lk350

# COMP6560 Assessment Report

## How Much I Managed to Achieve

I managed to achieve all the tasks that were given to me, however, there are some aspects of my code that I feel like could have been done more efficiently and produced better results.

## Model

The first part of the assignment consisted of creating a model, that would be used as the input data for the genetic algorithm. For this I created functions that would calculate the SMA, EMA, TBR, VOL and MOM for the majority of the price data in the excel file. I used a 50-day period for all my calculations, and a 14-day for a shorter-term calculations, such as a shorter SMA or EMA. There was some price data I didn't consider, such as some of the days at the start of the file, due it not fitting within my 50-day calculations, and the last 14 days which didn't have a value to indicate whether it went up or not.

I used these tests on the data using technical indicators: SMA_14 > SMA_50, EMA_14 > EMA_50, EMA_50 < Price_X, TBR_50 <= 0.00, VOL_50 <= 0.03, MOM_50 > 0

## Implementation (Genetic Algorithm)

The data is then passed onto my genetic algorithm. My genetic algorithm finds a well performing rule which was trained on 70% of the input data, and then its tested on the other 30% of the data to see how accurate it is in predicting whether the price will go up or down in 14 days.

I used a population size of 100 individuals, with each individual being represented as an array of 13 bits, 2 bits for each test (one to indicate TRUE, one to indicate FALSE) and 1 bit at the end for the prediction. My input data is also transformed into this format at the beginning of the algorithm. I also split the data into two separate ArrayLists, one with my training data, and one with my testing data, and then after the population is randomly generated, it evaluates the initial fitness of each individual using sequential covering.

I have also set the max generation to 10, which will loop and evolve my rules. In each loop of the algorithm, it will randomly choose whether it will mutate one individual or crossover two individuals based on a probability (80% for crossover, 20% for mutation). I used a two-point crossover, and a two-point mutation as well, with each parent being selected using a tournament selection method. At the end of the loop, it will store the new evolved population and calculate the new fitness. The best rule found after the max generation will then be tested against the test data to produce an accuracy.

## What Went Well / Didn't Go Well

I found it quite easy implementing most parts of the genetic algorithm, such as the crossover/mutation functions, as well as the tournament selection method. I also found getting my head around the overall flow of the genetic algorithm simple, leading to an easier time creating the structure of the algorithm.

However, I was having trouble implementing sequential covering into my algorithm and creating the fitness function. It took me a while to figure out how a rule would cover an instance, and how that comes into play when calculating the fitness. Another thing that didn't go well is the accuracy produced for the well performing rule. My implementation doesn't give me an accuracy above 50%, which means it would need further optimisation.

## Report on Experimental Results:

I ran the program 15 times, and these were the result rules that came up:

| Rule | Instances Covered | Accuracy (out of 1) | Number of Times it Appeared |
|---|---|---|---|
| 1010101011110 | 97 | 0.46 | 11 |
| 1010101111100 | 111 | 0.45 | 3 |
| 1011101011110 | 100 | 0.45 | 1 |

As you can see, the first rule was produced the majority of the time, with a few times where another rule came up which was not quite as accurate in testing. Those rules that appeared less times actually cover more instances, with only being slightly less accurate.

## Report on Different GA Parameters

**bits**: I used 13 bits as that is what was needed to represent my individuals so it would also work with my fitness functions using sequential covering.

**populationSize:** I used a population size of 100. I think this is the most optimal population size as it gives the algorithm enough space to evolve properly. When I lowered the size, I saw a loss in accuracy of the rules produced, and I didn't see any difference in results for any size above 100.

**maxGeneration:** I used a max generation of 10 as I think its not too low so that it doesn't give the individuals the chance to evolve properly, resulting in lower accuracies, and because I never saw a difference in results when I made it any higher. I tried with a max generation of 50, but I was still getting the same results majority of the time.

**crossoverProbability/mutationProbability:** I kept my crossover/mutation probability to 80%/20%, which I believe is the best probability for my genetic algorithm. I didn't see much change in the result if I gave the mutation a greater probability, and I would want my individuals to crossover more than they mutate in general producing more random offspring's, for a better evolution.

**tournamentSize:** I used a tournament size of 7, as I thought this was a good portion of the population to randomly select. I never saw much of a change when making it lower/higher.

**training:** this value indicated how much of the input data would be taken for training. I took 70% of my data for training and tested the result with the remaining 30% of it. When I give less data for training, I end up getting a slightly lower average of accuracies, and the same goes for when I give more data for training.

## Other Useful Functions

Some other useful functions I have are for transforming/splitting the input data, first transforming it into the same format as my individual representation and splitting it for training/testing. I also wrote a function that writes out my rule in text form instead of binary, and another function that calculates the accuracy of the rule given by the GA by running the rule over the test data.

## What I Would've Done Differently

If I were to write this assignment again, I would first put more research into my tests on the price data. Most of my tests have an above 50% accuracy, but some still lie on 50%. If all my tests had a bit greater accuracy, or if I had more tests, it would give me better results. Another thing I would do given more time would be to focus on optimising my GA. Currently, the rules produced never have an accuracy above 50% on the test data. This isn't a good result, as the rule doesn't predict the correct value the majority of the time. However, although the accuracy isn't very high, it still produces the rule with the best accuracy from the population, but the code just needs some work to produce more optimal results.