the main file that runs the algorithm utilizing the above classes. The file starts at a startTrading() function and runs a loop over 200 generations adding to a running population that is never discarded. The function assigns a fitness to every rule generated and uses that fitness to create a roulette wheel for selecting parent rules to generate new child rules for the population. There is a luck variable created to have the chance of crossover not happening. This variable simulates a new roulette wheel spin and creates a wider sample space for the population to be generated from. Inside the crossover section. Crossover happens twice and a chance of mutation happens once each for the two children generated. After the crossover and mutation both children are added to the population and fitness is re-evaluated as well as a new roulette wheel. The generation is only incremented if crossover happened.

Once the genetic algorithm starts running, the process takes some time to learn the optimal rule. Brute forcing the coding for the genetic algorithm yields large runtime and space as each iteration the fitness function is called once or twice and the fitness always runs through the entire population across a two year time span. I set the start date to November 21, 1984 and the end date to October 14, 1987. The start date is exactly 999 days after the February 14, 1980 and the end date is 730 days after the start date. The inconsistency in an exact two year difference between the dates is because of the days the stock market is closed across a two year span. This time span is used across all five companies imported into the genetic algorithm and for each rule.

Fitness is calculated based off earnings or loss during the rules runtime across the two year span. During the active period of when the rule is being evaluated, each day a buy or sell call is made. If neither a buy or sell is made a penalty is calculated. The penalty is halving the current money amount. This penalty does not exist in the experiment portion. The penalty is only in place to help the genetic algorithm find an optimal rule that will definitely make money and not converge to a rule that never loses money or gain money because it never buys or sells. Earnings made pass the initial $20,000 money pool is added to a gain account. At the end of the process all gains and money is summed up to create a fitness value for the rule. Rules that make money will have a higher fitness than rules that loss money or made less money. So the rules with the highest fitness will always be the rules that made the most money.

The roulette is calculated by summing all the fitness values in the population of rules and dividing each fitness by that sum. The selection uses this percentage to give individuals with high fitness a better chance of being selected than individuals with low fitness. The selected individuals are then assigned to a pseudo male and female variable for the crossover section.

The crossover section an 80% chance of happening each iteration. The 80% is hard-coded into the genetic algorithm process and allows for a chance for multiple roulette wheel spins to happen before creating new children. The children then also have a a 0.001 chance of mutation, which changes one character inside the rule to a random character of similar meaning. The crossover section ends with another fitness evaluation and roulette wheel. If crossover happens then the generation counter is incremented by 1.

After the crossover section, the genetic algorithm states what generation it is on and the current most fit rule before starting the next iteration.

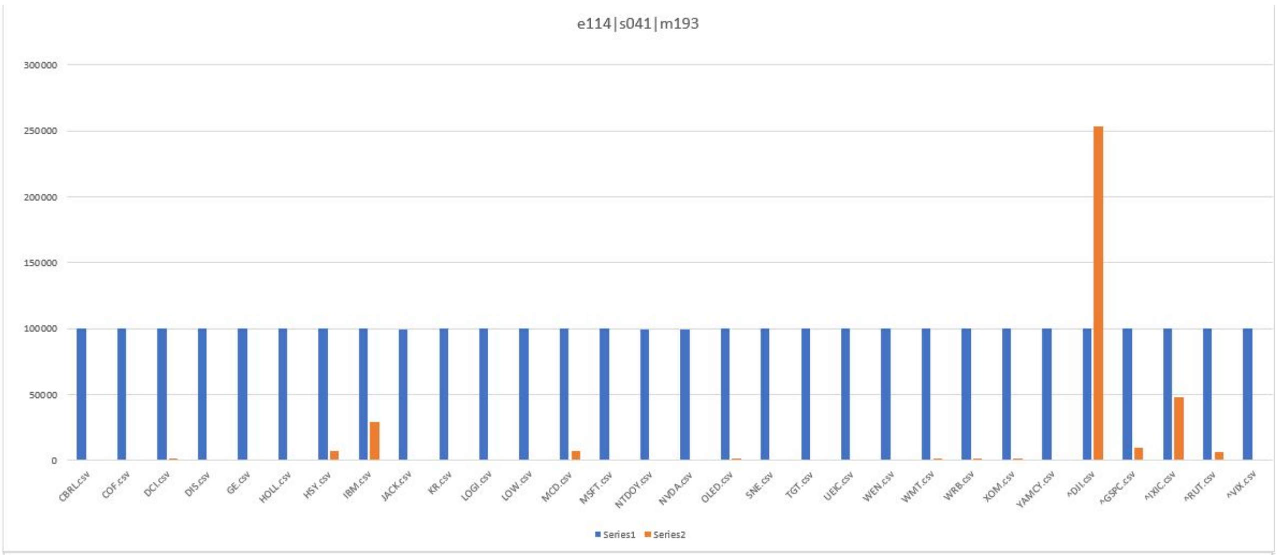
Experiment:

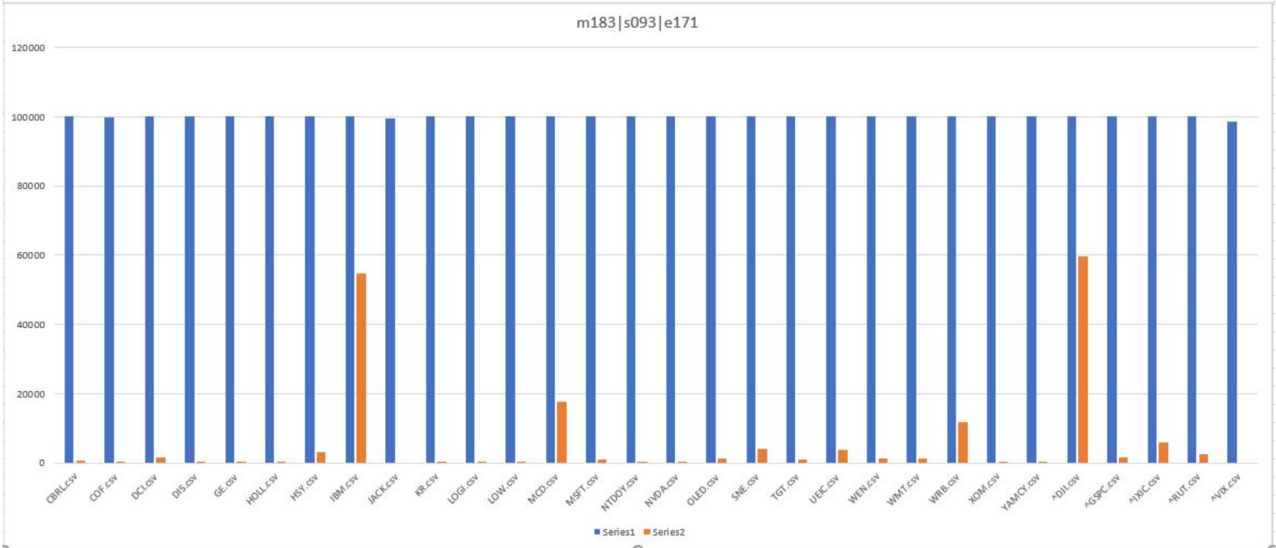
For the experiment, I generated three rules over the two year timespan. I changed the start date for the experiment to February 3, 2009 and the .csv files start at 2005. This was a necessary change as finding 30 companies that have lasted from 1980 to modern day was short in supply. The historical data ranges from companies such as Disney and Kroger to companies like DJI, which is a Chinese technology company.

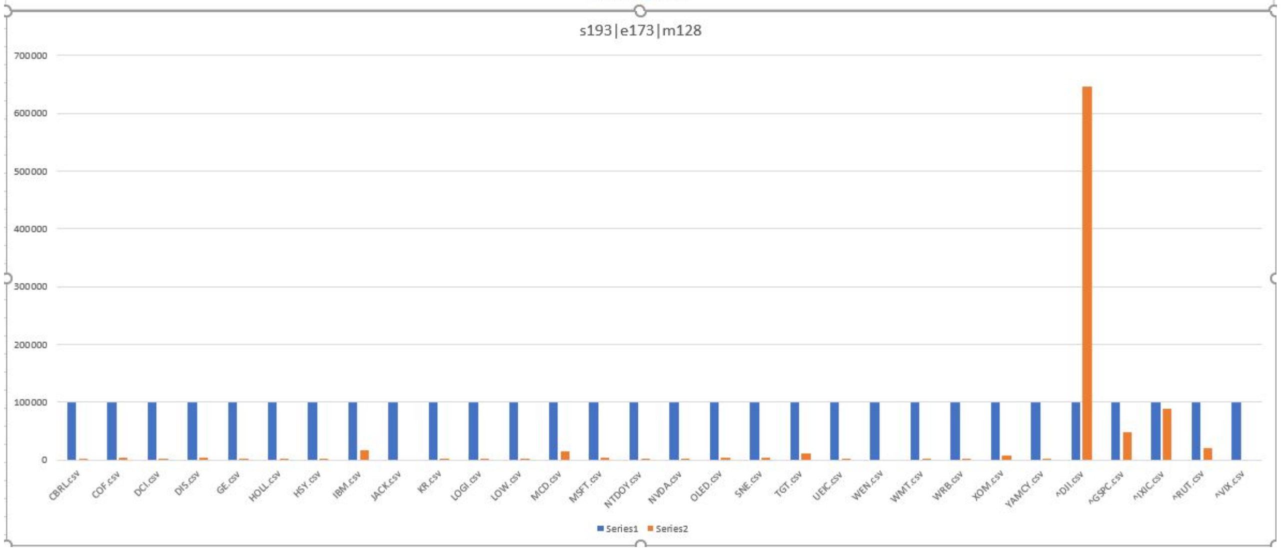
Rules: e114|s041|m193
m183|s093|e171
s193|e173|m128

When running the experiment I created individual objects that contained a rule like in the genetic algorithm and created a population of three. The three individuals are the above rules generated from the genetic algorithm. After that I ran the trading exactly like the fitness function. The Experiment.java iterates over the population of rules and then iterates over the .csv that are contained within an array. Each .csv file is imported into a new Company object and that handles retrieving the data. After that a checkBuy() call is made daily across the two years like in the fitness function. The penalty from the Genetic Algorithm is commented out and the starting pool for each rule into a single company is now $100,000 instead of $20,000.

The results show mostly a positive outcome for the rules. Three line charts are included below to show the result for the 30 companies.


e114|s041|m193
300000
250000
200000
150000
100000
50000
0
CBRL.csv
COF.csv
DCI.csv
DIS.csv
GE.csv
HOLL.csv
HSY.csv
IBM.csv
JACK.csv
KR.csv
LOGI.csv
LOW.csv
MCD.csv
MSFT.csv
NTDOY.csv
NVDA.csv
OLED.csv
SNE.csv
TGT.csv
UEIC.csv
WEN.csv
WMT.csv
WRB.csv
XOM.csv
YAMCY.csv
^DJI.csv
^GSPC.csv
^IXIC.csv
^RUT.csv
^VIX.csv
Series1
Series2


m183|s093|e171
120000
100000
80000
60000
40000
20000
0
CBRL.csv
COF.csv
DCI.csv
DIS.csv
GE.csv
HOLL.csv
HSY.csv
IBM.csv
JACK.csv
KR.csv
LOGI.csv
LOW.csv
MCD.csv
MSFT.csv
NTDOY.csv
NVDA.csv
OLED.csv
SNE.csv
TGT.csv
UEIC.csv
WEN.csv
WMT.csv
WRB.csv
XOM.csv
YAMCY.csv
^DJI.csv
^GSPC.csv
^IXIC.csv
^RUT.csv
^VIX.csv
Series1
Series2


s193|e173|m128
700000
600000
500000
400000
300000
200000
100000
0
CBRL.csv
COF.csv
DCI.csv
DIS.csv
GE.csv
HOLL.csv
HSY.csv
IBM.csv
JACK.csv
KR.csv
LOGI.csv
LOW.csv
MCD.csv
MSFT.csv
NTDOY.csv
NVDA.csv
OLED.csv
SNE.csv
TGT.csv
UEIC.csv
WEN.csv
WMT.csv
WRB.csv
XOM.csv
YAMCY.csv
^DJI.csv
^GSPC.csv
^IXIC.csv
^RUT.csv
^VIX.csv
Series1
Series2

From the data, the blue lines show a consistency in retaining the initial $100,000. There are some losses, notably ^VIX, JACK, and WEN stocks, which had zero profit for all three rules. For the other 27 companies, all of them created profit for the rules. Granted, each company generated varied results of profit.

The most apparent statistic across all three graphs is the ^DJI stock. This stock is for the company DJI, which is a Chinese technology company which specializes in drones. This stock had a gain of $645484.07 for the rule “s193|e173|m128”, which is by far the largest gain across all 30 historical pieces and the three rules. This large profit also explains the distortion in the third graph, as a over half a million dollar number dominates the other more uniform statistics. The second highest profit amount, that also came from another company, came from IBM’s stock.

The three rules produced good results for most companies in the pool of 30 historical data. The situations where the rules experienced loss were minor and the greatest loss happened in the NTDOY stock for rule “e114|s041|m193”; the final money pool was $99200.19. The NTDOY stock had gains for other rules, so the data was not consistent for the stock across all three rules, but that a single exception compared to the other rules which retained consistency.

Conclusion:

The genetic algorithm written was brute forced and had few optimization programmed in. For instance, the fitness evaluation, always iterates over the population. However, the fitness never changes and no individuals are deleted from the population. Iterating is unnecessary time spent in this case then. So, in hindsight, many improvements can be made for the genetic algorithm, which could help speed up the process of finding an optimal rule for the stock market.

The genetic algorithm is a randomized and a luck based path to finding a rule that can be used to net profit on the stock market, where the randomness is guided by fitness values and crossover. Research into which stocks are good is still necessary, as even good rules can experience loss in some markets. Time span matter as well. For the genetic algorithm and the experiment I ran a two year time span. Most of the stocks made decent profits somewhere between \$20 to \$15,000, with stocks like ^DJI distorting the statistics with extreme profits over the two years. Two years was a decently long time span and the results yielded consistent profit for most companies, but longer time spans could yield more consistent profit or in some cases more loss.

The money and code in this project was all simulated and the stock market still appears to be volatile to unexpected measures such as nature and international relations. With that disclaimer stated, the experiment demonstrated success for a genetic algorithm being used to help play the stock market and could be considered a viable way to increasing someone’s luck, but is not a guarantee profit maker.