PRIFYSGOL
BANGOR
UNIVERSITY

School of Computer Science and Electronic Engineering
College of Environmental Sciences and Engineering

Producing an optimum stock portfolio using Web Scraping, Information Extraction and Genetic Algorithms

Benjamin David Winter

Submitted in partial satisfaction of the requirements for the
Degree of Master of Science
in Computer Science

Supervisor Dr. William Teahan

January 2019

Contents

1 Acknowledgements **1**

2 Abstract **2**

3 Introduction **3**

3.1 Background and Motivation . . . 3

3.2 Aim and Objectives . . . 5

3.3 Summary of Dissertation . . . 6

4 Literature Review **8**

4.1 Overview of Stock Portfolios . . . 8

4.1.1 Automatic/Algorithmic stock trading . . . 10

4.2 Harvesting stock data . . . 11

4.3 Processing web data . . . 12

4.4 Portfolio selection using computational methods . . . 16

4.4.1 Fuzzy portfolios . . . 20

4.5 Summary and Discussion . . . 22

5 Design and Implementation **24**

5.1 Overall Design . . . 24

5.2 Design Objectives . . . 27

5.3 Technologies . . . 29

5.3.1 Java . . . 29

5.3.2 jsoup . . . 29

5.3.3 Selenium . . . 29

5.4 Classes . . . 30

5.5 Implementation . . . 31

5.5.1 Web scraping and Host Crawling for business news articles . . . 31

5.5.2 Text Preprocessing to remove stopwords, capitalisation and punctuation . . . 32

5.5.3 Named Entity Recognition on the extracted articles . . . 34

5.5.4 Sentiment Analysis of the extracted articles using AFINN-111 35

5.5.5 Developing the Second Web Scraping/Host Crawling Prototype 36

5.5.6 Selecting the optimum portfolio using a Genetic Algorithm . . 37

5.5.7 Creating the fitness function . . . 47

5.5.8 Selection 51
5.5.9 Crossover 52
5.5.10 Mutation 56
5.6 Discussion 57

6 Evaluation 58
6.1 Experiment 1 - Investigating different Crossover techniques 59
6.2 Experiment 2 - Comparing GenStock vs Manual portfolio selection . . 63
6.2.1 Comparing Time 63
6.2.2 Comparing Accuracy 64
6.3 Experiment 3 - Investigating different Crossover rates 66
6.4 Experiment 4 - Investigating different Mutation rates 67
6.5 Discussion Of Evaluation Experiments 68

7 Conclusion 69
7.1 Summary of Dissertation 69
7.2 Review of aims and objectives 70
7.3 Limitations 72
7.4 Future Work 73

References 75

A Appendix 81
A.1 P.E.S.T.L.E Analysis 81
A.2 Political Factors 81
A.3 Economic Factors 82
A.4 Sociological Factors 83
A.5 Technological Factors 83
A.6 Legal Factors 84
A.7 Environmental Factors 84
A.8 Code 85
A.8.1 Normalise 85
A.8.2 Selection 86
A.8.3 One-Point Crossover 86
A.8.4 Two-Point Crossover 87
A.8.5 Mutation 87
A.8.6 Fitness 88

List of Figures

3.1 The workflow of the program to create an optimum portfolio selection. . 7

4.1 The Efficient Frontier designed by Harry Markowitz. The Efficient Frontier is a way to measure risk and reward of a portfolio while removing less profitable portfolios. The dots that lie on the solid blue line show the most efficient portfolios, one giving a slightly less return but for a dramatically decreased chance of risk, whilst the other portfolio offers the opposite. All further portfolios that lie below the solid blue line should be disregarded [40]. 10

4.2 The equation used by Yahaya to determine a portfolios fitness based on Harry Markowitz' Mean-Variance model. p is each portfolio, n is the number of assets in the portfolio, w_i is the proportion of available funds allocated to asset i, Q_ij is the real-valued covariance of asset i with asset j. 18

4.3 Single-Point Crossover [48] 19

4.4 Two-Point Crossover [49] 19

4.5 Whole Arithmetic Recombination [23] 19

5.1 UML diagram describing how each class interacts with each other, eventually creating an optimum portfolio. The thin boxes containing the words: 'Hub', 'NLP', 'HistoricalDataInput', 'GeneticAlgorithm', depict the names of the classes responsible for each part of the program. The black dot represents the start of the program. 26

5.2 This figure shows part of one scraped article saved into a .txt file with a TimeStamp. 32

5.3 An individual portfolio created by the method individual(). 0.01 represents that the user should place 1% of their total budget on company 1, 0.08 represents the investor should place 8% of their total investment budget on company 2 and so on. 42

5.4 The return % and risk % of one particular run. 42

5.5 Whole-Arithmetic Crossover of two parents, combining their genetic information and reproducing to create two children. Gene 1 for Parent 1 is equal to 0.1, whilst Gene 1 for Parent 2 equals 0.3. The median between these two Genes is 0.2 therefore the genetic information for Gene 1 on both offspring is equal to 0.2. [28]. 55

5.6 The final output of the program, showing the company names, the fitness of the fittest individual, the weights of the portfolio as well as the risk and return percentage of the portfolio. The output showing that the user should place 3% of their budget on 'nestle', 2% of their budget on 'volkswagen', 1% of their budget on 'facebook', 2% on 'shell' and 92% on 'starbucks'. This portfolios expected return is 4.65 % and the risk is 5.71 %. 57

6.1 These figures are the monthly stock price for each respective company in US dollars, the prices were taken at closing from the 1st of each month(1 Aug 2018 - 1 Aug 2019). 58
6.2 Single-Point Crossover with a rate of 0.1% 60
6.3 The output from the program showing the companies involved, the fitness of the portfolio shown, the portfolio itself, and the risk and return percentages of said portfolio. The results show that this particular portfolio created, has a return of 4.21% and a risk of 5.82%, suggesting that the user should place 82% of their budget on shares in Starbucks. 60
6.4 Two-Point Crossover with a rate of 0.1% 61
6.5 Whole-Arithmetic Crossover with a rate of 0.1% 62
6.6 This figure shows 4 manually created portfolios, with various different weightings for each company (company names at the top). This figure also shows the expected returns and expected risks of each of these portfolios. In yellow are the percentage weights for each company, making up a five asset portfolio. The returns of the portfolio are using the calculations as seen in 5.5.7. Portfolio 4 has the best return percentage however it has a higher risk percentage than Portfolio 3, using traditional methods, a stockbroker would display both of these portfolios to a client and let them choose between a riskier portfolio or a safer one. Using Harry Markowitz' Efficient Frontier ([30]) a diagram can be displayed to determine which portfolios should be disregarded. 6.7. 64
6.7 This diagram shows the efficient frontier for this set of portfolios, with the return percentage on the vertical axis and the risk percentage on the horizontal axis. The diagram shows that Portfolio 3 (2.36%) and Portfolio 2 (1.52%) should be disregarded, as for the same risk a portfolio exists with a greater return. The two portfolios that would be viable for an investor are Portfolio 1 and Portfolio 4. 65
6.8 Comparing this portfolio with the manual portfolios from figure 6.6, it is evident that GenStock has created a portfolio with greater returns and a lower risk than all of the manual portfolios. A diagram shows this alongside the efficient frontier for this set of portfolios 6.9. 65
6.9 GenStock portfolio (4.79%) has a higher return and a lower risk than the fittest portfolio (Portfolio 4 (4.59%)) that was manually created. 65

6.10 This figure shows GenStock generated portfolio weights (yellow), a return percentage for the portfolio as a whole (light green) and a risk (orange). On the left of the figure, there are the corresponding crossover and mutation rates that were used to generate each respective portfolio. The fittest portfolio was created when the crossover rate was at 0.1%. The return percentage of this portfolio was 4.08% and the risk percentage was 6.47% (dark green), this portfolio had a fitness of 0.06159. 66

6.11 This figure shows GenStock generated portfolio weights (yellow), a return percentage for the portfolio as a whole (light green) and a risk (orange). On the left of the figure, there are the corresponding crossover and mutation rates that were used to generate each respective portfolio. The fittest portfolio was created when the mutation rate was at 100%, this portfolios fitness was 0.07502 and had a return of 4.65% and a risk of 5.77%. . . . 67

Acknowledgements

I would like to thank my family first of all, not only for their constant support throughout the making of this thesis but also for their love throughout my entire university career.

I would also like to thank my supervisor Dr. William. J Teahan for his continued support and guidance throughout the creation of this project.

Abstract

This dissertation describes the design and implementation of a program that combines Web Scraping, Information Extraction, Sentiment Analysis and Genetic Algorithms to create a program that finds the most optimum stock portfolio. The aforementioned program successfully automates the retrieval of news articles, searches for company names, parses online historical data on said companies, and generates a portfolio using a genetic algorithm.

Experiment results show that a portfolio can be created in just over 3 minutes 30 seconds, this is 2.1 thousand % faster than manual portfolio selection using the News Article web scraping tool, based on the research from sources ([15], [12], [14]), and 2.3 thousand % faster without using the News Article web scraping tool (6.2.1).

This program coined 'GenStock' has been evaluated with results showing that GenStock is not only faster than manual portfolio selection, it requires no previous stock knowledge for a portfolio to be created, creating the most efficient possible portfolio tested against the Mean-Variance model and Harry Markowitz' Efficient Frontier model [30].

GenStock produces a series of weights that account for percentages of the investors budget to be invested in shares of those respective companies based on the positioning of the weights (e.g index *i* of the portfolio = the percentage of the budget that should be invested into company *i*).

Introduction

3.1 Background and Motivation

The need for instant automatic stock data has become ever more prevalent, it is thought that over half of all trading of the S&P 500 Index is done by automatic trading programs [2]. Automatic Stock Trading otherwise known as Algorithmic Trading is a set of algorithms that use predetermined trading instructions to make the best decision on whether to buy, sell or keep stocks for investors, based on companies' public data and worldwide business news.

Algorithmic Trading has changed the way every stock market in the world makes its money. These computer programs can return millions of pounds to investors in milliseconds due to the speed they transmit and analyse companies' data, without the need for a financial advisor or stock broker. This has become so time-crucial that Michael Lewis [39] claimed that some traders would even "sell their grandmothers for a microsecond". It is because of this need for instant data that a new Fibre-Optic internet connection was created between a data centre in Chicago and New Jersey. The line created was said to be "the most insistently straight path ever dug into the earth."[14], At 825 miles and 13.3 milliseconds, this circuit shaves 100 miles and 3 milliseconds off of the previous route of lowest latency.

The dominance of Algorithmic trading even caused a market plunge in 2018, with a horde of programs all searching for a trigger point when deciding whether or not to buy or sell stock. This means that when a computer views a stock and that stock drops below a certain percentage, say 5% returns the program will decide to sell. However, having so many of these machines all selling at 5%, a free-fall occurs and sends the market into a downwards spiral. Piers Curran [19], head of trading at Amplify Trading discussed this as well as stating that a definite correlation exists between Algorithmic Trading and the volatility of stock markets.

The ever-growing volatility of stock markets has seen an increase in the number of investors who invest in stock portfolios as opposed to single stock purchases. A stock portfolio is a way to collate a myriad of assets together in order to reduce the risk and maximise the returns of the portfolio as a whole, as opposed to the individual assets, encouraging the idea that owning different kinds of financial assets is less risky than owning only one asset.

The selection process of creating a stock portfolio often involves the analysis of historical data on selective companies to predict their risk and returns. However, there are a number of factors an investor would have to weigh up when creating a stock portfolio such as: •What companies should be selected to place stocks on? •What is the most efficient way of dividing their budget over the portfolio? •And lastly, how long will it take to do all of this research and then place the stocks?

The job of answering these questions is typically done by a stockbroker. A stockbroker is someone who buys and sells stocks on behalf of their clients, as well as updating clients on new investment opportunities and staying up to date on the business news from around the world to offer a portfolio of stocks to their client. However, for a five asset portfolio there are different combinations. For example, for a five asset portfolio, there the total number of possible portfolio combinations is 3,763,608. This is simply too many combinations for a single stockbroker to select from, ultimately making the expectation that a stockbroker could find the perfect portfolio, impossible.

3.2 Aim and Objectives

The aim of this project is to create an optimum portfolio of stocks using genetic algorithms and web scraping. The program should give the user five split weights for specific companies, the investor can then split up their budget accordingly. The system will also alert the investor each time it runs if a company has a very bad review or article, an example being "Company 1 has had a very bad article published regarding them.".

The steps taken to complete this project are as follows:

- Design and implement a web scraping tool to parse through business news articles.
- Design and implement a Text preprocessing algorithm that; removes capitalisation, removes stopwords and removes punctuation from parsed articles using the 'Stopword' list from Python's NLTK.
- Design and implement a Named Entity Recognition tool to search the processed articles for company names with help from the research of Lample et al's paper 'Neural Architectures for Named Entity Recognition' [38].
- Design and implement a Sentiment Analysis tool that reads through the processed articles to determine if an article was positive or negative based on Nakov et al's work in their seminal papers SemEval [41].
- Design and implement a Web Scraping tool to search for historical data using a browser simulation called Selenium [52].
- Design and implement a Genetic Algorithm system based largely on the work by Yahaya [10] and Sefiane et al [53] to find an optimum portfolio given a company list.
- Evaluate the speed against manual portfolio selection and the accuracy of this program against a well known model for testing portfolios, called the Efficient Frontier created by Harry Markowitz [30].
- Evaluate the fitness of selected portfolios based on manual experiments.

Objectives:

- Produce a literature review on Web Scraping, Genetic Algorithms, Sentiment Analysis, and how stock portfolios were historically and currently created.
- Design a program ‘GenStock’ to scrape business news articles from the internet with which company names will be extracted, if a company name is found the program will use a sentiment analysis tool to judge whether the article is positive or negative and using this information combined with the historical stock price data on said companies a stock portfolio will be created using a genetic algorithm. The best portfolio will be a portfolio that has the lowest possible risk for the highest possible return.
- Implement ‘GenStock’ using Java alongside the Java libraries ‘jsoup’ and Selenium WebDriver for web scraping.
- Evaluate ‘GenStock’ for its efficiency, measuring the quality of the portfolio, as well as the time it takes to create the portfolio.

3.3 Summary of Dissertation

This dissertation will first show the research made to understand the problem as well as aid in finding a solution, followed by this the design chapter discusses the design objectives set to get the most out of this program including: how fast it should run, the ability to choose the users input, the need for an object oriented approach, as well as the discussion to the programs user interface.

The implementation chapter comes after this which talks about the various different technologies used to complete this project in its entirety as well as the techniques that are not so favourable and should not be pursued in any future work of this project. The implementation chapter also provides pseudo-code as well as code snippets showing exactly how certain parts of the project were developed. The evaluation of this program is the next chapter, the program was tested in a series of experiments that measure time, accuracy and precision against manual portfolio selection as well as measuring the accuracy of the portfolio against Harry Markowitz Mean-Variance model [30].

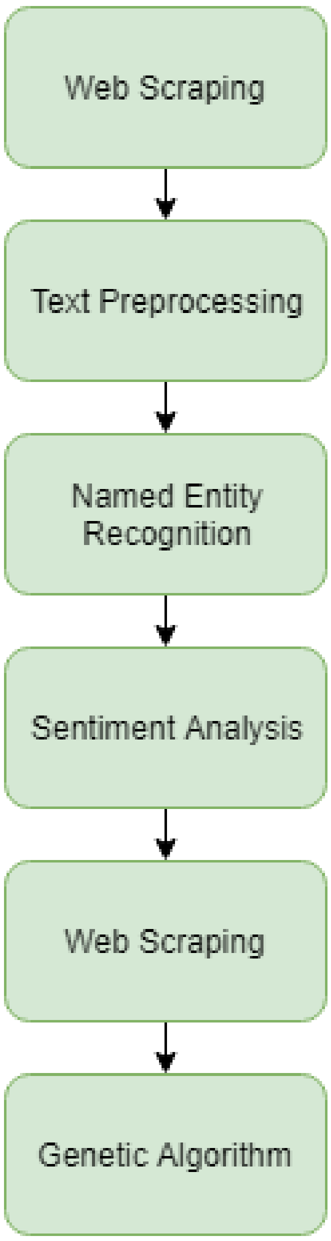


Figure 3.1: The workflow of the program to create an optimum portfolio selection.

Literature Review

This chapter reviews various publications that have benefited or contributed to this project. The articles in this chapter are all relevant to this project and include a mix of the following; Text Analysis, Data mining, Artificial Intelligence in general, Natural Language Processing and more specifically Evolutionary Algorithms.

This chapter is organised as follows. In the next section, an overview of what stock portfolios are is examined as well as how these portfolios are generally created, the pricing of each portfolio and lastly, how a portfolio is measured in terms of efficiency as well as an introduction to Algorithmic Trading and some methods currently being used in industry. Section 4.2 discusses how Stock data is harvested from the internet whilst section 4.3 explains how such information is processed to make it useful for stock selection. In section 4.4 some computational methods that have been used to create a stock portfolio are discussed and analysed as well as discussing what 'Fuzzy Portfolios' are and how they differ from regular stock portfolios. Section 4.5 concludes this chapter by summarising what has been found and how the information found is relevant for latter chapters.

4.1 Overview of Stock Portfolios

A stock portfolio is a collection of stocks used to reduce the risk and maximise the return for investors, by spreading their investment over multiple stocks. If created by a brokerage firm, they would generally cost an investor between £10 and £15 per month or per investment ([44]). A stock broker would talk to the investor and determine what sort of risk they would be willing to take per investment, and over a certain amount of time they will comprise and present multiple portfolios within a previously discussed budget to the investor. Typically they will present a riskier portfolio with a greater return and a safer portfolio with a lower return.

However, an investor may decide that they do not want to pay the brokerage fees and create their own stock portfolio. The downfall of this is that it typically takes a lot of time and can intrude over their personal lives. Ted Werschler, the current investment manager at Berkshire HathawayTime, stated in 2012 that he would not place an investment unless he spent 500 hours researching the idea [14]. For many investors, the time to research a single company can vary between 15 minutes for a smaller investment and a number of years for a larger investment [15], [12]. This means that if a single stock takes 15 minutes minimum to research, it takes a minimum of 75 minutes to research five stocks. On top of that, an investor will have to add the time it would take them to decide how to split their portfolio up.

The decision of how to split up a portfolio is not only time consuming, but can be very confusing to a novice investor. The most common way to determine how profitable a portfolio is, is by using the Efficient Frontier model created by Harry Markowitz. In his book the ‘Efficient Frontier’ [30], Markowitz focused on a repeated phrase, "The investor does consider expected return a desirable thing and variance of return an undesirable thing". Markowitz assumed that investors would want the maximum return for the lowest risk for a given portfolio. This thought was then expanded to a model called the Efficient Frontier, in which Markowitz said that for every portfolios’ expected return, an investor will only be interested in the portfolios with which the risk is lowest, therefore any portfolio with the same return but with a higher risk should be disregarded.

Markowitz also talked about how the investor should split up their budget amongst the portfolio in the most efficient way, stating that it would be up to the investor to decide which portfolio would be preferable to them. For example, one investor may want to gamble on the high risk high return investments, whereas another investor may play it safe and go for the low risk low reward investments.

The time and complexity needed to make these calculations are not preferable for casual investors or traders who have limited time, and as previously mentioned (3.1), all traders should be thinking ‘the quicker the better’.

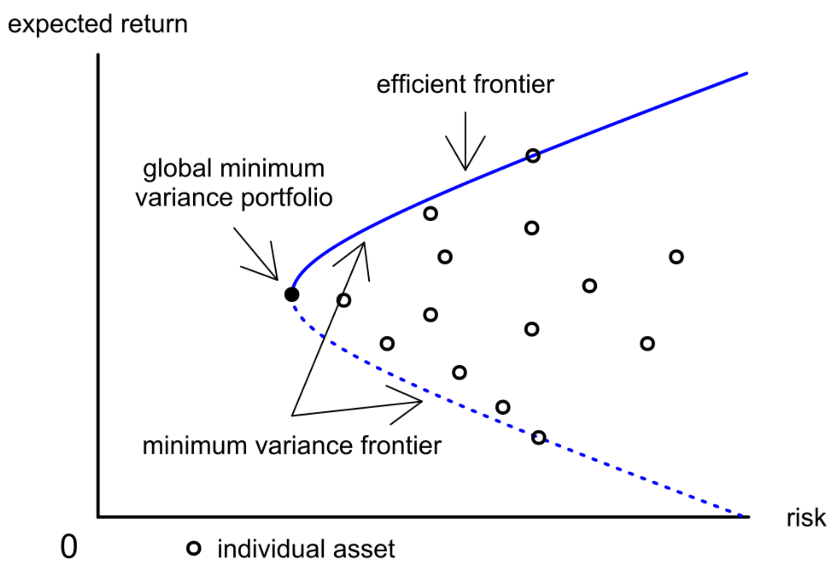


Figure 4.1: The Efficient Frontier designed by Harry Markowitz. The Efficient Frontier is a way to measure risk and reward of a portfolio while removing less profitable portfolios. The dots that lie on the solid blue line show the most efficient portfolios, one giving a slightly less return but for a dramatically decreased chance of risk, whilst the other portfolio offers the opposite. All further portfolios that lie below the solid blue line should be disregarded [40].

4.1.1 Automatic/Algorithmic stock trading

In order to speed up the selection for worthy investments, algorithmic trading was developed as an opportunity to remove unnecessary time spent wading through a companies data. These programs work by analysing a company's data and determining either when to invest, when to withdraw any stocks from a company or how to combine investments for the greatest possible return. The type of data that is analysed can range from: the company's stock price at the end of each month, environmental/geographical concerns, the company's stock price volatility, and even the company's CEO's past erratic actions.

Algorithmic trading started to become prevalent in the early 2000's after IBM published their paper "*Agent-Human Interactions in the Continuous Double Auction*" [20], which showed that computers can outperform non-expert human subjects by a clear margin in terms of producing larger gains. This paper caused a ripple effect throughout the trading world, resulting in more investors using standalone programs to set their own investments as opposed to a brokerage. From then on, faster networks were created around trading hotspots, meaning algorithmic trading programs could execute trades faster than human traders [27]. Since then, a vast amount of money has been invested to speed up the connection for the transfer of a company's data and traders. For example, the optical-fibre network between Chicago and New York, used only to shave 3 milliseconds off the transmission of data, cost around $300 million dollars to build [54].

Unfortunately, the downside to a lot of algorithmic trading programs is that they only factor in historical data, which does not necessarily reveal the true current situation of a company. News articles that inform the public on background factors of a company can drastically change the stock price of a company or can change the stock price of a company slowly. An example of a drastic stock price change is the release of a new product, this would cause a stock price increase due to the fact that the company would be making extra sales on those new products. An example of a factor that would change a stock price slowly is an increase of energy prices, if a company's only difference in the way they do business is an increase in how much the pay for the energy needed to make their product, they will end up paying more to create their product and making the same amount in sales as they did before the energy price surge [3].

4.2 Harvesting stock data

There are many different ways to harvest online data, from downloading RSS feeds instantly, parsing through online data manually, or developing other tools to read directly from the web. As previously mentioned, a company's stock price data is not good enough to make accurate stock predictions. Therefore, many algorithmic trading programs take into account business news. In fact, some websites have been developed specifically to make it easier for computers to read through the website's news articles. Lexicon, developed by Dow Jones does just that sending real time financial news to investors in a machine readable format, by helping machines "measure the sentiment, frequency and other relevant analytics to determine the 'tone' of the news" [1].

One method for harvesting online data is by using 'Web/Host Crawlers'. A web crawler is a program that systematically fetches web pages. They work by fetching a seed page, then searching for any links to further sites, and they then search those pages and so on. The World Wide Web Wanderer is what is described as 'The First Web Scraper'. It was created to measure the size of the World Wide Web in June 1993. It was developed by The Massachusetts institute of Technology, by Matthew Gray [29]. The crawler was used to generate an index of every webpage that existed. This index would be collated and returned to show the exact amount of webpages that existed at the time.

The Host crawler is similar to a web crawler. However, they stay within a host website and scrape data from the host site only. For stock data, a host crawler could fetch a web page that shows a company's data, scan that page looking for certain HTML tags, and outputting only information that the investor deems necessary.

4.3 Processing web data

Once data has been harvested from a website onto a machine, a developer needs to understand what they want to do with the data, what type of data it is, and the size of the data. For an automatic trading program that reads historical data only, the developer will only need to store numbers. However, if the developer wants to read news articles about companies, they will need to store text data as well. From there the developer will have to create some mechanism to show the correlation between the words printed in the news articles and the change of the company's stock well being.

Sentiment Analysis is one way to assign numerical value to words. Nakov et al [41], defines Sentiment Analysis (SA) in his series of seminal papers 'SemEval', as "the task of detecting whether a textual item (product review, blog post or editorial, etc.) expresses POSITIVE or NEGATIVE opinions." Nakov et al. also talks about how Sentiment analysis models can be used to classify very informal forms of communications such as tweets on Twitter, and explains how over the years Sentiment Analysis has been used by the political sciences, social sciences, market research and many others, to determine human behaviour and feelings on a subject.

With regards to Nakov et al. work on 'SemEval', his model created in 'SemEval-2016 Task 4' focused on the social media platform 'Twitter' [42] and was used to gauge whether users of the site react favourably or negatively to certain products, people, political party or policy. Nakov et al. also discusses the fact that during development they felt they had to replace the classic two-point ‘POSITIVE or NEGATIVE’ or three-point scale ‘POSITIVE’, ‘NEUTRAL’ or ‘NEGATIVE’, with a five-point scale ‘HIGHLY POSITIVE, POSITIVE, NEUTRAL, NEGATIVE, HIGHLY NEGATIVE’ model. The reasoning behind this is that five-point scales have become ‘ubiquitous in the corporate world where human ratings are involved’. An example in the stock market could be that the word ‘profit’ gets a positive response, whilst the word ’bankrupt’ gets a highly negative review.

On the other hand, some words in the English language have no value behind them, words like ‘as’, ‘a’, ‘the’ and ‘it’, have no value as they cannot be construed in a positive or a negative way. These words are known as ‘Stopwords’ and the removal of such words play a big part in Information Extraction (IE). IE is a branch of Natural Language Processing (NLP) which concerns the processing of natural language (speech, written text). A natural language is a language spoken by people (English, Chinese, Japanese etc) as opposed to artificial languages such as Java, C++ and Python. NLP is needed as computers cannot read text the way humans can, they simply see a string of characters. Therefore, using NLP we can follow rules onto the computer to perform routines given some text.

One routine as previously mentioned is the removal of stopwords, this involves scanning through a piece of text, reading every single word and comparing them against a list of stopwords. If the word is found in the list, the word can be removed from the text, and so on. Not only does this speed up the Sentiment Analysis part of the program by reducing the maximum amount of words it has to read through, it also reduces the size of the text, decreasing the amount of storage needed to store the text. Information Extraction techniques can further decrease the storage size needed for storing text by removing other unnecessary characters from the text, such as punctuation. Another technique is called Stemming, which reduces words to their root form ("information" -> "inform") thus further reducing the number of characters being processed.

However, there are techniques used in Information Extraction that do not help reduce the storage size, but makes the text easier to process. Case folding is a technique that replaces every capital letter with a lower case letter. This may seem trivial but when a program is looking for specific words, the algorithm used is case sensitive and will not return a positive result if the casing of the letters are not the same. Reducing all characters to lowercase allows for the program to search only for lowercase characters and removing a potentially harmful error.

Referring back to general NLP techniques, one particular technique is extremely useful for singling out specific words of phrases. This technique is called Named Entity Recognition. NER was first discussed in the MUC-6 conference in 1995 [43]. Whilst investigating other techniques such as Information Extraction and Coreference, the conference was most famous for its discussion on named entity recognition. In the conference, they described a 'Named Entity task' as one that consists of various subtasks, involving the recognition of entity names, temporal expressions and number expressions. The conference discussed how each of these expressions qualify as a "unique identifier of entities". Some things that would class as a unique identifier include: peoples names, company names, dates, times and locations. Named Entity Recognition has been defined by the European Bioinformatics Institute [24] as being "a sub task of information extraction that seeks to locate and classify named entity mentions in unstructured text into predefined categories such as the person names, organisations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc.".

The NER task involved automatically recognizing these different types of expressions based on their SGML tags. These SGML tags would have a format similar to this: "<ELEMENT-NAME ATTR-NAME="ATTR-VALUE" ...>text-string</ELEMENT-NAME>". It was the program's job to decipher what type of expression it was. The aim was to produce a "single, unambiguous output for any relevant string in the text".

During the evaluation of this program, some potential problems arose. The main problem being something called 'Elision', where more than one expression was used in a text string, an example given in the conference was the text string "North and South America". The program would mark 'North' as a location, and 'South America' as a location and mark them both up as being one location, whereas they should be marked up separately. This could result in the program naming them as the same expression when in reality they should be marked up separately. Another problem not discussed in the conference is if an individual or company had a particularly abnormal name such as a geographical location: "Paris", "Vienna", "India". The computer would class them as geographical locations instead of classing them as people, which as a result could skew results.

Since the MUC-6 conference, much more research has been put into developing more intelligent NER machines. One highly cited papers is Lample et al. ‘Neural Architectures for Named Entity Recognition’ [38]. Lample et al. discussed a fundamental problem with Named Entity Recognition (NER) models, which is that they have a dependence on domain-specific and language-specific knowledge Often times, they use a very small training corpus. A solution to this problem is the creation of carefully constructed orthographic features and language-specific knowledge resources. These however are costly to develop.

Lample et al. stated that they had obtained state of the art NER in four languages without resorting to any language specific knowledge or resources. Instead, Lample et al. used what they called a Bidirectional LSTM. This technique is typically a way to recognise entities using information from the past and future of a given word or phrase. A traditional approach of a Bidirectional LSTM would be: past: "I went to...", future: "... and then I played football.". In this example, the word could be ‘park’, and using a neural architecture, the computer is able to draw a conclusion that at the park people play football.

The problem that Lample et al. addresses will not directly affect this project, as the NER in this project searches for company names which are not domain-specific. However, it is also worth noting that the Sentiment Analysis used in this project is for the English language only. This is due to the difficulty of developing further language specific models.

Chiu et al. [17] also made use of Bidirectional LSTM’s for NER combined with a Convolutional Neural Network (CNN). A CNN is a neural network that is multi-layered. CNNs were developed to recognize visual patterns from pixel images without the need for extensive preprocessing, which takes time. Chiu et al. wanted to create a program that could read from an image, determine which words were written, and use a Bidirectional LSTM to guide the computer into determining what the next word will be.

Their project was aimed at reducing the amount of feature engineering. In other words, they wanted to make a program that is less reliable on the domain knowledge of the data, whilst retaining or bettering past results. They succeeded in their work using a Bidirectional LSTM conjoined with CNN architecture to read images and surpassed systems that employed heavy feature engineering, proprietary lexicons and rich entity linking information.

4.4 Portfolio selection using computational methods

Many researchers have attempted to find the most viable solution to creating a stock portfolio using various methods and techniques. Fernández et al. [25], for example, used a heuristic approach and a specific neural network called the Hopfield network to factor in cardinality and bounding constraints in order to trace out the efficient frontier associated to the portfolio selection problem.

Ayodele et al. [13] used particle swarm optimisation on a large search space of possible portfolio solutions to pick out the most efficient portfolio, Ayodele at al. also factored in a further constraint involving an ‘Expert opinion’. This ‘Expert opinion’ featured the knowledge of a financial analyst who would provide an in depth analysis of a company, providing resources such as: management calibre, dividend rate, book value and so on, these resources would typically be unobtainable for the general investor. Whilst this is clearly extremely useful, and would be very useful for a large company with the funds to expend on obtaining an experts opinion, the need for such an opinion is a major downfall for this solution, not only due to the cost but also the time it would take for the expert to factor in their opinion may very well take too long.

Another widely researched computational method used in the selection of stock portfolios, is called Genetic Algorithms. The history of Genetic Algorithms can be traced back to 1950, where Alan Turing wrote the seminal paper ‘Computing machinery and Intelligence’ [11]. In this paper, he discusses the potential need for natural selection to be the judgement of a program. Turing discussed how random changes in the program could be necessary to ‘stir the pot’ and stop trial and error programs becoming stagnant. Turing compared this necessity to mutations in natural selection. Turing also discussed a selection based mechanism to breed child programs based off their parents’ traits, so that after a certain amount of generations, only the fittest individuals would survive whilst the weak individuals would be weeded out of the genetic pool.

After this paper was released, many computer scientists from all over the world tried to use various evolutionary algorithms to solve real world problems (Barricelli (1954) [8], Bremmermann (1960-) [6], Holland (1970-) [7]).

In their book 'Artificial Intelligence: A Modern Approach' [51], Stuart Russell and Peter Norvig said that, Genetic Algorithms are a variant of a stochastic beam search, where parent states reproduce together to create a successor state. They stated that the difference between a stochastic beam search and Genetic Algorithms is that Genetic Algorithms uses sexual production as opposed to asexual reproduction.

Furthermore, John Holland [34] explained that such computer programs that are able to 'evolve' much like evolution in nature can solve complex problems that their creators may not fully understand. He stated the a genetic algorithm involves using a string of bits, where each bitstring is one chromosome, and each bit is one gene. Each bit usually corresponds to some value or operation. Given a target or goal for the search, if a chromosome has a higher fitness (closer to the target), it will survive and reproduce with other chromosomes that have a high fitness also, passing on their genetic information to their children. This is what is referred to as the 'Selection' operation. However, if a chromosome has a low fitness it will get weeded out of the gene pool. Working precisely how natural selection works in nature. For example, if an antelope is born with a genetic deficiency that makes it slower at running, a lion will be more likely to catch it and eat it as it would be a lot easier for the lion to catch. This results in the antelope not reaching sexual maturity thus not being able to breed and pass on its genetic information. Therefore, the genetic deficiency also does not get passed on.

David E. Goldberg [21], defined Genetic Algorithms as 'search algorithms based on the mechanics of natural selection and natural genetics, ... combining with the idea of 'the survival of the fittest'.', explaining that the more robust an artificial system is, the less it would cost to redesign such systems'.

Genetic Algorithms has also been used by researchers to create stock portfolios due to its ability to create solutions to problems that have no distinct yes or no answer.

Abubakar Yahaya [10] created an algorithm that combined Harry Markowitz' efficient frontier model with genetic algorithms. His paper describes the creation of an optimum portfolio of products based on a real dataset from a popular stock market. His paper also details the step by step mathematical instructions to create this algorithm, namely the risk and return equations of a given portfolio. His model works by using each portfolio weight in an individual as their 'genes'. These weights were mutated and reproduced over child generations. A generation of a set amount of portfolios with random weights is initially created. The fitness of each portfolio is assessed and those portfolios that have the best fitness will then breed to finally return the 'best' stock portfolio. The fitness function used was based on Harry Markowitz Mean-Variance model to determine the portfolios return % and risk %. The Mean-Variance equation can be found in figure (4.2).

$$\textit{Minimize} \qquad \Sigma_p^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} \omega_i Q_{ij} \omega_j$$

Figure 4.2: The equation used by Yahaya to determine a portfolios fitness based on Harry Markowitz' Mean-Variance model. p is each portfolio, n is the number of assets in the portfolio, w_{i} is the proportion of available funds allocated to asset i, Q_{ij} is the real-valued covariance of asset i with asset j.

Yahaya also describes the traditional Conventional Mean-Variance Model (CMVM), specifically the need to normalise the weight of the entire portfolio to unity (1). Each asset in the portfolio transcribes to a percentage of an investor's budget that should be placed with an asset. Because of this, the sum of all percentages can only ever be 100% otherwise investors will either have a portfolio that exceeds their budget, or an incomplete portfolio.

Similar to Abubakar Yahaya's work, Sefiane et al. [53] wrote another algorithm for selecting a portfolio using Genetic Algorithms. However, Sefiane et al. made some interesting experimental comparisons with three different crossover methods. To clarify, crossover is the combination of a parents genetic information to create a new offspring. The traditional crossover method is called 'Single-Point Crossover'. Found in figure 4.3, this method involves picking a random point in two parents chromosomes, and any bits that are to the right of this chosen point are swapped in the offsprings chromosome.

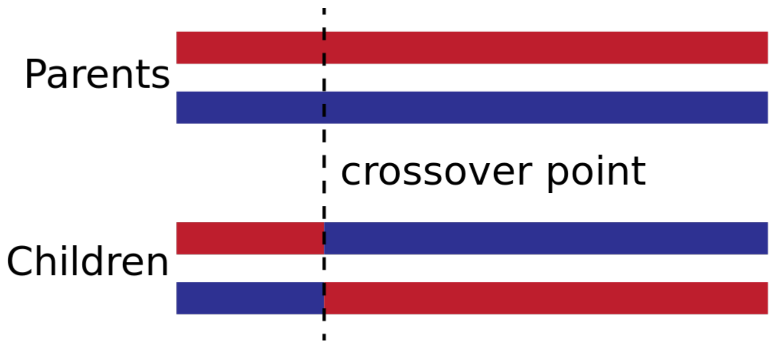


Figure 4.3: Single-Point Crossover [48]

Another method of performing crossover operations is called 'Two-Point Crossover', as seen in figure (4.4). Similar to Single-Point, a random crossover point is chosen. However, instead of swapping all the bits on the right of the crossover point, a second crossover point is created between the initial crossover point and the second to last bit in the chromosome. The crossover only affects the bits in between the two crossover points.

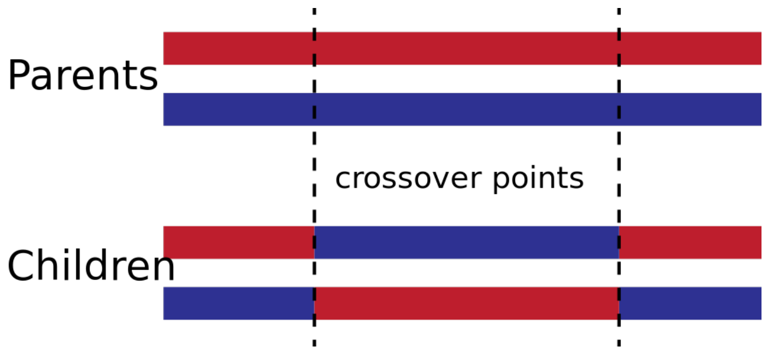


Figure 4.4: Two-Point Crossover [49]

Lastly, Sefiane at al. [53] performed a 'Whole Arithmetic Recombination' crossover procedure, see figure (4.5). This procedure takes the weighted average of two parents bits at any given point and places that average into the offspring's bit string.

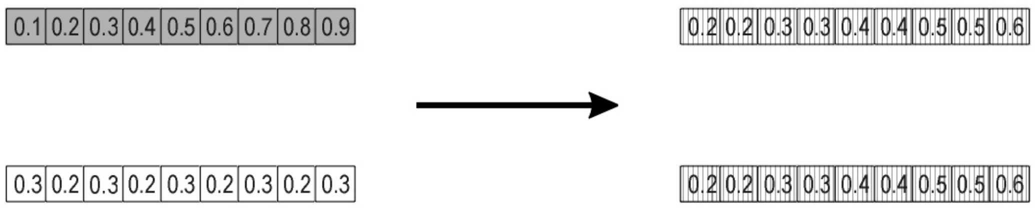


Figure 4.5: Whole Arithmetic Recombination [23]

Sefiane et al. concluded with reference to his test results, showing that the arithmetic cross over procedure gives better results than the two other procedures (i.e. single point and two points). As well as saying that With regards to the fitness function value, the arithmetic procedure should lead to the best choice of weights resulting in a portfolio being created that has the best possible return % and the best possible risk %.

4.4.1 Fuzzy portfolios

Fuzzy portfolios are stock portfolios using the practices of fuzzy set theory, which is an extension of classical set theory. The elements of a fuzzy set have varying degrees of membership. A comparison between classical and fuzzy set theory can be seen through the answering of a simple question: 'Do you like lemonade?' This would be represented using classical set theory as; 'Yes (1)' or 'No (0)'. And this would be represented in fuzzy set theory as; 'Very much so (1)', 'Yes somewhat (0.6)', 'Not particularly (0.4)', 'No not at all (0)'.

Calvo et al. [18], proposed a model that allows an analyst to offer investors not just the financially sound solutions but also some alternative solutions. This is built on Harry Markowitz' work on the Efficient Frontier.

However, Calvo et al. introduced an extra non-financial criterion. Calvo et al. discusses this criterion and states that the investor will be allowed to introduce in the model information about how far he or she is willing to go from the financially efficient portfolios knowing about the financial cost of these alternative solutions. This alternative solution means that the investor can choose to disregard the most 'efficient' portfolios, and go off their own whim if they choose to.

With regards to this, Calvo et al. therefore created an algorithm that uses the investor's decision on how far to diverge from the efficient frontiers as the 'fuzzy' set of data.

Calvo et al. [18] concluded that the applied fuzzy optimization technique leads to the best possibility according to a preference defined by means of fuzzy set theory, from the information that can be obtained from the investor.

From finding a deeper understanding of an investments risk [31], to creating a portfolio that takes into account the transaction costs of each asset [16], the use of fuzzy set theory to classify variables that are not inherently black or white has been used frequently in economics. In particular, Saborido et al. [50] described a method that combined Evolutionary Algorithms with a variation of the fuzzy portfolio selection model. Their approach involved the 'Mean-Downside Risk-Skewness' model, that sought to optimize it as a whole constrained three-objective optimization problem, optimising the expected return, the downside-risk and skewness of a given portfolio. They tested their hypothesis by using evolutionary algorithms, specifically the NSGAII, MOEA/D and the GWASF-GA algorithms which are each explained below. This method also used real data from the Spanish stock market as their input data.

The Nondominated Sorting Genetic Algorithm (NSGAII) algorithm was developed by Kalyanmoy Deb [37] and is a multi-objective optimization algorithm that aimed to solve the following three main problems with Multiobjective evolutionary algorithms: 1.) The O(MN^2) computational complexity where M is the number of objectives and N is the population size. 2.) Noneltisim approach to reproduction, in an Elitist program if the parents dominate the offspring, the offspring will be discarded and the parents will be added back into the population. Whereas, a Nonelitism approach means the offspring will always be placed into the population. 3.) The need for specifying a sharing parameter. A sharing parameter is used to check how similar an individual is to another, if the sharing parameter is very low then individuals with close to identical individual genetic information can reproduce together. Resulting in a low variety of individuals genetic information. Deb compared his NSGAII algorithm with more traditional MOEA's (multiobjective evolutionary algorithms) and found that his results of the constrained NSGAII on a number of test problems, including a five-objective seven-constraint nonlinear problem, compared with another constrained multiobjective optimizer, show that the NSGAII produces much better performance.

Multiobjective evolutionary algorithm using Decomposition (MOEA'D) was an algorithm created by Zhang et al [47]. Their work showed that if you decompose a multiobjective optimization problem into a number of scalar optimization sub-problems, in other words split a large problem up into smaller sub-problems and then make each sub-problem optimise purely by using its neighbouring sub-problems, it can have drastically lower computational complexity, faster than the NSGA-II algorithm.

The GWASF-GA (Global Weighting Achievement Scalarizing Function Genetic Algorithm) was developed by Saborido et al. [50], and was created to approximate the whole Pareto optimal Front, whilst using the switching of weights in their genetic algorithm as opposed to specific 1's or 0's that refer to some value. The GWASF-GA was found to have better performance regarding the hypervolume metric and epsilon indicator than the NSGA-II and the MOEA'D algorithms.

Saborido et al concluded that GWASF-GA with the new operators has the best performance, on average, with respect to the above performance metrics. From the financial point of view, the non-dominated solutions found provide a very diverse set of portfolios to the investors which optimize the three objectives at the same time. The approximations generated agree with previous results which assure that including skewness as an objective function in the portfolio selection problem greatly modifies the Pareto optimal front, providing a higher variety of efficient portfolios to the investors.

4.5 Summary and Discussion

In section 4.1 it was found that stock portfolio's are a collection of stocks used to replace single stock investments in order to reduce the risk of an investments and maximise an investments return by spreading their investment over multiple stocks. The downside to these stock portfolio's are that they typically are created by a Brokerage firm and cost money to create. On the other hand, an investor can create their stock portfolios themselves but this comes with the downfall of taking too much of the investors time to create.

A stock portfolio is evaluated in terms of risk and return and a popular model to measure the efficiency of a portfolio is called the Efficient Frontier created by Harry Markowitz, it works by aiming to reduce the risk and maximise the return of a set of portfolio's and returns the most efficient portfolio possible.

Section 4.1 also discusses algorithmic trading and how they typically scrape a companies data from the internet to use in calculations to determine the best possible investment strategies.

Section 4.2 discusses various ways in which data can be scraped from the internet and used by a developer. One such strategy is called Host Crawling, which stays within a host web site and reads HTML tags for specific necessary information, then returns that information to be stored locally on a developers computer.

The processing of web information is discussed in Section 4.3, with Sentiment Analysis offering a way to assign numerical value to words, to determine the positivity and negativity of a word. Section 4.3 also discusses how to discard useless information from scraped data. Information Extraction has been used to remove words that inherently have no value (stopwords), as well as offering techniques to remove punctuation and capitalisation of words in order to reduce the number of characters to be processed. Some words however that have no value such as company names, are very important and need to be plucked from the data, with Named Entity Recognition being one technique to pluck such words from a large body of text.

Section 4.4 mentions how certain computational methods can be created to replicate the production of a stock portfolio. One such computational methods discussed 'Genetic Algorithms' creates a bitstring and uses natural selection to reproduce bitstrings, creating offspring which contains the genetic information of its parents. Genetic Algorithms have been used in portfolio selection as they work through trial and error whilst removing the weakest portfolios from the genetic pool using a selection algorithm, thus leaving only the 'fittest' individuals remaining in the genetic pool.

Design and Implementation

This chapter discusses the design and implementation of this project individually. The first section discusses the overall design of the project, including an introduction to the Java classes used. The programs design objectives is talked about in the second section followed by the various technologies used and their importance to 'GenStock'. The fourth section discusses the classes used in more detail and how each class interacts with the additional libraries used.

The fifth section onwards discusses the implementation of this design including code snippets and the various mathematical techniques to produce an optimum stock portfolio, the implementation was created in the following order: Develop the web scraping / host crawling prototype to read business news articles. Develop some Text Preprocessing technique to remove stopwords, punctuation and capitalisation from news articles. These parsed articles are passed on to a Named Entity Recognition program that checks to see if a company is mentioned in the article. If a company is found a method to determine whether the article is positive or negative will be ran, searching and comparing each word in the article to the popular Sentiment Analysis model 'AFINN-111', an overall score of positivity is attributed to each article, the top five companies with the highest score will be placed into an investors portfolio. To determine how much of their budget should be spent on each company a genetic algorithm program is then ran and using crossover, mutation and selection operators a single optimum stock portfolio will be returned.

5.1 Overall Design

If the user decides to parse news articles, the first input to the program will be the downloaded news articles from a local .txt file. This .txt file may be as long as the user wishes, however it must contain at least five unique company names, else a five asset portfolio is impossible.

A further input to this program is a companies historical data. To search for a companies historical data, Host Crawling will be used every run of the program. This is to reduce the amount of storage necessary per data, and make sure no redundant data is stored. Web Scraping shall be used to search for business news articles and should combine with some NLP/IE techniques to remove capitalisation, stopwords and punctuation from every article.

Part of the program should search each word of every article and perform some sentiment analysis to determine if it was a positive or negative article, but before that Named Entity Recognition must be used to search for company names. The program should then rank the top five companies based on positive and negative reviews in the articles, these companies will go on to be featured in the investors portfolio.

A Genetic Algorithm class should be used to ultimately find the most efficient and profitable portfolio, using crossover, selection and mutation methods, as well as assessing the fitness of every portfolio that is produced.

The output of this program is a single stock portfolio accompanied by the company names and weights in the form of strings to the command line of the users computer, this will tell the investor what percentage of their budget should they spend on company i, j, k etc. A simple workflow diagram of how the program takes the necessary steps to create a portfolio can be found in figure (3.1).

A workflow of the design process showing how the different classes interact with eachother is shown in figure (5.1).

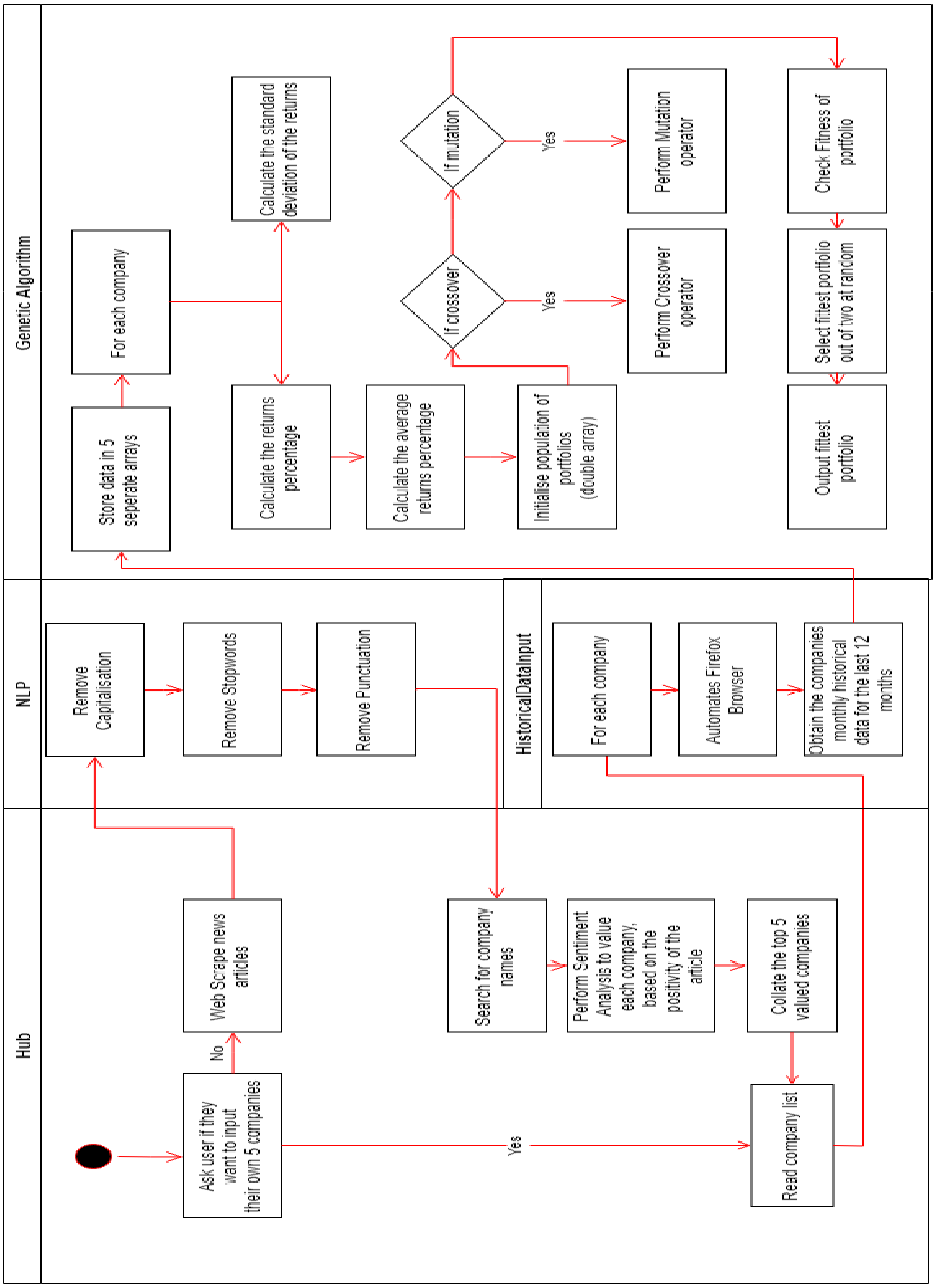

Hub
NLP
HistoricalDataInput
Genetic Algorithm
Ask user if they want to input their own 5 companies
No
Yes
Web Scrape news articles
Remove Capitalisation
Remove Stopwords
Remove Punctuation
Search for company names
Perform Sentiment Analysis to value each company, based on the positivity of the article
Collate the top 5 valued companies
Read company list
For each company
Automates Firefox Browser
Obtain the companies monthly historical data for the last 12 months
Store data in 5 seperate arrays
For each company
Calculate the returns percentage
Calculate the standard deviation of the returns
Calculate the average returns percentage
Initialise population of portfolios (double array)
If crossover
If mutation
Yes
Yes
Perform Crossover operator
Perform Mutation operator
Check Fitness of portfolio
Select fittest portfolio out of two at random
Output fittest portfolio

Figure 5.1: UML diagram describing how each class interacts with each other, eventually creating an optimum portfolio. The thin boxes containing the words: 'Hub', 'NLP', 'HistoricalDataInput', 'GeneticAlgorithm', depict the names of the classes responsible for each part of the program. The black dot represents the start of the program.

5.2 Design Objectives

GenStock is currently only developed for the developer's personal use, therefore has no need for a graphical user interface. However, there are some key design principles that must still be met for this project.

- GenStock should run from start to finish with one run of an executable file, to make it easier for an investor to run the program.

- GenStock should give users the option between inputting their own companies, and letting a web scraping tool choose positively trending companies, in order to add flexibility to the program.

- GenStock should use an object-oriented approach as it would be beneficial treating each portfolio as an object.

- GenStock should produce a single portfolio as output to a computer console. Only the fittest portfolio should be returned to reduce confusion and reduce the time that is needed for an investor to pick which portfolio is preferable.

- GenStock should accept an initial list of companies to search. This can be as broad a list as the investor wishes. This list is to help the Named Entity Recognition part of the program search for company names.

- GenStock should warn an investor of any potentially damaging news articles to any given stock. Whilst this is not 'needed', it would be useful to the investor and help them stay abreast of a companies current situation.

- GenStock should store any scraped articles in a .txt file for future training of the sentiment analysis program.

A big part of the design of GenStock is simplicity. An investor with no knowledge of the stock market should be able to run GenStock and a portfolio should be returned for them to follow and make money from. GenStock should however, offer options within the program such as the second (5.2) and fifth (5.2) design principle. Nevertheless the program should be streamlined, it should not take too long for a portfolio to be generated, and it should be accurate.

5.3 Technologies

5.3.1 Java

Java [33] is a widely used robust Object-Oriented programming language developed owned by Oracle. It was used for this project due to its vast amount of libraries and the ability to compile all code before runtime. This was extremely important for looking for errors in the development stage of this project. The two web scraping libraries used were ran through Java classes. These libraries are jsoup and Selenium.

5.3.2 jsoup

"jsoup" [36] is a Java library for working with real-world HTML pages, created by Jonathan Hedley [35]. It allows users the ability to; scrape HTML into a file or string, search for exact elements of the webpage, extract data using xpath or CSS selectors, make changes to the HTML page etc. It uses Java's object-oriented language to make objects of each element on the webpage. This allows users to select elements with more precision. Even elements that would be otherwise untouchable, such as "*Disappearing elements*".

5.3.3 Selenium

Selenium [52] is a tool to automate browsers. It is split up into two parts, the Selenium WebDriver and the Selenium IDE. This project focuses on the use of the Selenium WebDriver, specifically with the browser 'Mozilla Firefox'.

The Selenium WebDriver is a collection of language specific bindings to drive a browser, it "makes direct calls to the browser using each browser's native support for automation". How these direct calls are made, and the features they support depends on the browser you are using. Using Java's object-oriented approach, elements can also be selected using the WebDriver and stored as Java specific variables, therefore they can be saved and manipulated as the user may please.

5.4 Classes

In order to implement these design ideas, the following object-oriented classes were created. Figure (5.1) shows how each class interacts with each other class within the program.

- NewsScrape - contains a main method to read a business news website, read and store every article description and title into ‘NewsScrape.txt’. The main method will continue to run and scrape articles until the user enters a character on their keyboard.

- NLP - contains a method that accepts an integer, finds the corresponding article line number. Sends the article to a method to remove the capitalisation of the article, then sends it to another method to remove all stopwords in the article as well as the punctuation and returns the parsed article in the form of a String array.

- HistoricalDataInput - contains a method that accepts a String ‘companyName’, creates a mozila firefox driver using Selenium, uses browser automation to accept a business sites cookies. The class then uses the websites search bar to search for the inputted ‘companyName’, runs the search for the company, clicks the ‘Historical Data’ tab, changes the viewed data to be ’Monthly’ data as opposed to the default ‘Daily’. Finally, the method returns the companys closing stock price for each month, in the form of a Double array.

- Hub - contains a method that sends line numbers to ‘NLP’ and receives a String array (article) in return, and then searches in that array to see if a company name matches the ’companies’ preset list. If so, then it will check every other word in the article against the AFINN-111 Sentiment Analysis list, to determine whether the article was positive or negative and store the positivity of each company to a value, sending the five most positive companies to the GeneticAlgorithm class.

- GeneticAlgorithm - contains a method to fill five arrays with the historical data from ‘HistoricalDataInput’, finds the average returns of each company, performs risk/return calculations on each company, creates an array of random numbers (portfolio) between 0.01 and 0.96, the sum of which does not exceed unity (1). Create an initial population of portfolios, perform genetic operations (Selection, Crossover, Mutation) on portfolios at a chosen rate. Returns the fittest portfolio in the form of readable percentages.

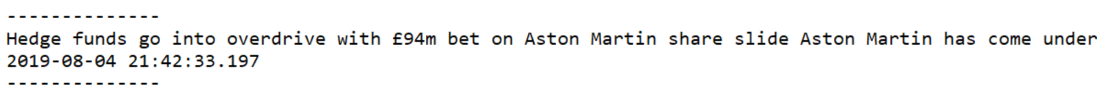
5.5 Implementation

5.5.1 Web scraping and Host Crawling for business news articles

To meet the second design principle this project could have simply used historical data to calculate the efficiency of the portfolios. On the other hand, historical data can ‘lie’. If a company has constant profits and looks like its doing great, a program using only historical data would pick it as the ‘for sure’ option. However, what if that company, despite profits, received some abnormal news? Such as an environmental disaster destroying their factory, or the CEO of the company deciding they had had enough and selling the company to another business? These would be examples of current direct facts that would change the company’s stock price immediately. Any program that uses only historical data would not factor this in and could very well tell you the best portfolio is with companies that are about to go out of business.

It should be made clear that this part of the project is optional. If an investor has the companies they want to make up their portfolio already chosen, they can simply input it into the program. Nevertheless, the need for current facts regarding a company has been surrounding the financial world globally and is often the job of a financial advisor or stock broker. It is their job to scan news sources and tell investors the good news and bad news of a company’s financial and economical situation, in addition to advising investors on who to invest in next. An important part of this project was picking the business news source. The optimum news source would be one that has a lot of news articles on a single page, one that is updated frequently and is ‘unbiased’.

This project uses a Web Scraping tool called 'Jsoup' (5.3.2) to scrape business news articles for data. Jsoup works by combining Java's Object-Oriented language to create objects for each element on the webpage. These objects can then be manipulated or stored by Java. The program first has to determine which part of the webpage is an article and which are other irrelevant elements. This was done manually through an inspection of the webpages elements [9]. The result was that all articles on the page had the "*<p>*" tag as opposed the "*<div>*" tags. Consequently, the program could search through every "*<p>*" tag and store its content as text into a .txt file, as well as a time stamp to tell the developer when an article has been stored into the file. The time stamp was added using Java's built in TimeStamp [32] library and allows for not only the date to be stored but also the time of day, an example of an article stored alongside a timestamp can be found in figure [5.2].



```
--------------
Hedge funds go into overdrive with £94m bet on Aston Martin share slide Aston Martin has come under
2019-08-04 21:42:33.197
--------------
```

Figure 5.2: This figure shows part of one scraped article saved into a .txt file with a TimeStamp.

For the purpose of this project, test data was generated. The data retrieved came from news articles that were acquired and stored every ten minutes for seven days, which resulted in a single .txt file with a size of 9.6 MB. This program could run once and if five or more companies were mentioned it would still work. If there is not a minimum of five companies mentioned the program would error out and report to the user that there was not enough companies in the articles to form a portfolio. The purpose of running the program for seven days was to test the Sentiment Analysis part of this project.

5.5.2 Text Preprocessing to remove stopwords, capitalisation and punctuation

Once the news scraping phase has completed, the text file is parsed by a Java class that uses certain Text Preprocessing methods on each article. The first method deals with the removal of stopwords. The removal of stopwords is vital for removing unnecessary natural language speech elements from sentences without disrupting the true meaning and sentiment of the sentence. Stopwords include words like: "as", "if", "and", "the" etc. None of these words would convey that an article is either positive or negative, therefore they can be removed. Doing so, aids in reducing the size of the articles that are going to be stored.

The removal of stopwords in this project was done by parsing through every line in the article and comparing every word with a word from a known ‘Stopword’ list. This list was created by the Python developers as part of their Natural Language Tool Kit (NLTK) package, this package contains 126 different English language words. One word was removed however, and that word is ’not’. The reason for this is discussed in the Sentiment Analysis part of this paper. The list of stopwords used in this project can be downloaded from GitHub [45]. For this project, the stopwords were stored in a .txt file and read into the class. If a stopword is found in the article it will remove it safely.

The next cause of action was to remove any unnecessary capitalisation of words. In the English language, company’s names are typically capitalised which makes it hard for computers to scan for them since computers are case sensitive, therefore a search for "company1" is entirely different to a search for "Company1". This is the reason why removing the capitalisation of words is vital. Java makes it very easy to remove the capitalisation of words of a String using the method "*toLowerCase()*". This removes the capitalisation of every word in every article. The next step was to remove the punctuation.

To remove the punctuation from a String in java, you first have to convert the String into each individual character (Char). Doing this allows you to check each character against a set of characters, in the case of Java a set of punctual characters would be "*[ˆa-zA-Z0-9]*". The program can then check each character against this set of legal characters. If there is a character that exists outside of this list then punctuation is present, the program then will replace it with "", in other words the system will replace it with nothing, thus removing the punctuation entirely. Using the text preprocessing part of this project greatly decreases the storage needed for each parsed article. This is simple to do in Java, the code below shows exactly how to achieve this result using Java’s built in ’replaceAll()’ method.

```
public static String removePunc(String full) {
    String noPunc = full.replaceAll("[^a-zA-Z0-9 ", "");
    return noPunc;
}
```

Lastly, the article can be jumbled up into what is known as a 'bag of words', This bag of words destroys the order of the text, this model gives the developer the ability to count the amount of unique words, this can then be analysed. An earlier development of this project used this function, but the current development of this project does not. However, there is no harm in keeping the bag of words model, therefore it was kept. The program then stores each article in a String array and sends each article through a Named Entity Recognition and Sentiment Analysis parser.

5.5.3 Named Entity Recognition on the extracted articles

The program has a preset stored forty-four companies chosen either from the FTSE-100 stock portfolio or hand picked. The FTSE-100 stock portfolio is heavily invested in because it "is an index composed of the 100 largest (by Market Capitalisation) companies listed on the London Stock Exchange (LSE). These are often referred to as 'blue chip' companies, and the index is seen as a good indication of the performance of major companies listed in the UK." [46]. The other companies that are preset are there due to the fact that articles might not review any of the FTSE-100 companies, therefore you would have a program that prints out a portfolio of nothing. The bigger the preset of stored companies is the better the program will run, as it can take in more information from articles. Nevertheless, a requirement for this list is that each company in the list must be compatible with the historical data collector web scraper used later on.

Each article that gets passed through to this method initially comes through as an array of strings. Each string representing one word. This method first checks each string against the preset company list. If a company name is found to be present in the article, it must mean that the article has mentioned that company in some context. This method of checking for a specific company/individual is called Named Entity Recognition. Now the program knows a company is featured, but does not know if the context of the article that has been written about the company is positive or negative. This is where the Sentiment Analysis section of the project is used.

5.5.4 Sentiment Analysis of the extracted articles using AFINN-111

Sentiment Analysis is a way of mining text to determine whether the text is inherently positive or negative. Sentiment Analysis uses training data containing words with a given value, determining the strength in positivity or negativity of each individual word. There are lots of training data for sentiment analysis available for developers, but this project used the AFINN-111 data. AFINN-111 was manually created by Finn Årup Nielsen [26], it contains 2477 words/phrases from the English language. Each word ranges between -5 to 5, with -5 being a very negative word and 5 being a particularly positive word.

The AFINN-111 word list was used in this project as the program will be reading news articles written by human beings, and therefore the articles will not only have straight to the point adjectives surrounding the state of a company ("bankrupt", "profit" etc), but also words that convey negative or positive feelings("poor", "terrible", "fantastic" etc), and it is for this reason that AFINN-111 is used, as it is very good at laying out the foundation for judging the negativity or positivity of a given word or phrase.

The program sends each word through a method that checks the word against the AFINN-111 word list. Furthermore, the program then checks to see if the word before the current word is a '*not*', this prevents situations such as '... this company is **NOT** great' and the program only finding '*great*'. The program then concatenates the value of each word in the article into a variable, which acts as the company's '*value*'. Once the article is fully parsed by the sentiment analysis method, the value is stored in a TreeMap along with the company name. A TreeMap is a data structure that gives an efficient way of sorting and storing key-value pairs. Similar to the way Dictionary's in Python work, key-value pairs are identifiers of any variable type that refer to each other. In the case of this project the Key was the value of the company and the Value was the company name, this allowed the value and the name of a company to be found using only one index.

Now the program has a TreeMap full of company names and their respective values, what next? The system needs to sort the TreeMap to find the five highest valued companies. These would be the five companies that will make up the investors portfolio.

5.5.5 Developing the Second Web Scraping/Host Crawling Prototype

It became clear early on in this project that there would be a need for a company's historical data. This could have been made technologically simple by manually entering the data into a .txt file. However, this would have been time consuming, very unmanageable with a large number of companies and more importantly it would have needed updating every month when there is new historical data created. Even the storage of the historical data would have become a problem, if you had stored it locally, you would run out of disk space very quickly. On the other hand, if an investor had stored it in a cloud architecture, they would have to pay.

Because of these reasons, this project uses Web Scraping to scrape for historical data every time the program is run. This allows for the program to get up-to-date data as well as removing the need for storing that data, thus saving disk space.

Selenium WebDriver was the library that was used. Selenium WebDriver is a browser automation which makes use of Java's Object-Oriented approach to allow elements of a HTML webpage to be stored into Java native variables. The WebDriver programmed imitates a 'Mozilla FireFox' browser and looks through a very popular website that has a plethora of statistics and data for each company one by one, the program navigates through the various tabs and buttons on the page to find the historical stock price at closing (closing refers to when a market closes for the day) for every month for the last year, and stores that data in a series of arrays.

However, the acquisition of a company's historical data may not be enough. The program must make risk / return calculations on each company and determine the expected risk/return of an investment into each company as well as the whole portfolio. Therefore, the second web scraping class 'HistoricalDataInput' sends 5 arrays back to the 'GeneticAlgorithm' class for interpreting, containing 12 months worth of a companies stock data.

5.5.6 Selecting the optimum portfolio using a Genetic Algorithm

All of the genetic algorithm methods and techniques that are mentioned in this section are created in the ‘GeneticAlgorithm’ class. However, one method is used to run an instance of the Web Scraping class as discussed in the the previous section of this paper (5.5.5). The method used to run this class is called ‘fillCompArrays()’, this method runs the Web Scraping class and stores 12 monthly values into 5 respective double arrays, these values are the monthly stock price of each company for the past year. The mean stock price of each company is also stored in a global variable called ’means[]’.

The calculation found below starts by collating the return percentage on the stock price for each month and each company, these results are also stored in respective arrays. This calculation works by taking a single months stock price, dividing it by the previous months stock price and subtracting 1. This will result in 11 values representing the percentage change in returns for each month of the respective companies history, the equation for the mean expected returns of a single company can be found below.

Mean expected returns of a company:

$$R_\iota = (\eta_\iota \,/\, \eta_{\iota-1}) - 1)$$

Constraints:

- R_ι is > 1

where:

- R_ι is the return percentage change each month
- η is a monthly result
- ι is a given index

After these means are collated an average of the means must be created per respective company. Given that there is 11 percentage return values, the average is calculated by the sum of all the percentage returns divided by 11. These means are stored in a single double[] array '*means[]*', pseudocode to accomplish this can be found below. Whereas, the detailed code snippet written in Java can be found on the next page.

```
create array called means of size 5
for every company
    create new array that called the MeansEach
    set previous to company 1
    for every company i starting at company 2
        add the mean of company i divided by the previous company – 1 to MeansEach
        if company i = company 2
            set sum to MeansEach[i]
        else
            append sum by MeansEach[i]
        set previous to the current company
    set mean to = sum divided by the length of MeansEach
    set means[i] to = mean
```

```
public static double means[] = new double[5];
for (int i = 0; i < companies.length; i++) {
    meansEachComp = new Double[11];
    previous = companies[i][0];
    Double sum = null;
    for (int j = 1; j < companies[i].length - 1; j++) {
        current = companies[i][j];
        meansEachComp[j-1] = (current / previous) - 1;
        if(j == 1) {
            sum = meansEachComp[j-1];
        } else {
            sum += meansEachComp[j-1];
        }
        previous = current;
    }
    mean = sum / meansEachComp.length;
    means[i] = mean;
}
```

The equation to determine the average of the mean expected returns of a single company can be found below.

Average of the mean expected returns of single company:

$$\bar{R} = \frac{\Sigma_{i=1}^{S} R_{\imath}}{S}$$

where:

- $\bar{R}$ is the average of the mean expected returns of single company
- S is the amount of expected return elements
- $\imath$ is a percentage difference between months

There are a couple of ways that the risk of a stock can be determined, namely: The Sharpe ratio [56], The Capital Asset Pricing Model [55] and The Standard Deviation [22] of the stock. The most common way of assessing risk of a single stock is to find the standard deviation.

The standard deviation of a stock [5] is calculated by using the average of the mean expected returns for each company. With this value the program can create a variable ‘*standardDev*’ and increment ‘*standardDev*’ by $(eachReturnPercentage - averageReturnPercentage)^2$ for each value in the array, ‘*standardDev*’ is then divided by the size of the ‘means[]’ array - 1. Finally, the squared root of ‘*standardDev*’ is found, each company’s risk is then stored in another array called ’*standardDevs[]*’. This calculation of a company’s risk is done by a method called ‘*initialStanDev*’, ‘*initialStanDev*’ uses the equation found at the top of the next page followed by the code used for this method.

Risk of a single company (Standard Deviation):

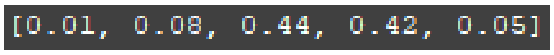
$$\sigma_c = \sqrt{(R_\iota - \bar{R})^2 + (\ldots - \ldots)^2 + (R_S - \bar{R})^2}$$

where:

- σ_c is the standard deviation of a company
- R_ι is the return percentage change each month
- $\bar{R}$ is the average of the mean expected returns of single company
- S is the amount of expected return elements

```
public static double initialStanDev (Double[] comp) {
      double sd = 0.0;
      double sum = 0.0;
      double avg = 0.0;
      for (int i = 0; i < comp.length; i++) {
            sum += comp[i];
      }
      avg = sum / comp.length;
      for (int i = 0; i < comp.length ; i++) {
            sd += Math.pow((comp[i] − avg), 2);
      }
      double standardDeviation = Math.sqrt(sd / (comp.length − 1));
      return standardDeviation ;
}
```

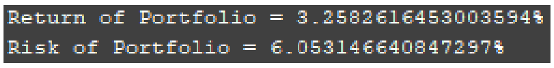
Given that the portfolio consists of a split amount of weights ranging from 1% to (100% - portfolio size). The portfolio size is not concrete, one could have a portfolio size of 25 if they chose. However, this would slow the program down drastically as it has to store and check all 25 weights. The portfolio size chosen for this program was **5**, the reason for this is it is the same as many of the papers that are covered in the literature review earlier on in this paper (Yahaya, 2009, ([10]), Sefiane, 2012, ([53])). The weights in the portfolio's will be random each time they are created. For this reason, a double[] method called 'individual()' was created, each time this method is called, a new portfolio with random weights of size (5) will be created. An example of this can be seen in figure (5.3).



```
[0.01, 0.08, 0.44, 0.42, 0.05]
```

Figure 5.3: An individual portfolio created by the method individual(). 0.01 represents that the user should place 1% of their total budget on company 1, 0.08 represents the investor should place 8% of their total investment budget on company 2 and so on.

With these weights, the program also works out the expected return and risk of the portfolio as a whole. The equation for calculating the risk of a portfolio is written in (5.2), and the equation for calculating the return of the portfolio is written in (5.5.7). An image of the return % and risk % of one particular run can be found in figure (5.4).



```
Return of Portfolio = 3.2582616453003594%
Risk of Portfolio = 6.053146640847297%
```

Figure 5.4: The return % and risk % of one particular run.

The risk return calculations is what will ultimately be the fitness target for the evolutionary algorithm.

The first point of call was to create a method that generated a random portfolio. The 'individual()' method did this, the weights of course needed to add up to unity. It was found that the best way to do this was to create 4 random weights that are all greater than 0.01 but the sum of them is less than 0.96. Then create a final weight that is equal to the difference between the sum of the other weights and 1.

The weights are all shuffled within the array, before finally being returned as a double array. The method 'individual()' is called however much the population size is set to. The code for the method 'individual()' is on the next page, whilst pseudo-code for this method is posted below.

It was found that the program had a positive linear correlation between the quality of initial portfolios and the initial population size, this is expected as the weights in the initial portfolios are random, therefore the more portfolios that are created, the larger the probability that a portfolio is created that already has optimum weights.

However, the population size has a direct trade off with the speed of the program, a population size of 10 was used and was found to be a good balance for speed and quality of portfolios alongside a generation count of 1,000,000. Another thing to note is that there is a maximum population size due to Java's JVM Heap Size, if you are on a 32x bit system, there will be a maximum of 1,000,000, and a maximum of around 1,500,000 - 2,000,000 for a 64x bit system. To reiterate for the testing of this project though, a initial population size of 10 was used.

```
create array of size 5 ' individual '
for the individuals length - 1
    Create random number 'gene' between 0 and 1
    if gene is greater than 0.01
        check to see if gene + the other genes is less than or equal to 1
        if so add gene to individual
        else set gene to another random number between 0 and 1
set the last gene in the individual to equal 1 - the sum of other genes
Randomly swap all genes in the individual
return individual
```

```
public  static  double[]  individual ()  throws  IOException {
    Random rn = new Random();
    double amount = 0.0,  double  max = 1.9,  double  left  = 0.4;
    double[]  individual  = new double[5];
    boolean  t  =  true;
    for  ( int  i  = 0;  i  <  individual . length  − 1;  i++) {
        t  =  true;
        while  ( t )  {
            double  genes  = Math.abs(rn . nextDouble()  \% max);
            if  (genes  >= 0.01)  {
                if  (amount + genes  +  left   <= 1.0)  {
                    individual [ i ]  = genes;
                    amount += individual [ i ];
                    max = max − amount;
                    left  =  left  − 0.01;
                    t  =  false ;
                }  else  {
                    genes  = Math.abs(rn . nextDouble()  \% max);
            } }   else  {
                genes  = Math.abs(rn . nextDouble()  \% max);
            }
        }
    }
    individual [4]  = 1.0  − amount;
    Random ran = new Random();
    for  ( int  i  = 0;  i  <  individual . length ;  i++) {
        int  randomPos = ran. nextInt ( individual . length ) ;
        double  temp =  individual [ i ];
        individual [ i ]  =  individual [randomPos];
        individual [randomPos] = temp;
    }
    return  individual ;
}
```

A method named ‘init()’ creates the initial population of portfolios, but before the portfolios are stored, they must be normalised. This need to normalise portfolios to add up to unity has been previously discussed, and backed up by the papers [53], [10]. Every portfolio must be normalised every generation and after every crossover/mutation. The normalisation method had to make sure it kept the integrity of the weights, this meant that if a portfolio before normalisation had weights such as [0.1, 0.09, 0.01, 0.6, 0.4] the last two weights should still remain the greater weights, and not set them all to [0.2, 0.2, 0.2, 0.2, 0.2]. Another feature the normalisation method had to factor in is that no weight should go below 0.01 (1%), therefore the method cannot just reduce each weight by $x \div 5$, if x is equal to the difference between the sum of the portfolio’s weights and 1.

Nevertheless, the normalisation method first checks the portfolio for weights that are less than or equal to 0.01, if there are any add 0.02 to them. The next step is to find the percentage of each weight with regards to the sum of the portfolio weights and store each of these results in an array ’*perc[]*’. The program then checks to see if the sum of the portfolio weights is > 1. If so, set a variable ’*diff*’ to equal sum - 1. This will be the difference between 1 and the sum of the portfolio weights. Next create a variable ’*each*’ to split ’*diff*’ between the size of the portfolio (5). The next step is to loop through each weight of the portfolio and multiply their ’*diff*’ with their percentage difference (’*newPerc*’) between their weight and the sum (’*perc[thisWeight]*’). Then you can set the weight to the weight - ’*newPerc*’.

If the portfolio weights are < 1, the same rule is applied however ’*diff*’ is equal to 1 - sum, and each weight is changed by: eachWeight = eachWeight + ’*newPerc*’.

A code snippet of this is in the appendix of this document (A.8.1).

Once the initial portfolios are created and normalised, the risk of each portfolio must be calculated, this is because the lowest risk of every portfolio created is used in the fitness method. The method ’standardDev()’ calculates the risk of each portfolio, the calculations to do this are on the next page.

Risk of portfolio: The calculated risk of a portfolio as calculated using the source [5].

$$\begin{gathered}r_\rho = (((\omega_\imath\sigma_\imath)^2 + (\omega_\jmath\sigma_\jmath)^2 + (\omega_\kappa\sigma_\kappa)^2 + (\omega_\ell\sigma_\ell)^2 + (\omega_\mathrm{m}\sigma_\mathrm{m})^2)+ \\ ((\omega_\imath\omega_\jmath\sigma_\imath\sigma_\jmath) + (\omega_\imath\omega_\kappa\sigma_\imath\sigma_\kappa) + (\omega_\imath\omega_\ell\sigma_\imath\sigma_\ell) + (\omega_\imath\omega_\mathrm{m}\sigma_\imath\sigma_\mathrm{m}) + (\omega_\jmath\omega_\kappa\sigma_\jmath\sigma_\kappa)+ \\ (\omega_\jmath\omega_\ell\sigma_\jmath\sigma_\ell) + (\omega_\jmath\omega_\mathrm{m}\sigma_\jmath\sigma_\mathrm{m}) + (\omega_\kappa\omega_\ell\sigma_\kappa\sigma_\ell + (\omega_\kappa\omega_\mathrm{m}\sigma_\kappa\sigma_\mathrm{m}) + (\omega_\ell\omega_\mathrm{m}\sigma_\ell\sigma_\mathrm{m}))\end{gathered} \tag{5.1}$$

$$r_\rho = \sqrt{r_\rho} \tag{5.2}$$

where:

- r$_\rho$ is the risk of the portfolio
- ω is a given weight
- σ is a standard deviation of the company
- $\imath$ is company $\imath$
- $\jmath$ is company $\jmath$
- κ is company κ
- ℓ is company ℓ
- m is company m

There are a number of ways to find the lowest risk between all portfolios, but the easiest is when the initial population is created, create a global variable '*lowestRisk*', then check each individuals risk, run a simple '*if*' statement to see if the portfolio's risk is lower than the global '*lowestRisk*', if it is then set '*lowestRisk*' to equal the the portfolio's risk, and repeat this for every individual.

5.5.7 Creating the fitness function

Once the lowest possible risk has been found, the fitness of each portfolio can be found. As previously discussed (4.4), Genetic Algorithms uses the selection technique to pick the fittest individuals, but what is the fittest individual in a series of random numbers? For this, we have to map each gene of the individual to some values, this will allow us to calculate which is the fittest and which is the weakest. The goal of this project is to create a portfolio that searches for the lowest risk with the highest return. Therefore we use this thought to calculate our fitness method.

To calculate the fitness of a portfolio we have to first have access to the portfolio's weights, in order to assess them, and we also must have the mean returns of each company. Remember, weight 1 of a portfolio directly corresponds to company 1, weight 2 of a portfolio corresponds to company 2 and so on.

The method ‘calcFitnessOfPortfolio()’ finds the fitness of each portfolio when called. Firstly, the method finds what the return of the portfolio is by calling the method ’calcReturnOfPortfolio()’. The method, sets a ’double’ variable ’**retrn1**’ to equal the return of the portfolio, using the equation (5.5.7), the equation for this method can be found below followed by the code used in this program.

Return of Portfolio: The expected returns of a portfolio can be found by the the weights multiplied by the mean returns of each individual. Expected returns therefore has an equation of:

$$\mathrm{R}_\rho = (\omega_\iota\ \mu_\iota) + (\dots * \dots) + (\omega_\nu\ \mu_\nu)$$

where:

- R_ρ is a given portfolio
- ω is a given weight
- ι is a given company
- μ is a mean percentage return
- ν is the amount of companies

```
static double calcReturnOfPortfolio(double[] portfolio) {
    double return1 = (portfolio[0] * means[0]) + (portfolio[1] * means[1]) +
(portfolio[2] * means[2]) + (portfolio[3] * means[3]) + (portfolio[4] *
means[4]);
    return return1;
}
```

The fitness function is calculated over 3 parts, firstly a variable '*first*' is created to hold the first part of the equation: '(1 - ψ) * **lowestRisk**', ψ represents a penalty factor, this is to normalise the expected returns, the idea of a penalty factor has been used by both Yahaya, 2009, ([10]) and Sefiane et al, 2012, ([53]). Correlating with these papers, it was found that a penalty factor with the a value of 0.5-1 was best. The second part of this equation starts by creating another variable called '*second*' and computes the equation below:

$$
\begin{aligned}
second = \psi * (((\omega_\imath \bar{R}_\imath) - retrn1) + ((\omega_\jmath \bar{R}_\jmath) - retrn1)+ \\
((\omega_\kappa \bar{R}_\kappa) - retrn1) + ((\omega_\ell \bar{R}_\ell) - retrn1) + (\omega_{\mathrm{m}} \bar{R}_{\mathrm{m}}) - retrn1))); \quad (5.3)
\end{aligned}
$$

where:

- $\bar{R}_\imath$ is the average of the mean expected results for company $\imath$
- ψ is the penalty factor
- $retrn1$ is the return of the portfolio

Finally, a third variable '*third*' is created which adds '*first*' and '*second*' together. The fitness is then returned, the fitness of a portfolio can be between -1 and 1. The full method can be found below.

```
1   public static double calcFitnessOfPortfolio (double[] portfolio , double mean1,
    double mean2, double mean3, double mean4, double mean5) {
2       double weight1 = portfolio [0]; double weight2 = portfolio [1];
3       double weight3 = portfolio [2]; double weight4 = portfolio [3];
4       double weight5 = portfolio [4];
5       double port [] = {weight1, weight2, weight3, weight4, weight5};
6       double retrn1 = calcReturnOfPortfolio (port);
7       double lRisk = lowestRisk;
8       double fitness = 1;
9       double first = (1 − penalty) ∗ lRisk;
10      double second = penalty ∗ (((weight1 ∗ mean1) − retrn1)
                + ((weight2 ∗ mean2) − retrn1) + ((weight3 ∗ mean3) − retrn1)
                + ((weight4 ∗ mean4) − retrn1) + ((weight5 ∗ mean5) − retrn1));
11      double third = first + second;
12      fitness = third;
13      if ( fitness < lowestPortfolioFitness ) {
14          lowestPortfolioFitness = fitness;
15          lowestPortfolioWeights [0] = weight1;
16          lowestPortfolioWeights [1] = weight2;
17          lowestPortfolioWeights [2] = weight3;
18          lowestPortfolioWeights [3] = weight4;
19          lowestPortfolioWeights [4] = weight5;
        }
20      return fitness;
    }
```

Inside the 'calcFitnessOfPortfolio()' on lines 13-19 in the code snippet, is also a check to assign two variables 'lowestPortfolioFitness' and 'lowestPortfolioWeights' to global variables if the fitness of the current portfolio is better than the previous fittest portfolio. Once the portfolios are normalised, the risk and fitness of each portfolio is calculated, the portfolios are then stored in an ArrayList of type double[], the ArrayList is called 'population'.

An ArrayList is known as a 'Dynamic Array' this means the size of the array can change, elements in the array can be added, removed and sorted. As the population of portfolios would constantly have parent portfolios removed and child portfolios added, the ArrayList's dynamicity was exactly what was needed. Another reason the ArrayList was needed, is that the program had to have the ability to check each portfolio's fitness against all other portfolios for the 'selection' genetic operator, to determine which portfolios to reproduce based on their fitness.

The 'init()' method then runs a while loop used to replicate the generation changes, each iteration through the while loop is another generation. The number of generations can be determined by the user, a generation count of 1,000,000 has been found to be optimal.

5.5.8 Selection

Through each generation the program selects two portfolios to reproduce, for this program a tournament style selection rule was used. The program finds four random portfolios, checks that they are not the same, calculates the fitness of each portfolio, and chooses the fittest two individuals to reproduce. This selection method continues through every generation. A code snippet of the selection method is below.

```
public static double[] selection(double[] portfolio1, double[] portfolio2)
throws FileNotFoundException, IOException {
    double portfolio1Fitness = calcFitnessOfPortfolio(portfolio1, means[0],
means[1], means[2], means[3],
            means[4]);
    double portfolio2Fitness = calcFitnessOfPortfolio(portfolio2, means[0],
means[1], means[2], means[3],
            means[4]);
    double[] bestPortfolio;
    if (portfolio1Fitness < portfolio2Fitness) {
        bestPortfolio = portfolio1;
    } else {
        bestPortfolio  =  portfolio2 ;
    }
    return  bestPortfolio ;     }
```

5.5.9 Crossover

Once two portfolios are selected, the program checks to see if a crossover condition has been met. The crossover method does not get called every generation, but gets called at chance. This chance is called the Crossover Rate, through the testing of this project it was found that a crossover rate of 0.1% performed best as seen in section 6.3. There are also three different types of crossover methods that were tested, ‘Single-Point’, ‘Two-Point’ and ‘Whole Arithmetic’, it was found that ’Single-Point’ performed the best, see section 6.1 for evaluation results.

The ‘onePointCrossover()’ method takes two portfolios, generates a random ’crossover point’ in the portfolio between portfolio[1] and portfolio[4], the program then loops from the crossover point to the end of the portfolio, for each iteration of the loop(i), select portfolio1[i] and swap it with portfolio2[i], and vice versa. Code for this method can be found below.

```
public  static  ArrayList<double[]> onePointCrossover(double[]  portfolio1 ,
double[]  portfolio2 ) {
        Random rn = new Random();
        ArrayList<double[]> twoChildren  = new ArrayList<double[]>() ;
        // Select  a  random crossover  point
        int  crossOverPoint  = rn. nextInt ( portfolio1 . length ) ;
        double[]  childOne  =  portfolio1 ;
        double[]  childTwo  =  portfolio2 ;
        // Swap values  among parents
        for  ( int  i  = crossOverPoint ;  i  > 0;  i−−) {
            double  temp = childOne[ i ];
            childOne[ i ]  = childTwo[ i ];
            childTwo[ i ]  = temp;
         }
         twoChildren . add(childOne) ;
         twoChildren.add(childTwo);
         return twoChildren;
     }
```

The ‘twoPointCrossover()’ method also takes two portfolios, but also generates a random ’end crossover point’ between portfolio[0] and portfolio[4], it then generates a random ’start crossover point’ between the ’portfolio[1]’ and the ’end crossover point’.

The program loops through the ’start crossover point’ and the ’end crossover point’, and swaps portfolio1[i] with portfolio2[i]. Resulting in only the section between the start and end crossover points being swapped. There is a code snippet below.

```
public static ArrayList<double[]> twoPointCrossover(double[] portfolio1,
double[] portfolio2) {
        Random rn = new Random();

        // Select a random crossover point
        double[] childOne = portfolio1;
        double[] childTwo = portfolio2;
        ArrayList<double[]> twoChildren = new ArrayList<double[]>();
        int crossOverEnd = rn.nextInt(portfolio1.length - 1) + 1;
        int crossOverStart = rn.nextInt(crossOverEnd) + 1;
        // Swap values among parents
        for (int i = crossOverStart; i <= crossOverEnd; i++) {
            double temp = childOne[i];
            childOne[i] = childTwo[i];
            childTwo[i] = temp;
        }
        twoChildren.add(childOne);
        twoChildren.add(childTwo);
        return twoChildren;
}
```

Finally, the ‘wholeArithmeticCrossover()’ method takes two portfolios and for every weight, it measures the average weight between the two, and places this averaged weight into the child portfolio, once again the code snippet of this method can be below, the figure (5.5) shows how this technique works.

```
public static ArrayList<double[]> wholeArithmeticCrossover(double[]
  portfolio1 , double[] portfolio2 ) {
      // Select a random crossover point
      double[] childOne = new double[5];
      double[] childTwo = new double[5];
      ArrayList<double[]> twoChildren = new ArrayList<double[]>();
      double avg1 = 0.0;
      double avg2 = 0.0;
      double avg3 = 0.0;

      // Swap values among parents
      for (int i = 0; i < portfolio1.length; i++) {
          avg3 = ( portfolio1 [i] + portfolio2 [i]) / 2;
          childOne[i] = avg3;
          childTwo[i] = avg3;
      }
      twoChildren.add(childOne);
      twoChildren.add(childTwo);
      return twoChildren;
}
```

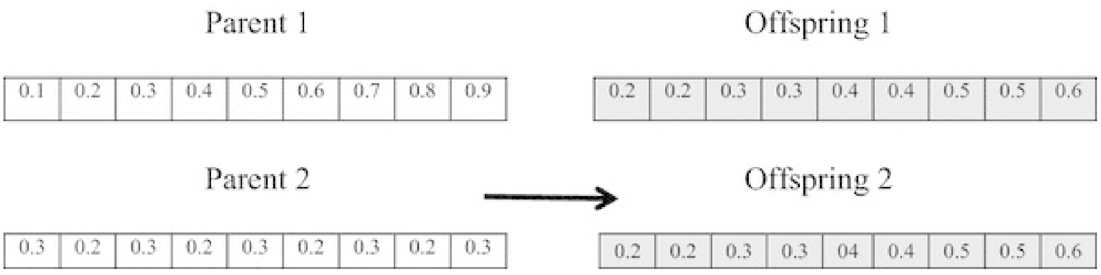


Figure 5.5: Whole-Arithmetic Crossover of two parents, combining their genetic information and reproducing to create two children. Gene 1 for Parent 1 is equal to 0.1, whilst Gene 1 for Parent 2 equals 0.3. The median between these two Genes is 0.2 therefore the genetic information for Gene 1 on both offspring is equal to 0.2. [28].

After this crossover method, the two parents are removed from the population (killed off), the risk and fitness of the new child portfolios are calculated, and the child is added to the population, the child is also checked for normalisation.

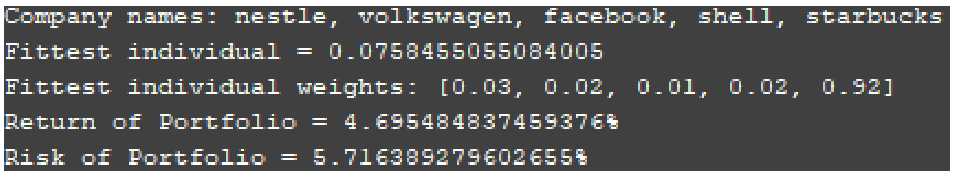
Given only a crossover, it is correct to think that, after a certain amount of time, all portfolios will eventually be more or less the same, as they will all have similar parents through the selection of only the fittest individuals. The way to make sure this doesn't happen is by using the mutation operator to 'stir the pot'.

5.5.10 Mutation

If a mutation condition has been met, the 'mutation()' method will get called. This chance of the mutation method being called is named the 'Mutation rate', and unlike the crossover rate, the best mutation rate was 100%, see section 6.4 for evaluation results and a discussion regarding this mutation rate. The mutation method works by taking a single portfolio, generating a random value between 0.01 and 0.96, generating a random point in the portfolio and swapping the weight at that random point in the portfolio with the random value between 0.01 and 0.96, the code for the mutation method is below.

```java
public static double[] mutation(double[] portfolio1 ) {
        Random rn = new Random();
        double[] newMutatedPortfolio = portfolio1 ;
        int mutationPoint = rn. nextInt ( portfolio1 . length );
        Double randomMutWeight = rn.nextDouble();
        while(randomMutWeight >= 0.96) {
            randomMutWeight = rn.nextDouble();
        }
        newMutatedPortfolio[ mutationPoint ] = randomMutWeight;
        double sum = 0.0;
        double each = 0.0;
        return newMutatedPortfolio;
}
```

Once, the generation count has been reached the program will look output the global variables 'lowestPortfolioFitness' and 'lowestPortfolioWeights', as well as the company names that each weight is representing and the expected risk and returns of the portfolio, this can be seen in figure (5.6). This portfolio will be the fittest individual portfolio over the duration of the program.



```
Company names: nestle, volkswagen, facebook, shell, starbucks
Fittest individual = 0.0758455055084005
Fittest individual weights: [0.03, 0.02, 0.01, 0.02, 0.92]
Return of Portfolio = 4.695484837459376%
Risk of Portfolio = 5.716389279602655%
```

Figure 5.6: The final output of the program, showing the company names, the fitness of the fittest individual, the weights of the portfolio as well as the risk and return percentage of the portfolio. The output showing that the user should place 3% of their budget on 'nestle', 2% of their budget on 'volkswagen', 1% of their budget on 'facebook', 2% on 'shell' and 92% on 'starbucks'. This portfolios expected return is 4.65 % and the risk is 5.71 %.

5.6 Discussion

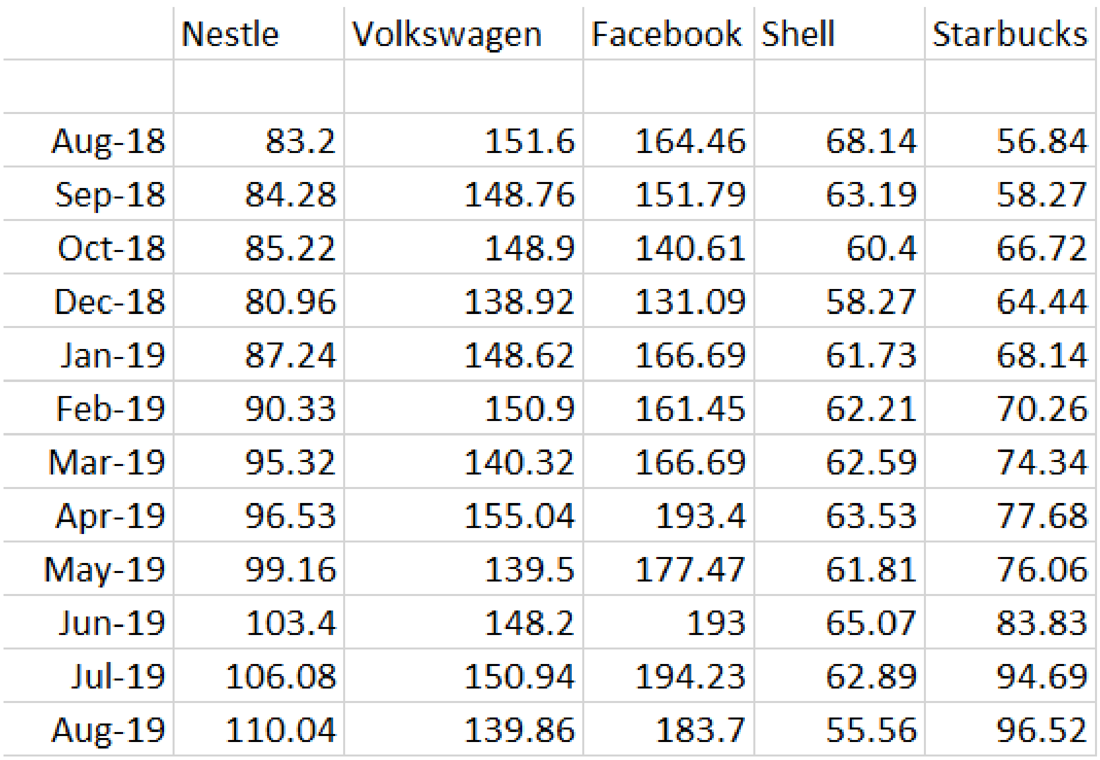
This chapter discussed the design and implementation of 'GenStock' using Web Scraping, Named Entity Recognition, Sentiment Analysis and a Genetic algorithm to produce a stock portfolio that achieves the optimal result based on historical data. Java was used in conjunction with jsoup and Selenium WebDriver as a way to source input data from the internet without the need for a large amount of storage space.

The output of the program shows the company names, the highest overall fitness throughout all generations, the exact weights for each company relating to a users budget, as well as the expected return and expected risk of the portfolio as a whole, based upon the Standard Deviation and Mean-Variance model created by Harry Markowitz [30].

The Genetic Algorithm section of this chapter discussed the selection, mutation and crossover techniques used as well as providing pseudo-code and detailed code snippets. Three crossover techniques were developed in the implementation of this project, Single-Point crossover, Two-Point Crossover and Whole-Arithmetic Recombination. The evaluation of these techniques will be made in the following chapter.

Evaluation

The series of experiments ran in this chapter use real historical data from companies, acquired from the web scraping subsection of this project. The companies include: ‘Nestle’, ‘Volkswagen’. ‘Facebook’, ‘Shell’ and ‘Starbucks’, the historical data is shown in figure 6.1.



	Nestle	Volkswagen	Facebook	Shell	Starbucks
Aug-18	83.2	151.6	164.46	68.14	56.84
Sep-18	84.28	148.76	151.79	63.19	58.27
Oct-18	85.22	148.9	140.61	60.4	66.72
Dec-18	80.96	138.92	131.09	58.27	64.44
Jan-19	87.24	148.62	166.69	61.73	68.14
Feb-19	90.33	150.9	161.45	62.21	70.26
Mar-19	95.32	140.32	166.69	62.59	74.34
Apr-19	96.53	155.04	193.4	63.53	77.68
May-19	99.16	139.5	177.47	61.81	76.06
Jun-19	103.4	148.2	193	65.07	83.83
Jul-19	106.08	150.94	194.23	62.89	94.69
Aug-19	110.04	139.86	183.7	55.56	96.52

Figure 6.1: These figures are the monthly stock price for each respective company in US dollars, the prices were taken at closing from the 1st of each month(1 Aug 2018 - 1 Aug 2019).

6.1 Experiment 1 - Investigating different Crossover techniques

This experiment was done to determine whether or not the crossover method used made a difference worth discussing. This experiment is similar to the ones ran in Sefiane et al's paper [53], the experiments involve the use of all three crossover techniques (Single-Point, Two-Point and Whole-Arithmetic), Sefiane et al discussed that the Whole-Arithmetic crossover technique worked best, so it is hypothesised that this will be the case in this project as well.

These experiments were ran with an initial population of 50, whilst having 1,000,000 generations. The mutation rate and crossover rate stayed constant for these two experiments: Mutation rate = 0.1%, Crossover rate = 0.1%, these rates were chosen as the experiment to test which rate produced the best results had not been completed yet. These are the rates in which Sefiane et al [53] and Yahaya et al [10] claimed were the most optimal. Results were returned every 10000 generations as 1,000,000 results would be too many for any visual graphic.

The first experiment was using the Single-Point Crossover technique, the results can be seen in figure (6.2).

As you can see in figure (6.2), the best fitness increased from 0.028 to 0.063 as more generations were added. It is also clear that the best portfolio results started to slow down, this could be due to the low crossover and mutation rate or even the relatively low generation count. If the crossover rate is too low, it will mean that portfolios will not reproduce and spread their genetic information, therefore if the best portfolio cannot reproduce it will not be able to pass its good traits on to future generations, thus the best portfolio will plateau.

The worst fitness is also interesting, the worst fitness actually gets worse as generations goes on. The reason for this is due to the selection method that was chosen for this project. The tournament style selection method only picks the fittest individuals to reproduce with each other, therefore if an individual is unfit, it will never get picked to reproduce with a fitter individual and become stagnant at the bottom of the fitness tree.

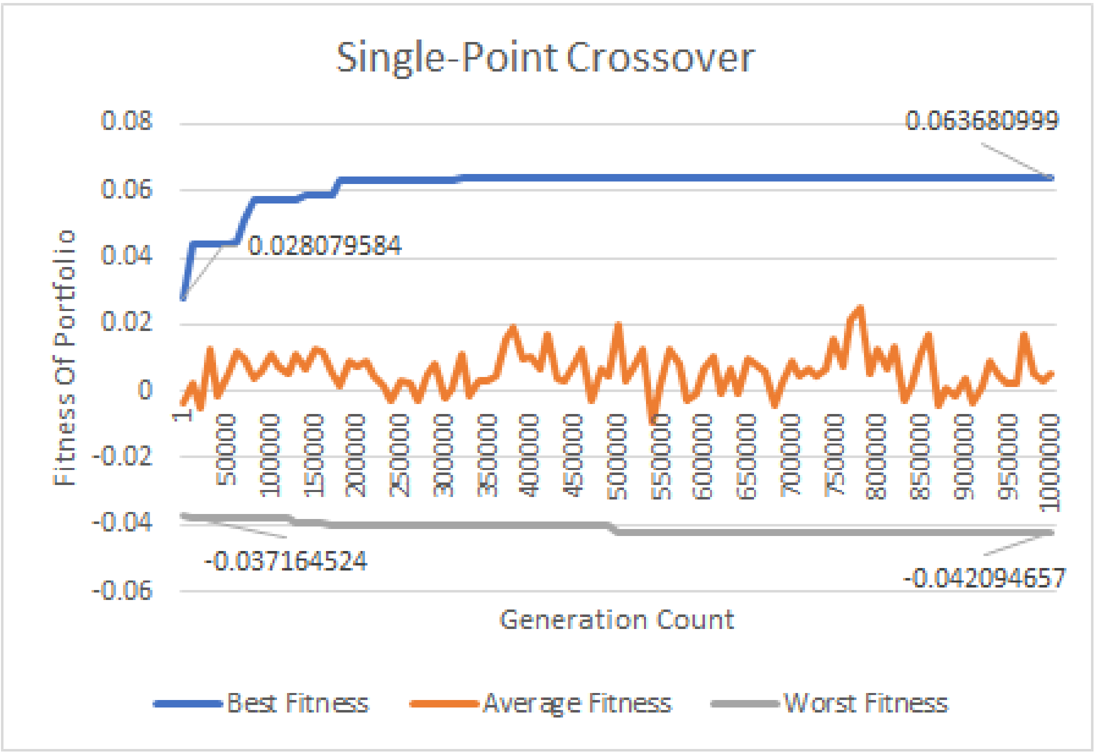
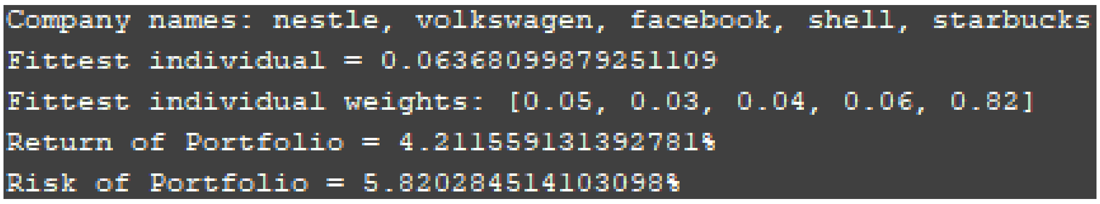


Figure 6.2: Single-Point Crossover with a rate of 0.1%

However, the mutation modifier does not prejudice against a portfolios fitness, meaning the portfolios that are unfit cannot gain 'known' good traits from reproduction, and they can also get worse traits through mutation changing their weights. The worst fitness does not matter in this project as it is only the best portfolio that is chosen, however it is worth discussing.

The output from this experiment can be seen in figure (6.3).



```
Company names: nestle, volkswagen, facebook, shell, starbucks
Fittest individual = 0.06368099879251109
Fittest individual weights: [0.05, 0.03, 0.04, 0.06, 0.82]
Return of Portfolio = 4.211559131392781%
Risk of Portfolio = 5.820284514103098%
```

Figure 6.3: The output from the program showing the companies involved, the fitness of the portfolio shown, the portfolio itself, and the risk and return percentages of said portfolio. The results show that this particular portfolio created, has a return of 4.21% and a risk of 5.82%, suggesting that the user should place 82% of their budget on shares in Starbucks.

This output correlates with the test data from figure (6.1), this figure shows that Starbucks has the safest and best returns from the other companies, therefore a portfolio should be created that has Starbucks as their main asset.

The next experiment was completed using the Two-Point Crossover technique (6.4).

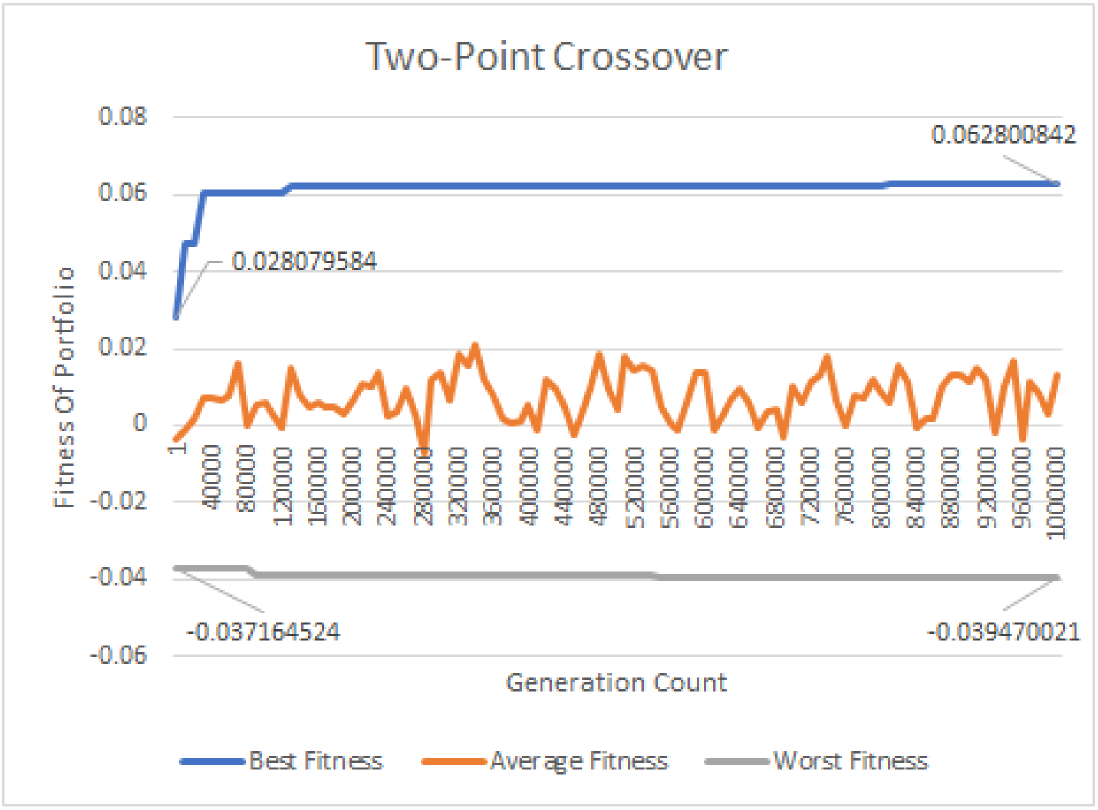


Figure 6.4: Two-Point Crossover with a rate of 0.1%

The best fitness for this crossover technique was lower than the Single-Point crossover technique. There seems to be not much difference in the fluctuations of the average fitness results between both Single-Point crossover and Two-Point crossover, this is expected due to the nature of the mutation operation making a portfolios weights random for better or for worse.

The last experiment was done with a Whole-Arithmetic Crossover technique (6.5);

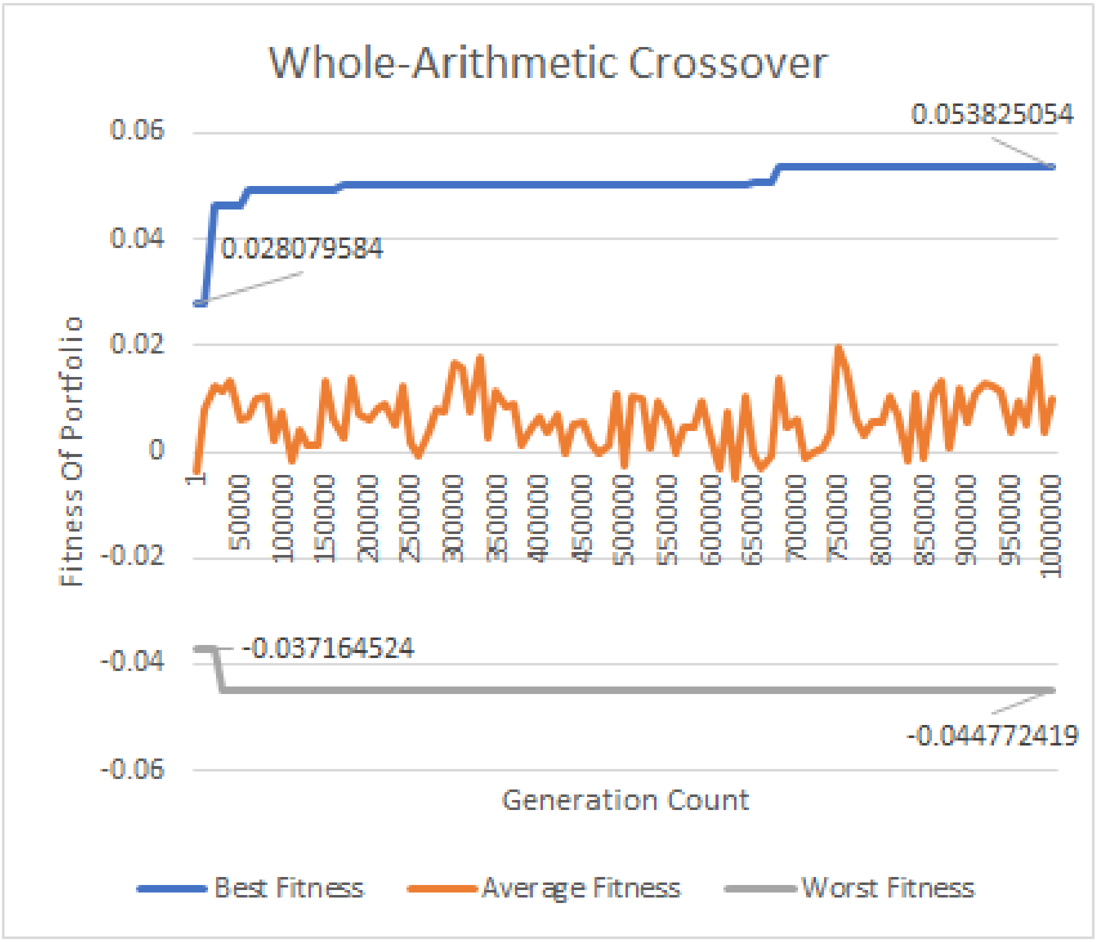


Figure 6.5: Whole-Arithmetic Crossover with a rate of 0.1%

This result is surprising, considering Sefiane et al [53] claimed it would be the most optimal. It can be assumed that this is due to the selection method also, Sefiane et al used the Roulette Wheel selection method, whereas this project used the Tournament Style selection method.

The best portfolio didn't change at all throughout 1,000,000 generations. It is worth noting however that the average fitness was a lot less erratic, in the Single-Point and Two-Point crossover methods, the average fitness for Single-Point crossover ranged between -0.01 and 0.025 having a range of 0.035, whilst the average fitness for Two-Point crossover ranged between -0.01 and 0.02 having a range of 0.03. The Whole-Arithmetic crossover method results varied from -0.005 to 0.0199 thus having a range of only 0.0249.

6.2 Experiment 2 - Comparing GenStock vs Manual portfolio selection

For this series of experiments four portfolios are created manually 6.6. These will be tabled alongside a portfolio created by GenStock, and the results will be compared and discussed in terms of two factors: time, and accuracy.

6.2.1 Comparing Time

The two sub-experiments below will consist of; GenStock creating a five asset portfolio, with no company names inputted by the user and using the web scraping tool to scrape news articles, the second experiment will consist of 5 user inputted company names (for fairness they will be the same company names as the web scraper returned: ’Nestle’, ’Volkswagen’, ’Facebook’, ’Shell’ and ’Starbucks’).

Experiment 1 (Time): The time from when the programs ’Run’ button is pressed to when a portfolio is returned will be measured, with and without the web scraping tool and compared with manual portfolio selection. The internet speed of the computer that ran this experiment is as followed: Ping: 29ms, Download: 10.30 Mbps, Upload: 0.85 Mbps. Since this experiment uses the internet, it is clear that with a faster internet connection this experiment will run faster.

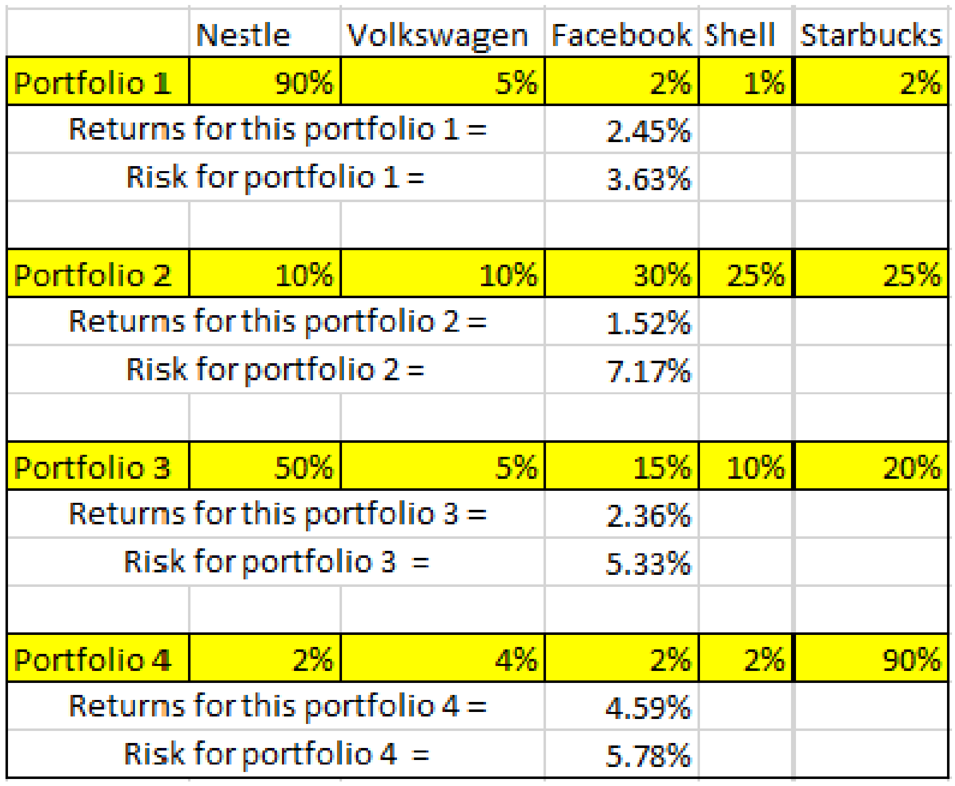
Using Web Scraper: 00:03:31.10

Without using Web Scraper: 00:03:17.21

This experiment shows that GenStock is 2142 % faster than manual portfolio selection whilst using the web scraping tool (3 minutes 31 seconds vs 75 minutes), and approximately 2307 % faster without using the web scraping tool (3 minutes 17 seconds vs 75 minutes).

6.2.2 Comparing Accuracy

To measure accuracy between manual portfolio selection and GenStock portfolio selection, these experiments will look at their returned risk/returns for comparison. A positive result for this experiment would show that the GenStock portfolio selection has a higher percentage return and a lower percentage risk than the manual selections. The manually chosen portfolios are seen in figure 6.6.



	Nestle	Volkswagen	Facebook	Shell	Starbucks
Portfolio 1	90%	5%	2%	1%	2%
Returns for this portfolio 1 =			2.45%		
Risk for portfolio 1 =			3.63%		
Portfolio 2	10%	10%	30%	25%	25%
Returns for this portfolio 2 =			1.52%		
Risk for portfolio 2 =			7.17%		
Portfolio 3	50%	5%	15%	10%	20%
Returns for this portfolio 3 =			2.36%		
Risk for portfolio 3 =			5.33%		
Portfolio 4	2%	4%	2%	2%	90%
Returns for this portfolio 4 =			4.59%		
Risk for portfolio 4 =			5.78%		

Figure 6.6: This figure shows 4 manually created portfolios, with various different weightings for each company (company names at the top). This figure also shows the expected returns and expected risks of each of these portfolios. In yellow are the percentage weights for each company, making up a five asset portfolio. The returns of the portfolio are using the calculations as seen in 5.5.7. Portfolio 4 has the best return percentage however it has a higher risk percentage than Portfolio 3, using traditional methods, a stockbroker would display both of these portfolios to a client and let them choose between a riskier portfolio or a safer one. Using Harry Markowitz' Efficient Frontier ([30]) a diagram can be displayed to determine which portfolios should be disregarded. 6.7.

The next experiment is to test GenStocks portfolio selection against these manual results. Using a 1% Crossover rate, and a 1% Mutation rate, which is shown to have the most accurate results (6.3, 6.4), as well as an initial population of 500,000 portfolios, Genstock created the portfolio seen in figure 6.8.

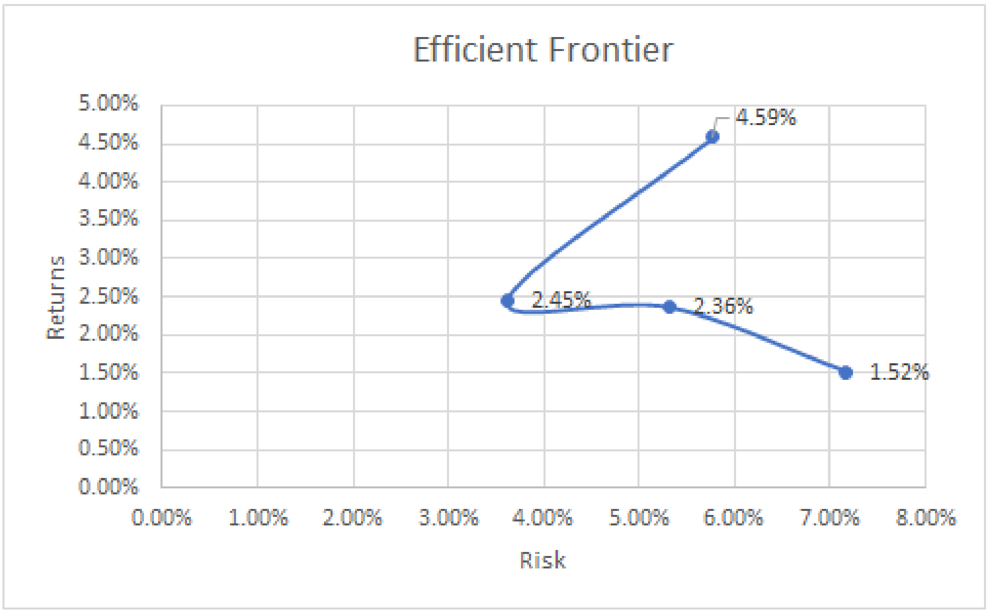
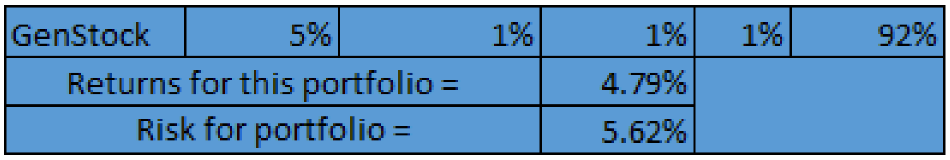


Figure 6.7: This diagram shows the efficient frontier for this set of portfolios, with the return percentage on the vertical axis and the risk percentage on the horizontal axis. The diagram shows that Portfolio 3 (2.36%) and Portfolio 2 (1.52%) should be disregarded, as for the same risk a portfolio exists with a greater return. The two portfolios that would be viable for an investor are Portfolio 1 and Portfolio 4.



GenStock	5%	1%	1%	1%	92%
Returns for this portfolio =			4.79%		
Risk for portfolio =			5.62%		

Figure 6.8: Comparing this portfolio with the manual portfolios from figure 6.6, it is evident that GenStock has created a portfolio with greater returns and a lower risk than all of the manual portfolios. A diagram shows this alongside the efficient frontier for this set of portfolios 6.9.

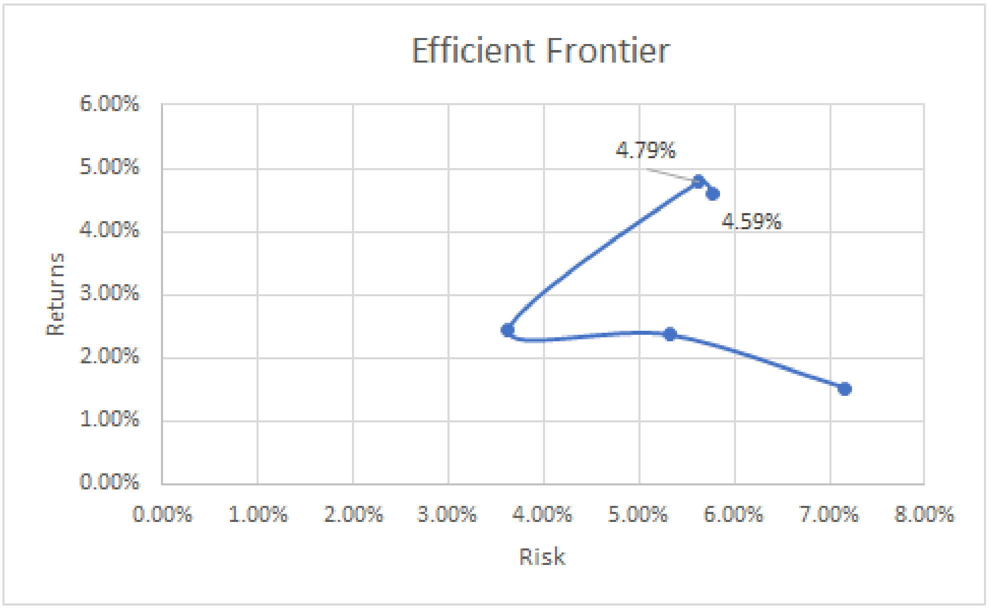
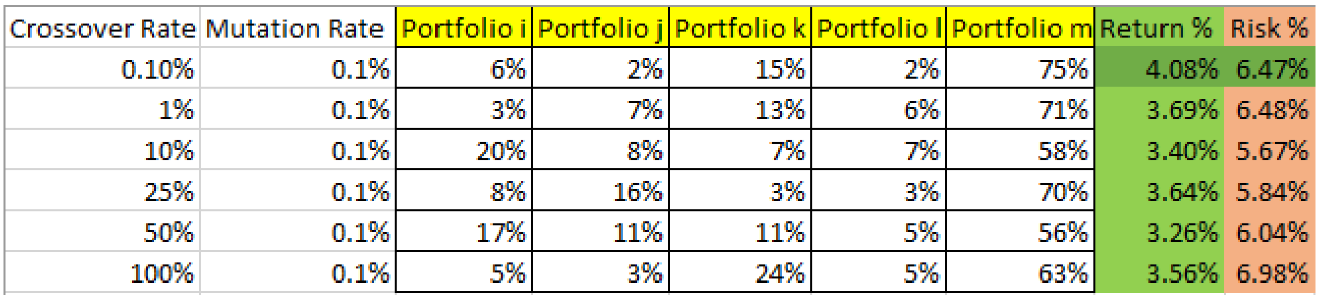


Figure 6.9: GenStock portfolio (4.79%) has a higher return and a lower risk than the fittest portfolio (Portfolio 4 (4.59%)) that was manually created.

6.3 Experiment 3 - Investigating different Crossover rates

These experiments involved changing the rate at which two portfolios would crossover, and ultimately aim to find the optimum crossover rate at an initial population of 10, running through 1,000,000 generations. The ten initial portfolios were the same every run, the highest initial fitness within the population was 0.028. These experiments all ran with a mutation rate of 0.1%, and crossover rates that ranged from 0.1%, 1%, 10%, 25%, 50% AND 100%. Single-point crossover was also used for all of these experiments, as results (6.1) show it outperformed Two-Point crossover and Whole-Arithmetic Crossover.

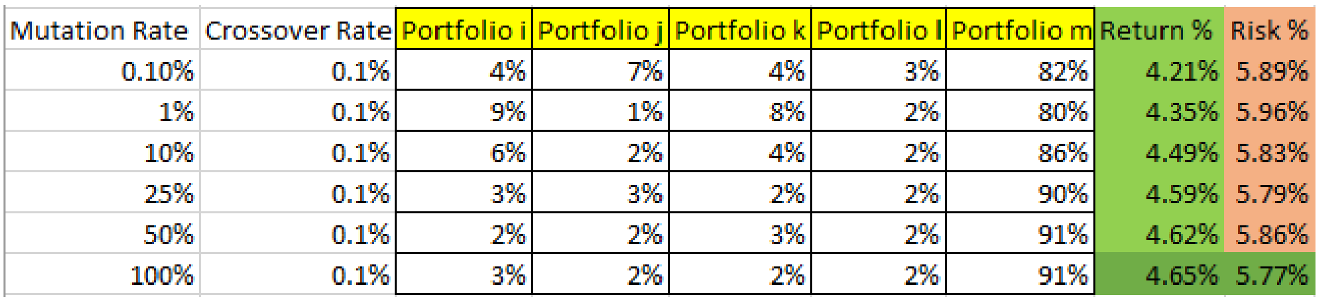


Crossover Rate	Mutation Rate	Portfolio i	Portfolio j	Portfolio k	Portfolio l	Portfolio m	Return %	Risk %
0.10%	0.1%	6%	2%	15%	2%	75%	4.08%	6.47%
1%	0.1%	3%	7%	13%	6%	71%	3.69%	6.48%
10%	0.1%	20%	8%	7%	7%	58%	3.40%	5.67%
25%	0.1%	8%	16%	3%	3%	70%	3.64%	5.84%
50%	0.1%	17%	11%	11%	5%	56%	3.26%	6.04%
100%	0.1%	5%	3%	24%	5%	63%	3.56%	6.98%

Figure 6.10: This figure shows GenStock generated portfolio weights (yellow), a return percentage for the portfolio as a whole (light green) and a risk (orange). On the left of the figure, there are the corresponding crossover and mutation rates that were used to generate each respective portfolio. The fittest portfolio was created when the crossover rate was at 0.1%. The return percentage of this portfolio was 4.08% and the risk percentage was 6.47% (dark green), this portfolio had a fitness of 0.06159.

6.4 Experiment 4 - Investigating different Mutation rates

These experiments involved changing the rate at which a portfolio would be mutated, and ultimately aim to find the optimum crossover rate at an initial population of 10 and running 1,000,000 generations. To keep it fair These experiments all ran with a crossover rate of 0.1%, and mutation rates that ranged from 0.1%, 1%, 10%, 25%, 50% and 100%.



Mutation Rate	Crossover Rate	Portfolio i	Portfolio j	Portfolio k	Portfolio l	Portfolio m	Return %	Risk %
0.10%	0.1%	4%	7%	4%	3%	82%	4.21%	5.89%
1%	0.1%	9%	1%	8%	2%	80%	4.35%	5.96%
10%	0.1%	6%	2%	4%	2%	86%	4.49%	5.83%
25%	0.1%	3%	3%	2%	2%	90%	4.59%	5.79%
50%	0.1%	2%	2%	3%	2%	91%	4.62%	5.86%
100%	0.1%	3%	2%	2%	2%	91%	4.65%	5.77%

Figure 6.11: This figure shows GenStock generated portfolio weights (yellow), a return percentage for the portfolio as a whole (light green) and a risk (orange). On the left of the figure, there are the corresponding crossover and mutation rates that were used to generate each respective portfolio. The fittest portfolio was created when the mutation rate was at 100%, this portfolios fitness was 0.07502 and had a return of 4.65% and a risk of 5.77%.

6.5 Discussion Of Evaluation Experiments

Results show that a mutation rate of 100% and a crossover rate of 0.1% was found that with a greater initial population, a portfolio with higher return and lower risk would be found. It is very shocking that a mutation rate of 100% was found to return the best results. However, it is thought that the way the lowest fitness portfolio is stored as a global variable is responsible for this. It means, that whilst on the 1,000,000 generation for example, the fittest portfolio of that generation may be extremely low, but in the stored global variable, a portfolio could have been found from a past generation that was found to be the fittest and stored. The method used for this to work can be found in the appendix of this paper (A.8.6). With regards to the different crossover functions, The Single-Point crossover function has slightly better results on average per generation than the Two-Point crossover function and the Whole-Arithmetic crossover function.

All of these experiments have a positive linear relationship between both: the risk of a portfolio and the generation count. And, the return of a portfolio and the generation count. However, this relationship does tend to plateau at with a high crossover rate. The results show that the genetic algorithm part of this program works, by having a linear relationship between the risk/return and the generation count you can see that as time goes on and the children of the previous generation become fitter, the portfolio becomes more rewarding for the investor.

As discussed, the ‘Time’ experiments shows that this program is 2142 % times faster than manual portfolio selection whilst using the web scraping tool (3 minutes 31 seconds vs 75 minutes), and approximately 2307 % times faster without using the web scraping tool (3 minutes 17 seconds vs 75 minutes), and the ‘Accuracy’ experiments show that it returned a very efficient portfolio each time, using Harry Markowitz’ Efficient Frontier model as a framework to evaluate against.

Conclusion

This chapter summarises the dissertation as a whole with references to the evaluation of GenStock. The second section in this chapter reviews the initial aims and objectives set at the start of this project and how they have been met. The limitations of this project is discussed in the third section particularly focusing on the Sentiment Analysis techniques and technological limitations. The fourth section of this chapter talks about the potential improvements that could bolster this project in the future.

7.1 Summary of Dissertation

This project combines, Web Scraping, Natural Language Processing, Sentiment Analysis, Named Entity Recognition and Genetic Algorithms to generate an optimum portfolio of stocks, based on a risk and return fitness function. This program has proved that it can create an optimum portfolio for an investor based on recent news articles as well as past historical data of chosen companies. Java was used to create this program, using the imported libraries: Selenium to scrape historical data and Jsoup to parse business news articles.

Test results show that this program creates an accurate efficient portfolio tested against the well known Efficient Frontier and Mean-Variance model devised by Harry Markowitz (6.2.2). The speed of this program to: research companies, value those companies using current news articles, searching for the companies historical data and put together a portfolio outweighs manual portfolio selection by over 2000% (6.2.1).

7.2 Review of aims and objectives

The aims and objectives originally set at the start of this project are displayed below with an explanation on how they were met.

- Design and implement a web scraping tool to parse through business news articles.

This project created a host crawling tool, to read from business articles from a popular news source and return articles in full. This tool can be ran for as long as the user wishes and stores each article in a txt file local to the user. On the other hand, if the user has a set list of companies they wish to invest in, they can input them straight to the program and cut this tool of web scraping out of the program. The methodology for this part of the program can be found here (5.5.1).

- Design and implement a Text Preprocessing algorithm that removes capitalisation, removes stopwords and removes punctuation from parsed articles using the 'Stopword' list from Python's NLTK.

Text Preprocessing was used to remove capitalisation, punctuation and stopwords from each article. For the removal of Stopwords, Python's NLTK was used and each word in it was checked against each word in every article. If a stopword was found in the article it would be removed safely (5.5.2).

- Design and implement a Named Entity Recognition tool to search the processed articles for company names with help from the research of Lample et al's paper 'Neural Architectures for Named Entity Recognition' [38].

Articles were parsed from the txt file and checked against a preset list of company names, if the company names was found inside the article. The program would store the article and read the article in more detail (5.5.3).

- Design and implement a Sentiment Analysis tool that reads through the processed articles to determine if an article was positive or negative based on Nakov et al's work in their seminal papers SemEval [41].

After the program has successfully recognised company names, the program then checks each word against a AFINN-111, a list of english words used frequently in Sentiment Analysis models. If a positive word is found a positive value will be given to that company and vice versa. The 5 companies with the highest value (the companies with the most positive articles written about them) will be passed on and a portfolio will be created containing them (5.5.4).

- Design and implement a Web Scraping tool to search for historical data using a browser simulation called Selenium [52].

Selenium was used to replicate a Mozilla Firefox browser and searches a popular financial website using each of the five companies names to search with. The program navigates the site and finds the Historical Data of said company, it then returns the last 12 months of the companies data and stores it in an array (5.5.5).

- Design and implement a Genetic Algorithm system based largely on the work by Yahaya [10] and Sefiane et al [53] to find an optimum portfolio given a company list.

A Genetic Algorithm was used to find the fittest portfolios that offer the best return for the least risk. The fitness function was created using the calculations from Yahaya's [10] work in his paper 'Portfolio Selection using Genetic Algorithms'. The program runs for a certain amount of generations, performs crossover and mutation methods on portfolios and returns the portfolio with the highest return for the lowest risk in the population (5.5.6).

- Evaluate the speed against manual portfolio selection and the accuracy of this program against a well known model for testing portfolios, called the Efficient Frontier created by Harry Markowitz [30].

The evaluation section of this paper showed that the speed of the creation of an optimum portfolio using this program outweighed the speed that a manual portfolio selection would be by 2000 %. The accuracy was tested against manual portfolios and used the Harry Markowitz Efficient Frontier model to visualise the efficiency of the portfolios created (6.2).

- Evaluate the fitness of selected portfolios based on manual experiments.

Using real data, multiple crossover techniques, crossover rates and mutation rates were tested. It was found that One-Point crossover was preferable to the Two-Point and Whole-Arithmetic crossover methods (6.1). A crossover rate of 0.1% (6.3) and a mutation rate of 100% (6.4) performed best.

7.3 Limitations

This project has little limitations, however being a software project there are some technical limitations. The first limitation is that any change to the DOM structure of the websites used in the Web Scraping part of ‘GenStock’ could mean the program is unable to retrieve vital historical data as well as read the news articles, severely limiting the effectiveness of this program.

A further technical limitation is Java’s JVM Heap Size. This has been briefly discussed in section (5.5.6), Java’s JVM maximum heap size means there is a maximum population size, roughly around 1,500,000 to 2,000,000. Whilst this could be a significant factor for some other projects, the evaluation section of this paper shows that even with a population size of 50 you can get much improved results.

Another limitation is the Sentiment Analysis model not being able to pick up ’business-savvy’ words and phrases, as the AFINN-111 was manually written it includes lots of words and phrases however it does not include all of them. Therefore, if a news article contained a phrase that the AFINN-111 did not contain then it would not count that word as being positive or negative whereas in reality it could be. This notion of not being able to recognise a word could also be a limitation of the Named Entity recognition model, the NER model checks against a preset list of companies, however if a company exists, yet is not on the list it will be ignored. The NER list should be updated regularly with company names to ease this limitation.

7.4 Future Work

This chapter will discuss potential improvements for the future of this program and will discuss the difficulties of achieving such improvements.

One aspect of this program that could be improved is the Named Entity Recognition section. The problem with it in its current duration is that it requires a preset list of company names, which results in the program ignoring all other companies unless they are on the company list, this could result in a company which would be a valuable asset to an investors portfolio being ignored. One way to overcome this problem is to have a trained NER model which recognises not specific company names but the fact that a word in an article is actually a company, this could be done by using a Bidirectional LSTM as used by Chiu et al [17], these are typically challenging to develop but with the right expertise a Bidirectional LSTM could take this project to a whole other level.

A further aspect that could be changed in this program is to convert the classic 'Portfolio selection' which takes into account the portfolios return and risk using the Mean-Variance model, and replacing it with a 'Fuzzy Portfolio Selection'. Many developers and economists ([18], [31], [16], [50], [47]) have tried to use 'Fuzzy Portfolio Selection' to take other factors into account when finding the optimal portfolio for a specific investor. Calvo et al [18] proposed that the investors choice of whether or not to reject a portfolio based on their own whim could itself be a criterion to add to the 'Fuzzy' selection. The idea of 'Fuzzy Portfolio Selection' offers the investor more options and flexibility however it has the downside of needing an investor to understand the criterion they would be inputting.

This program is currently developed for the developer. Although, if it was to be put out in the public domain, it would certainly need a User Interface, as currently it prints only to a terminal. This could be a potential for future work and the difficulty of creating a UI depends on the perfectionism of the developer as well as the results of user study's that would need to be ran.

AFINN-111 is a great Sentiment Analysis word list for natural language text. Nonetheless, a more 'business savvy' sentiment analysis corpus would be preferred. There are words in the business world that are specifically only said in the business world, and for this reason are absent on the AFINN-111 word list. A Sentiment Analysis word list takes a long time to create and a tremendous amount of difficulty to perfect. A solution could be to manually add business words such as ‘deficit’ and ‘bear-market’ with values suitable to such words, into the AFINN-111 word list.

The Web Scraping/Host crawling section of this program that scrapes historical data from companies using the web browser ‘Selenium’ could be improved by simply having a faster internet connection. In the ‘Time’ evaluation test (6.2.1) in this paper, it showed that using a relatively modest internet connection the speed of the Host Crawling was just over 3 minutes and 15 seconds, with access to a faster internet connection there would be no doubt this would speed up.

Another possible future improvement would be to use threading in the creation of portfolios. If a generation has 10 portfolios in, 10 threads could be created that are responsible for calculating the fitness of each portfolio, as opposed to the CPU calculating them one at a time. This is known as running in parallel and could greatly speed up this program. Threading is not massively difficult to do, however using too many threads can actually slow down the program due to Amdahl’s law [4].

The final future improvement of this program would be the storage of parsed news articles alongside the company names. Whilst this would take a great deal of storage, the ability to go back over past articles and train the programs sentiment analysis model by adjusting the value of each word with the change per month in the companies stock price. For example, if a company consistently gets bad reviews in news articles, the program could adjust the value of the ‘bad’ words in the Sentiment analysis model to be slightly less negative as they clearly haven’t made a big dent in the companies stock value.

References

[1] 2010. [Online]. Available: https://mendoza.nd.edu/news/new-dow-jones-lexicon-analyzes-patterns-in-the-news-to-create-more-powerful-trading-strategies/ (p. 11).

[2] 2019. [Online]. Available: https://www.bbc.co.uk/news/business-42959755 (p. 3).

[3] 2019. [Online]. Available: https://www.getsmarteraboutmoney.ca/invest/investment-products/stocks/factors-that-can-affect-stock-prices/ (p. 11).

[4] 2019. [Online]. Available: https://en.wikipedia.org/wiki/Amdahl%5C%27s_law (p. 74).

[5] - and Finance Train. (2012). How to calculate portfolio risk and return, [Online]. Available: https://financetrain.com/how-to-calculate-portfolio-risk-and-return/ (visited on 8th Dec. 2019) (pp. 40, 46).

[6] - and WikiPedia. (2018). Hans-joachim bremermann, [Online]. Available: https://en.wikipedia.org/wiki/Hans-Joachim_Bremermann (visited on 8th Dec. 2019) (p. 17).

[7] ——, (2018). John henry holland, [Online]. Available: https://en.wikipedia.org/wiki/John_Henry_Holland (visited on 8th Dec. 2019) (p. 17).

[8] ——, (2018). Nils aall barricelli, [Online]. Available: https://en.wikipedia.org/wiki/Nils_Aall_Barricelli (visited on 8th Dec. 2019) (p. 17).

[9] - and Wikipedia. (2019). Web development tools, [Online]. Available: https://en.wikipedia.org/wiki/Web_development_tools (visited on 8th Dec. 2019) (p. 32).

[10] Abubakar Yahaya and Research Gate. (2009). Portfolio selection using genetic algorithms, [Online]. Available: https://www.researchgate.net/publication/280069209_Portfolio_Selection_Using_Genetic_Algorithms (visited on 8th Dec. 2019) (pp. 5, 18, 42, 45, 49, 59, 71).

[11] Alan Turing and The Mind Association. (1950). I.—computing machinery and intelligence, [Online]. Available: https://academic.oup.com/mind/article/LIX/236/433/986238 (visited on 8th Dec. 2019) (p. 16).

[12] A. Altman, *How to research stocks*, 2017. [Online]. Available: https://slickbucks.com/articles/how-to-research-stocks/ (pp. 2, 9).

[13] A. A. Ayodele and K. A. Charles, 'Improved constrained portfolio selection model using particle swarm optimization', PhD thesis, Covenant University, 2015 (p. 16).

[14] N. Buhayar, *Weschler rise from grace leads to role advising buffett*, 2012. [Online]. Available: https://www.bloomberg.com/news/articles/2012-10-22/weschler-rise-from-grace-leads-to-role-advising-buffett (pp. 2, 3, 9).

[15] G. Cannon, *How much time do you spend researching a stock?*, 2012. [Online]. Available: https://www.gurufocus.com/news/197569/how-much-time-do-you-spend-researching-a-stock (pp. 2, 9).

[16] Y. Chen and Y. Wang, 'Two-stage fuzzy portfolio selection problem with transaction costs', PhD thesis, Hebei University, 2015 (pp. 21, 73).

[17] J. P. Chiu, 'Named entity recognition with bidirectional lstm-cnns', PhD thesis, University of British Columbia, 2016 (pp. 15, 73).

[18] Clara Calvo, Carlos Ivorra, Vicente Liern and Springer Link. (2014). Fuzzy portfolio selection with non-financial goals: Exploring the efficient frontier, [Online]. Available: https://link.springer.com/article/10.1007/s10479-014-1561-2 (visited on 8th Dec. 2019) (pp. 20, 73).

[19] P. Curran, *Did robots trigger the market plunge?*, 2019. [Online]. Available: https://www.bbc.co.uk/news/business-42959755 (p. 3).

[20] R. Das, J. E. Hanson, J. O. Kephart and G. Tesauro, 'Agent-human interactions in the continuous double auction', PhD thesis, IBM T.J. Watson Research Center, 2001 (p. 10).

[21] David E. Goldberg and Slovak University of Technology in Bratislava. (2016). Genetic algorithms in search, optimization & machine learning, [Online]. Available: http://www2.fiit.stuba.sk/~kvasnicka/Free%5C%20books/Goldberg_Genetic_Algorithms_in_Search.pdf (visited on 8th Dec. 2019) (p. 17).

[22] David M Lane and davidmlane. (2013). Standard deviation and variance (2 of 2), [Online]. Available: http://davidmlane.com/hyperstat/A40397.html (visited on 8th Dec. 2019) (p. 40).

[23] A. Eiben and J. Smith. (2003). Introduction to evolutionary computing, [Online]. Available: https://www.cs.vu.nl/~gusz/ecbook/ecbook-illustrations.html (visited on 6th Aug. 2019) (p. 19).

[24] European Bioinformatics Institute and European Bioinformatics Institute. (2016). Named entity recognition (ner), [Online]. Available: https://www.ebi.ac.uk/training/online/glossary/named-entity-recognition-ner (visited on 8th Dec. 2019) (p. 14).

[25] A. Fernández and S. Gómez, 'Portfolio selection using neural networks', PhD thesis, University of Tarragona, 2005 (p. 16).

[26] Finn Årup Nielsen and WordPress. (2015). Finn årup nielsen's blog, [Online]. Available: https://finnaarupnielsen.wordpress.com/2011/03/16/afinn-a-new-word-list-for-sentiment-analysis/ (visited on 8th Dec. 2019) (p. 35).

[27] J. Folger, *Automated trading systems: The pros and cons*, 2019. [Online]. Available: https://www.investopedia.com/articles/trading/11/automated-trading-systems.asp (p. 11).

[28] V. R. Ghezavati and M. Beigi, 'Solving a bi-objective mathematical model for location-routing problem with time windows in multi-echelon reverse logistics using metaheuristic procedure', *Journal of Industrial Engineering International*, vol. 12, no. 4, pp. 469–483, 2016. DOI: 10.1007/s40092-016-0154-x (p. 55).

[29] M. Gray, *World Wide Web Wanderer*. MIT, 1993 (p. 12).

[30] Harry Markowitz and The Journal Of Finance. (2007). Portfolio selection, [Online]. Available: https://www.math.ust.hk/~maykwok/courses/ma362/07F/markowitz_JF.pdf (visited on 8th Dec. 2019) (pp. 2, 5, 6, 9, 57, 64, 71).

[31] X. Huang, 'Risk curve and fuzzy portfolio selection', PhD thesis, University of Science and Technology Beijing, 2008 (pp. 21, 73).

[32] Java and Oracle. (2019). Class timestamp, [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/sql/Timestamp.html (visited on 8th Dec. 2019) (p. 32).

[33] Java and Oracle Corporation, *Java*, 8 x64 bit, 1991-2019. [Online]. Available: `https://www.java.com` (visited on 19th Apr. 2018) (p. 29).

[34] John Henry Holland and JSTOR. (1992). Genetic algorithms, [Online]. Available: `https://www.jstor.org/stable/24939139?seq=1#page_scan_tab_contents` (visited on 8th Dec. 2019) (p. 17).

[35] Jonathan Hedley. (2016). Jonathan hedley, [Online]. Available: `https://jhy.io/` (visited on 8th Dec. 2019) (p. 29).

[36] JSoup and Jonathan Hedley, *Jsoup*, 1.12.1, 2009-2019. [Online]. Available: `https://jsoup.org/` (visited on 19th Apr. 2018) (p. 29).

[37] Kalyanmoy Deb, Associate Member, IEEE, Amrit Pratap, Sameer Agarwal, and T. Meyarivan and Indian Institute of Technology Kanpur. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii, [Online]. Available: `https://www.iitk.ac.in/kangal/Deb_NSGA-II.pdf` (visited on 8th Dec. 2019) (p. 21).

[38] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami and C. Dyer, 'Neural architectures for named entity recognition', PhD thesis, Carnegie Mellon University, 2016 (pp. 5, 15, 70).

[39] M. Lewis, *Flash boys*. Recorded Books, 2014 (p. 3).

[40] Munasca. (2015). Minimum variance flontier of mpt, [Online]. Available: `https://commons.wikimedia.org/wiki/File:Minimum_variance_flontier_of_MPT.svg` (visited on 30th Apr. 2019) (p. 10).

[41] P. Nakov and T. Zesch, 'Computational semantic analysis of language: Semeval-2014 and beyond', *Language Resources and Evaluation*, vol. 50, no. 1, 2016. DOI: `10.1007/s10579-016-9337-8` (pp. 5, 12, 70).

[42] Paul Gil and Lifewire. (2018). What is twitter & how does it work?, [Online]. Available: `https://www.lifewire.com/what-exactly-is-twitter-2483331` (visited on 8th Dec. 2019) (p. 12).

[43] Philip C Jackson Jr and New York University Computer Science. (2017). Introduction to artificial intelligence, [Online]. Available: `https://cs.nyu.edu/cs/faculty/grishman/muc6.html` (visited on 8th Dec. 2019) (p. 14).

[44] T. Plaehn, *How much does it cost to start a stock portfolio?*, 2019. [Online]. Available: `https://finance.zacks.com/much-cost-start-stock-portfolio-11341.html` (p. 8).

[45] Python and GitHub. (2010). Nltk's list of english stopwords, [Online]. Available: https://gist.github.com/sebleier/554280 (visited on 8th Dec. 2019) (p. 33).

[46] Python and The Share Centre. (2018). Waht is the ftse 100?, [Online]. Available: https://www.share.com/a-guide-to-investing/before-you-start/what-is-the-ftse-100 (visited on 8th Dec. 2019) (p. 34).

[47] Qingfu Zhang, Hui Li and IEEE. (2007). Moea/d: A multiobjective evolutionary algorithm based on decomposition, [Online]. Available: https://ieeexplore.ieee.org/document/4358754 (visited on 8th Dec. 2019) (pp. 21, 73).

[48] R0oland. (2013). Onepointcrossover, [Online]. Available: https://commons.wikimedia.org/wiki/File:OnePointCrossover.svg (visited on 6th Aug. 2019) (p. 19).

[49] ——, (2013). Twopointcrossover, [Online]. Available: https://commons.wikimedia.org/wiki/File:TwoPointCrossover.svg (visited on 6th Aug. 2019) (p. 19).

[50] Rubén Saborido, Ana B. Ruiz, José D. Bermúdez, Enriqueta Vercher, Mariano Luque and Science Direct. (2016). Evolutionary multi-objective optimization algorithms for fuzzy portfolio selection, [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494615007164 (visited on 8th Dec. 2019) (pp. 21, 22, 73).

[51] S. J. Russell and P. Norvig, *Artificial intelligence*, 2nd ed. Prentice Hall, pp. 116–117 (p. 17).

[52] Selenium and Jason Huggins, *Selenium*, 1.12.1, 2004-2019. [Online]. Available: https://www.seleniumhq.org/ (visited on 19th Apr. 2018) (pp. 5, 29, 71).

[53] Slimane Sefiane, M. Benbouziane and Research Gate. (2012). Portfolio selection using genetic algorithms, [Online]. Available: https://www.researchgate.net/publication/232041512_Portfolio_Selection_Using_Genetic_Algorithm (visited on 8th Dec. 2019) (pp. 5, 18, 19, 42, 45, 49, 59, 62, 71).

[54] C. Steiner, *Wall street's speed war*, 2010. [Online]. Available: https://www.forbes.com/forbes/2010/0927/outfront-netscape-jim-barksdale-daniel-spivey-wall-street-speed-war.html#42458131741a (p. 11).

[55] WILL KENTON and Investopedia. (2019). Capital asset pricing model(capm), [Online]. Available: https://www.investopedia.com/terms/c/capm.asp (visited on 8th Dec. 2019) (p. 40).

[56] William F. Sharpe and Wikipedia. (2019). Sharpe ratio, [Online]. Available: `https://en.wikipedia.org/wiki/Sharpe_ratio` (visited on 8th Dec. 2019) (p. 40).

Appendix

A.1 P.E.S.T.L.E Analysis

The P.E.S.T.L.E analysis method is a tool for evaluating the factors of the outside world, and how/if it will affect a new product being launched or researched. This chapter talks about the important benefits and issues revolving around this project. As well as discussing how the launch of this project into the world could affect peoples lives or well-being.

A.2 Political Factors

This product uses a U.K. based news source for it's data gathering. Therefore the biggest political factor that can affect this project, is the ongoing Brexit discussion. Any change to a companies stock price will likely affect the results of this algorithm. However, the algorithm would still find the best portfolio given historical data. Nonetheless, the return of the stock will still be lowered.

On the other hand, if for example all U.K. businesses would suddenly drop drastically in stock price, the portfolio's that are featuring U.K. companies would not factor in the change in stock price immediately. Later on however, the change in stock price would be factored in and the risk of the company would greatly increase. Thus increasing the likeliness that more efficient safe companies would be preferred by the genetic algorithm, and place a low percentage on those U.K. companies.

Another point about Brexit is that the uncertainty around job security could affect a companies stock price dramatically, as well as affect the public's reputation on said company/companies. Finally, with regards to Brexit, the future of trade and funding agreements with the EU is still very ambiguous, any change in the country's trade agreement with the EU will affect a companies stock price also.

If this project was to be sold freely to the general public, it could result in high political pressure from the government and a possible call for regulations to the stock market. It would mean that everyone in the U.K. could place stocks **ONLY** on portfolios that would have return the most profit per Risk/Return. This could eventually result in a new mass haul of investors, and contributing massively to the U.K. economy.

The final political factor is the political bias that the chosen news source is affiliated with could affect the truthfulness of the articles and therefore affect whether that company is chosen to be part of the portfolio. For example, if a CEO of a company is openly left wing, a right wing paper may not shine a light on that company, and may even react unfavourably towards that article and write untruthful stories.

A.3 Economic Factors

One particular economic factor that will affect this project is, the hiding of historical data. If a company hides their historical data for whatever reason, there is no way that that company can be placed in a portfolio by this program. This program is after all inherently reliable on companies releasing their historical data. This does not seem to be a problem however as every company that goes public are required to release their historical data.

Another point is that this program not only requires the release of historical data, it also requires the companies to be honest about their historical data. There have been a number of companies in the past who have 'Cooked the books', and released data that have resulted in a higher stock price than it should have been. This is illegal and companies found doing this will be charged, however it is worth mentioning.

There is a problem in that companies with less news sources or many negative news sources will never have a portfolio created around them as they wont be chosen by the algorithm, this could cause those companies to go bust due to lack of investment. This could be countered by the argument of 'Well they have to contribute positively, or they deserve to go bust'. Regardless, it is still worth noting this could affect businesses massively.

This program could also result in a loss of jobs for financial advisers and stock brokers. Causing their jobs to be redundant as investors will only need this program to create an efficient portfolio and not their expert advice.

A.4 Sociological Factors

The main sociological factor for this project is the lack of knowledge and lack of education of the stock market as a whole. If people do not know about the stock market or care to learn about investments, they will never have any need in this projects work.

Having said that, due to the simpleness of this project to run and output the data, people with no idea of the stock market or how it works could still place investments based on the portfolio that is returned to them, in order to 'get rich'. This could work beneficially, by creating wealth for people that wouldn't typically use the stock market, and it could also work disadvantageously as people with no education of the stock market would not be able to see the signs coming that an investment even though it looks good on paper would not be good in the long run, investments are inherently risks after all.

Their may be a religious sentiment involved as some Religious groups may see investments as a form of 'gambling' due to its risk/reward nature, and therefore those religious groups may frown upon investments and sequentially this project.

One final sociological factor is that the amount of people that place stocks is based entirely on the amount of people that have the money to place stocks. If the economy tanks due to some unforeseen circumstances and results in people not having enough money to invest, they wont. Ultimately deriving a lack of need for a project like this.

A.5 Technological Factors

A drastic change to any of the websites used to Web Scrape would result in the need to redesign the web scraping program. This would not take long (maximum 3 days). On the topic of Web Scraping, there is a chance that a new law could be passed that would deny the use of Web Scraping on certain websites, if this is the case the program can be monitored to take in RSS feeds, a Java class has already been developed to do this (currently not in use).

The need for historical data is imperative to this program, however new companies that haven't been around long enough (<1 year) will not have the required historical data, changes could be made in this case to only take in the maximum amount of historical data that a company has. This could then be averaged out in the same format as it currently is.

Finally, the malicious change of the two websites data (hacking) could manipulate the outcome of this program and depending on the timeliness of a fix could result in portfolios that should be voided.

A.6 Legal Factors

A change to any regulation/legislation/law regarding Web Scraping would affect this project, however a change to investments would also drastically affect this project. For example, if a law passed saying that any one person is only allowed to place maximum 3 shares into a company at any one time, this would of course affect the program as it outputs 5. However this is a very far stretch as no law like this has ever existed.

It is also worth observing that, a company may sue the website that supplies our historical data, for falsifying that data. This is very unlikely as well as the website takes data from data vendors, which are typically reliable and trustworthy.

A.7 Environmental Factors

There are few environmental factors with software in general, however the 'Judgement day' scenario of a total power outage country wide, would of course render this program useless as it requires a computer to run.

A side note would also be that an environmental disaster like a flooding could negatively affect/damage a companies headquarters and destroy inventory or the building itself.

This would of course mean that the company would lose money and result in a drop in stock price.

A.8 Code

A.8.1 Normalise

```
public static double[] normalise(double[] portfolio) {
    double sum = 0.0;
    double diff = 0.0;
    double[] perc = new double[5];
    double percentage = 0.0;
    for (int i = 0; i < portfolio.length; i++) {
        if(portfolio[i] <= 0.01) {
            portfolio[i] += 0.02;
        }
        sum += portfolio[i];
    }
    for (int i = 0; i < portfolio.length; i++) {
        percentage = portfolio[i] / sum;
        perc[i] = percentage;
    }

    if(sum > 1.0) {
        diff = sum - 1;
        each = diff / portfolio.length;
        for (int i = 0; i < portfolio.length; i++) {
            double newPerc = diff * perc[i];
            portfolio[i] = portfolio[i] - newPerc;
        }
    }
    if(sum < 1.0) {
        diff = 1 - sum;
        each = diff / portfolio.length;
        for (int i = 0; i < portfolio.length; i++) {
            double newPerc = diff * perc[i];
24              portfolio[i] = portfolio[i] + newPerc;
            }
        }
25      return portfolio;
    }
```

A.8.2 Selection

```
public static double[] selection(double[] portfolio1, double[]
    portfolio2) {
        double portfolio1Fitness = calcFitnessOfPortfolio(portfolio1,
            means[0], means[1], means[2], means[3],
                means[4]);
        double portfolio2Fitness = calcFitnessOfPortfolio(portfolio2,
            means[0], means[1], means[2], means[3],
                means[4]);
        double[] bestPortfolio;
        if (portfolio1Fitness < portfolio2Fitness) {
            bestPortfolio = portfolio1;
        } else {
            bestPortfolio = portfolio2;
        }
        return bestPortfolio;
}
```

A.8.3 One-Point Crossover

```
public static double[] onePointCrossover(double[] portfolio1, double[]
    portfolio2) {
        Random rn = new Random();

        //Select a random crossover point
        int crossOverPoint = rn.nextInt(portfolio1.length) + 1;

        //Swap values among parents
        for (int i = 0; i < crossOverPoint; i++) {
            double temp = portfolio1[i];
            portfolio1[i] = portfolio2[i];
            portfolio2[i] = temp;
        }

        return portfolio1;
    }
```

A.8.4 Two-Point Crossover

```
public static double[] twoPointCrossover(double[] portfolio1, double[]
    portfolio2) {
        Random rn = new Random();

        //Select a random crossover point
        int crossOverEnd = rn.nextInt(portfolio1.length - 1);
        int crossOverStart = rn.nextInt(portfolio1.length -
            crossOverEnd) + 1;
        //Swap values among parents
        for (int i = 0; i < crossOverStart; i++) {
            double temp = portfolio1[i];
            portfolio1[i] = portfolio2[i];
            portfolio2[i] = temp;
        }
        return portfolio1;
    }
```

A.8.5 Mutation

```
public static double[] mutation(double[] portfolio1) {
        Random rn = new Random();
        double[] newMutatedPortfolio = portfolio1;
        //Select a random crossover point
        int mutationPoint = rn.nextInt(portfolio1.length);
        double randomMutWeight = rn.nextInt(1);
        newMutatedPortfolio[mutationPoint] = randomMutWeight;
        return newMutatedPortfolio;
    }
```

A.8.6 Fitness

```
public static double calcFitnessOfPortfolio(double[] portfolio,
double mean1, double mean2, double mean3, double mean4, double
mean5) {
        double weight1 = portfolio[0]; double weight2 = portfolio[1];
        double weight3 = portfolio[2]; double weight4 = portfolio[3];
        double weight5 = portfolio[4];
        double port[] = {weight1, weight2, weight3, weight4, weight5};
        double retrn1 = calcReturnOfPortfolio(port);
        double lRisk = lowestRisk;
        double fitness = 1;
        double first = (1 - penalty) * lRisk;
        double second = penalty * (((weight1 * mean1) - retrn1)
              + ((weight2 * mean2) - retrn1) + ((weight3 * mean3) -
                  retrn1)
              + ((weight4 * mean4) - retrn1) + ((weight5 * mean5) -
                  retrn1));
        double third = first + second;
        fitness = third;
        if (fitness < lowestPortfolioFitness) {
            lowestPortfolioFitness = fitness;
            lowestPortfolioWeights[0] = weight1;
            lowestPortfolioWeights[1] = weight2;
            lowestPortfolioWeights[2] = weight3;
            lowestPortfolioWeights[3] = weight4;
            lowestPortfolioWeights[4] = weight5;
        }
        return fitness; }
```