Automated Stock Market Trading Using Machine Learning

Thesis · April 2018

DOI: 10.13140/RG.2.2.19887.59045

1 author:

Luke Rose
University of Nottingham
1 PUBLICATION **0** CITATIONS

SEE PROFILE

Automated Stock Market Trading Using Machine Learning

Luke K. Rose
Student ID: 4250118
Computer Science with Artificial Intelligence
psylr5@nottingham.ac.uk

Supervised by:
Thomas Gärtner
Professor of Data Science
School of Computer Science
Thomas.Gaertner@nottingham.ac.uk

April 21, 2018

Abstract

Stock market prediction regards the forecasting of the price of any given stock within a desired time-frame and has been a heavily researched topic over past years due to the difficulty of predicting time-series that are considered to be *random walks*. Whilst there are those that use traditional Technical Analysis methods such as the calculation and consideration of trends, more recently the problem has attracted the attention of Machine Learning and Artificial Intelligence approaches.

This project explores and compares the current Machine Learning approaches involved in predicting the direction and prices of selected stocks for a given time range, considering short, medium and long-term investments. Using these models alongside Natural Language Processing of financial news to predict sudden, extreme fluctuations and Portfolio Optimisation to balance risk and expected return prior to trading, an automated trading agent is designed, implemented and evaluated against the index performance that the stocks are traded upon (NASDAQ100).

Acknowledgements

Many thanks to my supervisor Thomas Gärtner, who gave me many suggestions for features and improvements to add to the application. Thanks to my mother Joanne, sister Jemma and grandmother Heather for supporting me throughout University and this project.

In memory of my grandfather

Patrick Buckle
1952-2016

who gave me my first book on Java programming and inspired me to reach my goals.

Contents

List of Figures

List of Tables

Introduction

Producing accurate methods for predicting stock market price changes has been a goal of both the financial and technological communities for many years. Such research includes utilising various Artificial Intelligence and Machine Learning methods to predict both sudden and long-term changes in a given share's price, minimising loss and maximising profit. Existing software attempting to deliver such results rely mostly on simple assertions on raw statistical data and trend indicators such as 'Sell shares when 50-day moving average goes below 200-day moving-average'. This approach is known in the industry as 'Algotrading' or 'Algorithmic Trading' and often requires expert knowledge of investment bankers or experienced traders to configure and maintain. Another common method is time-series analysis such as Auto-Regressive Integrated Moving Average (**ARIMA**) to allow for forecasting, being particularly useful where data is non-stationary, as it is with financial data.

Many research papers have been published concerning the use of models such as Artificial Neural Networks[30], Support Vector Machines[17] and Linear Regression[13] to provide more accurate predictions based on mining of historical data and in some cases to be used for making trading decisions[26]. One issue encountered when considering stock market data is that it is of a time-series format, requiring predictions based on prior history within a suitable time frame (weekly, monthly, yearly) rather than a single data point. Financial time-series have low signal-to-noise ratios and fall victim to noise-based over-fitting and require some preprocessing before they are used. In this domain, the *signal* would be composed of trend information that can indicate the current *sentiment* of a market, whereas *noise* is caused by programmatic trading, irrational/uninformed investments by traders (for example due to misinterpretation of news) or excessively frequent buying and selling without considering any influential information. A core challenge facing stock market prediction, is the unpredictability of extreme-case events, such as when a price drops dramatically within a very short period. Most stock prediction models utilise per-day historical data over the last n years for long-term prediction but often neglect the use of alternate information for predicting more frequent changes.

Use of prediction information for making trading decisions requires consideration by a human. This therefore introduces restriction and uncertainty in terms of stock choice and purchase/sale quantity, which can affect the performance of investments, potentially reducing expected returns [28]. Allowing control of the trading process by an autonomous agent would bypass emotion-based restrictions and could yield greater results in a reduced time-frame. Given a high-frequency stock market data feed, such freedom would allow exploitation of the fast-paced fluctuations in real-time day-trading as well as long-term portfolio improvement. It can also be beneficial to optimise the portfolio prior to automated trading, by selecting a good balance of *risky* and *safe* stocks, whilst diversifying amongst various industries/sectors and allocating a portion of overall investments to each stock accordingly.

1.1 Motivation

Stock Market time-series behave differently to other applications; whereas scenarios such as using a retail store's historic customer volume and frequency data to predict busiest visiting times has time-series that can contain repetitive surges (for example, yearly increased footfall at Christmas) and are generally similar on a day-to-day basis, Stock Markets have been considered to act as a *Random Walk*, showing very little correlation between days and can have sudden, large fluctuations with very few (if any) monthly or yearly cycles. Nevertheless, similar to other time-series, Stock Market data can have trends in the short, medium and long terms, despite noisy data, allowing the possibility of prediction.

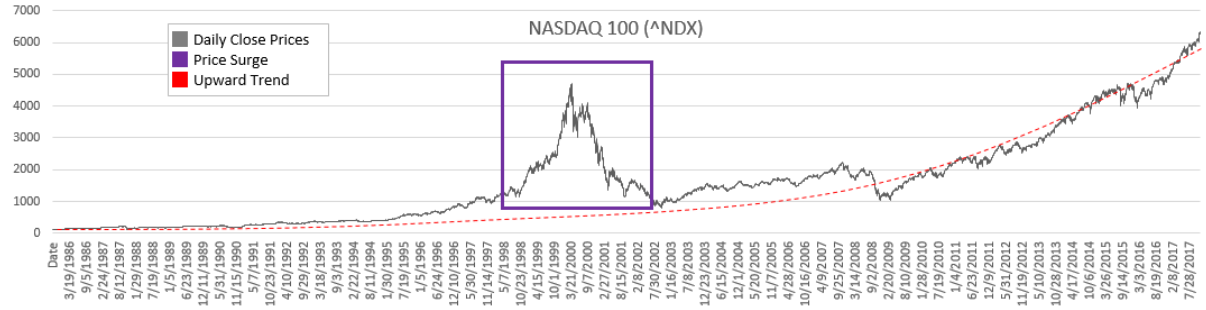

Figure 1.1: NASDAQ 100 Closing Price History with Price Surge and Upward Trend

The ability to accurately predict the stock market, by minimising misclassification of price direction (increase or decrease) or minimising the difference between predicted and expected price values, would provide a significant advantage over other traders, with the potential to greatly increase returns on investments. Coupling such a framework with Portfolio Optimisation and an intelligent trading agent would reduce risk and allow for long-term investment security. The application of this system is not limited to stocks and could be applied to commodities such as energy, metals, livestock and agriculture, Foreign Exchange (*Forex*) and most recently, *Cryptocurrencies* such as *Bitcoin* or *Ethereum*.

1.2 Aims and Contributions

The aim of this project is to analyse and compare the latest Machine Learning stock market prediction methods and to develop a new variant of Artificially Intelligent Automated Trading System, improving upon existing applications, that utilises a prediction model, Portfolio Optimisation and Natural Language Processing to provide an 'All-in-One' software suite to automate trading without the need for human intervention. Using an appropriate brokerage API, this software could be used to trade actual stocks in future work, however the scope of this project focuses on simulating market performance offline (with actual stock market data) to allow for user practise. Most research concerning stock market prediction refers to the empiricle analysis of a given set of Machine Learning models or Technical Analyses but lacks consumer-level applications; within this project, such research is investigated and used to implement a full application, incorporating multiple methods of improving investment returns. The final deliverables include a practical and customisable consumer-level software package with uses outside the configuration described in this project, allowing non-expert users a greater opportunity for investing in stocks.

1.3 Key issues

Prediction of the stock market has been noted in previous research to be limited by the *Efficient Market Hypothesis* or *Random Walk Theory*, stating that it is impossible to predict the market because all available information that could affect the stock has already been reflected in the price. The hypothesis does however have multiple variants, each easing the constraints slightly[9]:

- **Weak Form**: The current price only contains information based on past prices and therefore using price data alone is not enough to predict the market. Non-financial information such as public opinion or publications can therefore be used to predict the market.
- **Semi-Strong Form**: The current price incorporates *all public information* such as price, company performance (net profit, company growth etc.), competitor company performance, political and economic climate, news articles, research papers etc. This therefore means that only *private* or *insider* information can be used to predict the market.
- **Strong Form**: The current price considers *all information* and therefore can not be predicted via any means of analysis.

Financial time-series data suffers from a low signal-to-noise ratio, where the *signal* comprises of useful data and *noise* is the result of outliers and irrelevant data, which therefore makes the data difficult to predict. Such an issue can usually be remedied through the use of noise filtration techniques or consideration of an averaged representation of the data.

Selection of features can greatly affect the effectiveness of a model and stock market data has a wide array of options in this respect. Common features include daily price data (Open, High, Low, Close), trading volume, Moving Averages (Simple and Exponential), Volatility, Momentum - addition of more variables can be beneficial but can also complicate the feature space, leading to over-fitting and excessive use of memory and computation time; as such, suitable features must be carefully considered.

1.4 Research Question

In this project we will investigate the question:

Using current research and considering historic price/technical indicator based prediction, sentimental information and an initial optimal portfolio, can an effective automated stock-trading application be developed that can significantly improve returns over traditional trading?

Related Work

There are a wide array of software packages available for use in Automated Trading, both Artificially Intelligent and Algotrading based, however many of these systems are based solely on a single or rigid set of prediction methods. NeuroShell Trader[1] is a system that allows the forecasting of stocks using a custom selection of technical indicators, multiple methods of creating an initial portfolio and sending of commands to a brokerage to execute trades, although its prediction system is based only on Neural Networks, when research has shown that there are many other models that can potentially perform better. The system is also only based on raw data and technical indicators to create rules, with no inclusion of text-based information. Portfolio Optimisation services such as WealthSimple[2] allow for rebalancing the portfolio when investments no longer keep within their allocated capital. Whilst good results can come from such as system, it fails to utilise any form of prediction mechanism to forecast the performance of stocks within their portfolios, potentially missing out on additional profits.

These existing systems can be improved upon by joining aspects from multiple systems into one, considering data-based prediction, Portfolio Optimisation and text-based information mining. In this section we will consider the existing methods for performing these tasks.

2.1 Stock Prediction

Many attempts have been made at predicting the **price** and/or **direction** of a stock (Increasing, No Change, Decreasing) at a given time, using many different models. These models can range from simple linear regression to more recent novel attempts at hybridisation of multiple models such as Neural Networks and Support Vector Machines. This project considers the applications of standard (non-hybrid) models alongside alternative trading and prediction methods, to increase profitability when investing in stocks. Standard models have been researched extensively in the past few years, with promising results, for example 76.5% F-Score when selecting and classifying which stocks will increase by 10% within a year, using Random Forests[21]. While this is far from a more desired 90%+ accuracy, it certainly shows the possibility of a performance greater than the *coin-flip* probability that is common with binary classification problems.

Ensemble methods have also shown great potential in improving upon these *base learners*, combining multiple weak predictors to create a strong predictor. *Bagging* (Bootstrap Aggregation), *Stacking* and *Random Subspace* methods have been used with Support Vector Machine, Decision Tree and Artificial Neural Network models, with results suggesting that base-learners can be especially improved using Bagging[8].

In this chapter, we will consider the various methods that have been used to predict stock prices and determine their suitability towards the project, considering the results they have produced and their prominence in previous work.

[1] http://try.neuroshell.com/index/
[2] https://www.wealthsimple.com/en-gb/

Linear/Polynomial/Logistic Regression is regarded to be one of the simplest methods of Machine Learning, modelling the relationship between an output variable and one or more (*Multiple Regression*) input variables using an unknown function. Multiple Regression is usually required in stock market prediction using Regression (such as in Gustaf Forslund and David Åkesson's attempt[13]) due to the many variable inputs that can affect a price's value. Whilst Regression techniques have been used extensively in the algorithmic trading community[3] due to its ease of understanding for newcomers to the field, it is rarely used by itself for predicting stock prices in research. Instead, Linear/Polynomial Regression is used for technical analysis, being useful in identifying price trends in the long-term but often rarely outperforms the prediction accuracy of other models in the short-term.

Artificial Neural Networks are non-linear models that simulate the processes between neurons and synapses within the brain in order to model complex problems. They have been popular for use in predicting the Stock Market due to their black-box nature, allowing users to try an arbitrary amount of features to generate usable results, without having to know how it works. There are many variations of ANN that can be applied to time-series forecasting, such as Multilayer Perceptrons (**MLP**)[30] - the simplest variant or Convolutional Neural Networks (**CNN**)[26] which encode price information into image formats to allow for convolutions to be performed. Memory-based networks such as Recurrent Neural Networks (**RNN**) and Long Short-Term Memory (**LSTM**)[7] networks allow for predictions to be *remembered* and *forgotten*; this can be useful when predicting extreme changes in a short space of time via the remembering of previous examples. Compared to other models, RNN and LSTM are suited towards sequential data such as stock prices, whereas other models must take a non-sequential, sample-independent interpretation of the data, considering only a single input and output sample at a time.

Support Vector Machines (SVM) are a popular alternative to other models due to their ability to find global maxima and their inherent over-fitting control using regularisation and minimisation of generalisation error. Separating points by finding a maximum-margin between *Support Vectors* and a *hyperplane* allows them to have greater reliability over methods such as linear regression, which find a *suitable* but not *optimal* separation . When carefully considering the selection of parameters used for input-vectors, SVMs have been shown to outperform models such as back-propagation Artificial Neural Networks (**ANN**)[19] in terms of accuracy and minimisation of errors. Due to the transformation of data into a linearly-separable space, these input-vectors can suffer from *the curse of dimensionality*, requiring a lot of memory and computation time to train[29]. Variations of SVM can be used for regression (*Support Vector Regressors* - ***SVR***) and classification (*Support Vector Classifiers* - ***SVC***) problems, providing a framework for predicting both price direction and value.

Random Forests are ensemble models that utilise a set of *weak learners* (in this case *trees*) and apply them to a separate subspace of the input matrix, taking the majority or mean output of all the trees to create a single *strong* learner. Based on previous research, this method is very popular, reaching accuracies of up to 94.533%[16] for predicting the price direction of a stock at 3 months. Random Forests in general provide very accurate results and are very efficient during runtime, compared to single trees which suffer from high variance and bias. They are more interpretable than models such as Neural Networks, with important *deciding* features being observable based on the *splitting attributes* at each node.

[3] https://www.quantinsti.com/blog/machine-learning-trading-predict-stock-prices-regression/

Natural Language Processing Sentiment Analysis is the process of determining the positive or negative connotation of a mass of text. This can be done through various methods, including the use of sentiment-labelled dictionaries, matching document metadata to the text (for example reviews can be labelled with ratings out of 5) and labelling text in a semi-supervised manner by mining additional metrics in which to rate the text. This project will investigate the suitability of the third example, utilising price increase and decrease data on any given day, to label news articles published on the same trading date. Previous applications of Sentiment Analysis to stock prediction have used multiple sources, including Social Media information such as Twitter posts, which has the advantage of having a greater range of opinions (i.e. no media bias) but the disadvantage of comprising of slang, poor grammar and irregular characters such as *Emojis*. It should be considered that influential public information does not have instant effects on stock performance and has been found to have delayed impact of three to four days[22], which may need to be accounted for within this project when predicting based on Natural Language Processing techniques.

Crowdsourcing the combined knowledge and opinions of Internet communities can be useful in predicting stock direction and velocity, by allowing users access to online voting systems to express their own predictions. This information can be combined into a feature vector alongside prices and technical indicators with appropriate weighting of objective versus subjective data[23] or used individually. Similar insights can be obtained through collection of *expert analyst* opinions however such information can be enough in itself to cause price shifts when influencing decisions made by other entities[10]. This method is not ideal as the variance between subjectivity of opinions of different users is too great and availability of such data is unreliable and inconsistent. Models must also ensure that the prediction of a user alone cannot outperform the performance of the final system as a whole, including the additional machine learning[23]. The focus of this project is the implementation of a system that acts upon the most influential publicly available data such as news and therefore this method will not be used in the final software.

2.2 Portfolio Optimisation

Portfolio Optimisation concerns the selection of stock market indices or stocks from a large feature space, such that the selected indices or stocks provide the least investment risk. This provides a good base-point for automatically trading stocks, by eliminating particularly unpredictable or volatile stocks and giving a more reliable outcome. The majority of the methods of Portfolio Optimisation are based on Mean-Variance Portfolio Optimisation Theory, proposed by Harry Markowitz[18] and have since improved upon his work, allowing for various constraints, such as total amount of shares held, to be considered. Modern attempts at solving this problem include the use of Artificial Intelligence and metaheuristics such as Genetic Algorithms and Simulated Annealing. Although each method has its strengths and weaknesses[15], they have shown good results, boasting returns of over 26%[4]; considering this information, the project will focus on the use of Artificial Intelligence methods when determining optimal portfolio configurations.

In this section, we explored many possible methods of predicting the stock market, including well-founded attempts, in addition to novel algorithms in their early stages. Due to the lack of peer-reviewed papers or further results/evidence to support these novel methods, this project will only consider existing methods with strong foundations and consistently promising results, including SVM, ANN and Random Forests. We also considered modern methods of Portfolio Optimisation in to improve upon Markowitz' original work, for a more optimal solution.

Design

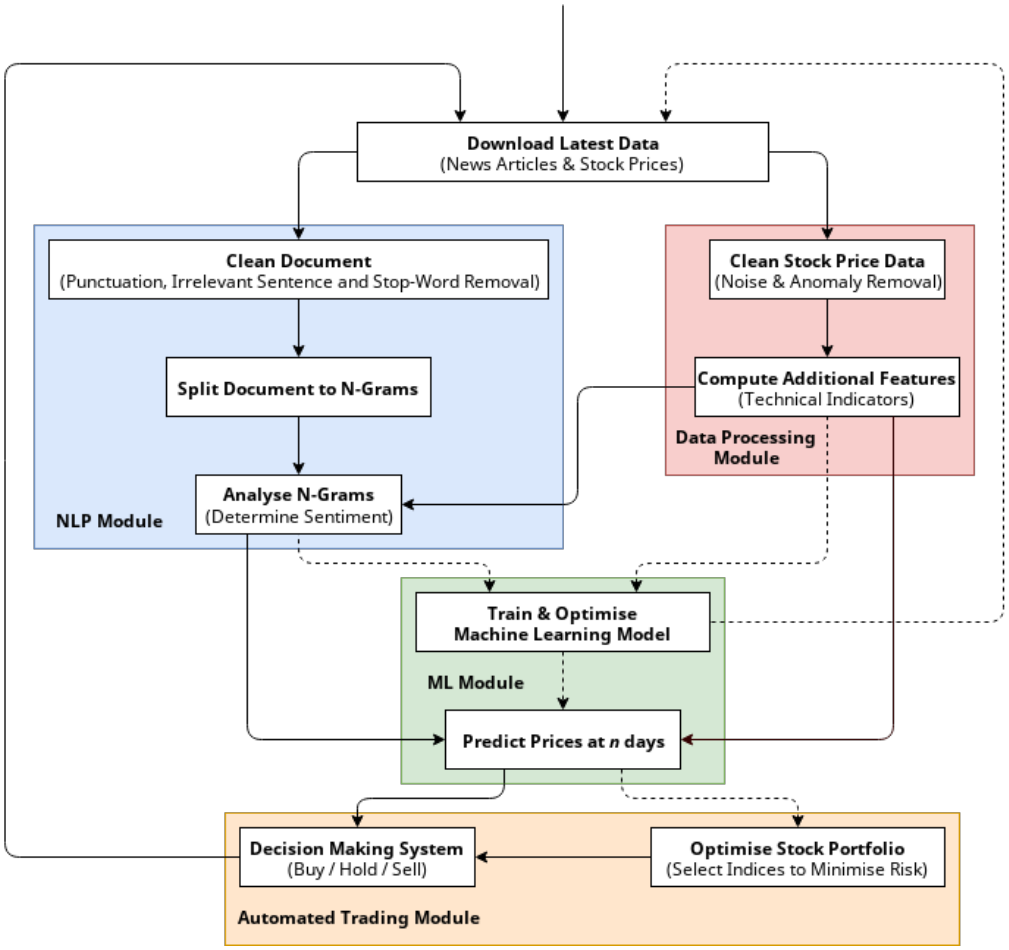

Figure 3.1: Process flow of Automated Trading System

In this section, we will explore the various modules that are required in the system. This includes the consideration of Portfolio Optimisation, Machine Learning (Classification) and Natural Language Processing algorithms. Machine Learning model suitability and optimal hyper-parameter selection is determined based on performance with processed data (accuracy, speed, space requirements), using *Auto-WEKA*[27][1], which performs thousands of experiments on given input and output vectors, to determine suitable models and hyper-parameter configurations [2]. Extensive feature testing is then performed on these models using the standard WEKA[14] environment, which also provides various auxiliary tools such as data conversion functions from Comma Separated Value (**CSV**) to Attribute-Relation File Format (**ARFF**).

[1] Automatic model and hyper-parameter selection using the Waikato Environment for Knowledge Analysis.
[2] Thanks to the University of Nottingham High Performance Cluster (**HPC**) for processing these experiments.

3.1 Data Sourcing

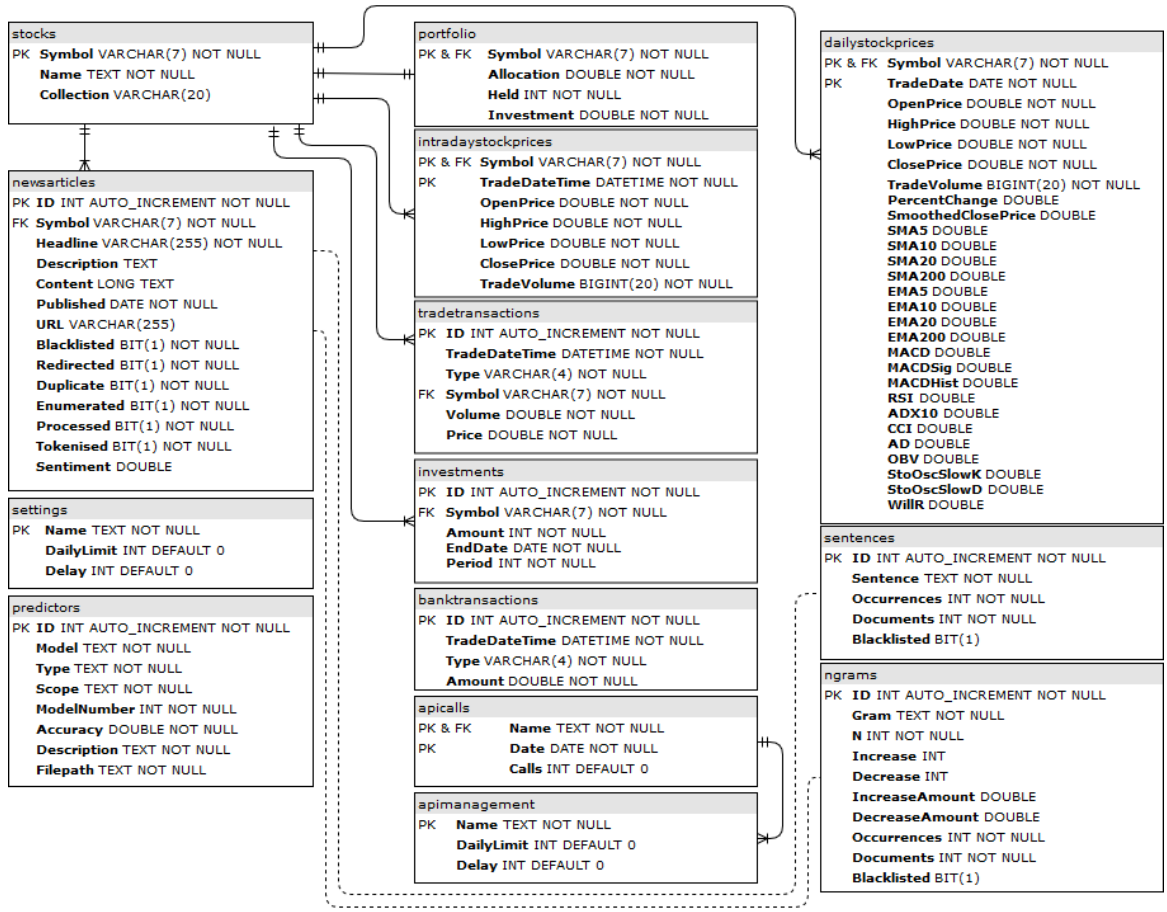

Figure 3.2: Entity Relationship Diagram of Automated Trading System database

Historic data dating back to the Initial Public Offering (**IPO**) of each stock is sourced from *Yahoo! Finance*, which provides per-day **O**pen, **H**igh, **L**ow and **C**lose prices and trade **V**olume (**OHLCV**) data. This data has been criticised within the financial industry for not being as accurate as paid-for data sources such as *Quandl*, however it will be used in this research due to its free availability.

Real-Time data including 1-minute interval intraday OHLCV data and the latest per-day data not covered in the historic data set, is downloaded regularly via the *AlphaVantage* Application Programming Interface (**API**).

News article corpus containing headlines, summaries and URLs of news publications from various sources is downloaded via the *Intrinio* API, covering both historic and recent information. URLs available within this information are then crawled for the main content of each article.

3.1.1 Stock Tickers

This project will consider the stocks listed under the NASDAQ-100 collection, covering the top 100 actively traded companies on the NASDAQ Exchange, including those from the healthcare, services, technology, financial and consumer goods sectors. A full list of companies from this index is available in appendix B.

3.2 Data Pre-Processing

3.2.1 Data Cleaning

To prevent over-fitting to noise or outliers, data is cleaned before use. This includes removing *useless*, *missing* or *null* values from the dataset. The series is then smoothed exponentially, to prevent the impact of noise and make any clear trends easily identifiable. Based on research by Dzikevičius[11], the formula used for exponential smoothing in the application is as follows:

$$x'_0 = x_0 \tag{3.1}$$

$$x'_i = \alpha * x_i + (1 - \alpha) * x'_{i-1} \ \forall i > 0 \tag{3.2}$$

where **x** is the set of close prices for a stock, **x'** is the smoothed price set, i is the day under consideration and the α parameter determines the magnitude of smoothing, taking values between 0 and 1.

3.2.2 Technical Indicator Extraction

Using exponentially smoothed stock prices, some of the most commonly used technical indicators are to be extracted, for use within feature vectors used for training and testing models.

Simple Moving Average (SMA)

$\frac{\sum_{i=d-n}^{d} x_i}{n}$

x = Closing Prices
i = Current Day
n = Number of Days
d = Day to calculate

Exponential Moving Average (EMA)

$EMA_0 = SMA$
$EMA_i = (x_{i-1} - EMA_{i-1}) * \frac{2}{n+1} + EMA_{i-1}$

x = Closing Prices
i = Current Day
n = Number of Days

Moving Average Convergence Divergence (MACD)

$MACD_d = EMA12_d - EMA26_d$

EMA12 = 12 Day EMA
EMA26 = 26 Day EMA
d = Day to calculate

Stochastic Oscillator (StoOsc)

$StoOsc = 100 * \frac{x - Low_{14}}{High_{14} - Low_{14}}$

x = Current closing price
Low_{14} = Lowest price of last 14 days
$High_{14}$ = Highest price of last 14 days

Relative Strength Index (RSI)

$RSI = 100 - \frac{100}{1+RS}$
$RS = \frac{AVG(Gains_{i-14...i})}{AVG(Losses_{i-14...i})}$
Gains = Price increase sum over last 14 days
Losses = Price decrease sum over last 14 days

Williams % R

$\%R = \frac{h_n - c_i}{h_n - l_n} * -100$
h_n = Highest price over last n days
l_n = Lowest price over last n days
c_i = Current Close Price

Accumulation Distribution Line (AD)

$MFM_i = \frac{(c_i - l_i) - (h_i - c_i)}{h_i - l_i}$
$MFV_i = MFM_i * v_i$
$AD_0 = MFV_0$
$AD_i = AD_{i-1} + MFV_i$

MFM = *Money Flow Multiplier*
MFV = *Money Flow Volume*
i = Current day
Prices: **c** = Close, **h** = High, **l** = Low
v = Trade Volumes

On-Balance Volume (OBV)

$OBV_0 = 0$

$$OBV_i = \begin{cases} OBV_{i-1} + v_i, & \text{If } c_i > c_{i-1} \\ OBV_{i-1} - v_i, & \text{If } c_i < c_{i-1} \\ OBV_{i-1}, & \text{If } c_i = c_{i-1} \end{cases}$$

i = Current Day
c = Close Prices
v = Trade Volumes

Commodity Channel Index (CCI)

$CCI = \frac{t_i - SMA_i}{0.015 * MD(t_i)}$
$t_i = \frac{(h_i + l_i + c_i)}{3}$
$MD = \frac{1}{n} \sum_{i=1}^{n} |t_i - \bar{\mathbf{t}}|$

t_i = Typical Price
h_i = High Price on day i
l_i = Low Price on day i
c_i = Close Price on day i
MD = Mean Absolute Deviation

Average Direction Index (ADX)

$Rise = h_i - h_{i-1}$
$Fall = l_{i-1} - l_i$

$$+DM = \begin{cases} Rise, & \text{If Rise} > \text{Fall and Rise} > 0 \\ 0, & \text{else} \end{cases}$$

$$-DM = \begin{cases} Fall, & \text{If Fall} > \text{Rise and Fall} > 0 \\ 0, & \text{else} \end{cases}$$

$+DI = 100 * EMA_{14}(+DM)/ATR$
$-DI = 100 * EMA_{14}(-DM)/ATR$
$ADX = 100 * EMA_{14} \frac{(|+DI - -DI|)}{(+DI + -DI)}$

$EMA_{14}(x)$ = 14 day EMA of x
ATR = Average True Range ($\frac{1}{n} * \sum_1^n TR$)
TR = True Range ($max((h_i - l_i, |h_i - c_{i-1}|, |l_i - c_{i-1}|)$)
i = Current day
h = High Prices, **l** = Low Prices, **c** = Close Prices

3.3 Natural Language Processing

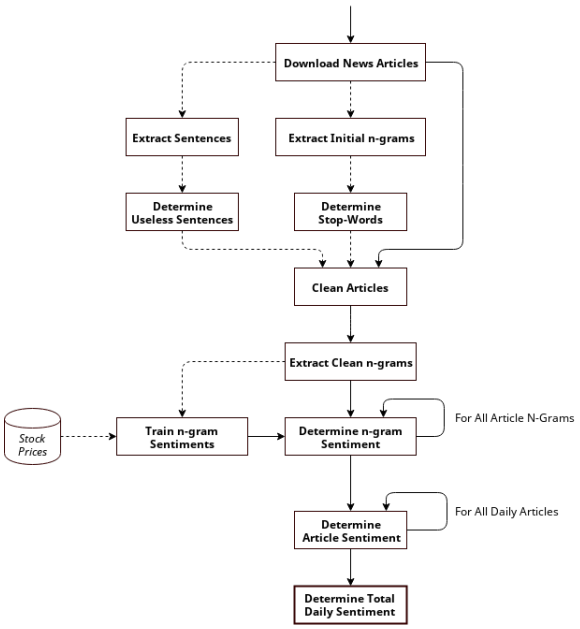

Figure 3.3: Process flow of news article Sentiment Analysis

3.3.1 News Article Retrieval

News Articles are collected on a per-stock basis, requesting data via the *INTRINIO* API which allows filtering of articles by stock ticker. Data is then returned in JavaScript Object Notation (**JSON**) format with a limit of 100 articles per page or a CSV file with a limit of 10,000 articles per page.

3.3.2 News Article Storage

News documents are stored in a MySQL/MariaDB database with relevant auxiliary identification flags to allow for further filtering. These flags include:

- BLACKLIST: Ignores the article in further processing - necessary where articles are no longer available or are hidden behind a *pay-wall* but metadata such as headlines and summaries are still stored.
- DUPLICATE: Ignores the article in further processing - necessary where news websites are based on collecting articles verbatim from other outlets.
- REDIRECT: Allows for further article gathering - necessary where a URL no longer points to the relevant news article.
- ENUMERATED: True when sentences have been entered into the Sentences table and checked for uniqueness (only completely unique sentences are considered - the majority of duplicate sentences are advertisements or other irrelevant information).
- TOKENISED: True when n-grams have been extracted from the document after removing *useless sentences*.
- PROCESSED: True when the article has been given a sentiment value based on assigning values to n-grams from historic price changes.

3.3.3 Sentiment Analysis

Sentiment Analysis is performed on a corpus of news articles relating to a certain stock. The algorithm used is a common n-gram approach that counts the respective number of days with increasing and decreasing prices and uses the separate counts as probabilities of price direction on a given day, based on unseen n-grams contained within news for that day.

Algorithm 1 N-Gram Enumeration

```
1: procedure EnumerateNGrams(NewsArticles)                    ▷ Input the document corpus
2:     numberOfDocuments ← length(NewsArticles)
3:     for currentDocument = 1 to numberOfDocuments do
4:         NGrams[ ]← splitToNGrams(currentDocument)
5:         numberOfNGrams ← length(NGrams)
6:         for NGram = 1 to numberOfNGrams do
7:             NGram.incOcc ← inc = getIncreasingOccurrences(PriceData, NGram)
8:             NGram.decOcc ← dec = getDecreasingOccurrences(PriceData, NGram)
9:             NGram.totalOcc ← inc + dec
```

Algorithm 2 Sentiment Calculation

```
1: procedure CalculateSentiment(NGrams)                       ▷ Input the n-gram
2:     numberOfNGrams ← length(NGrams)
3:     for NGram = 1 to numberOfNGrams do
4:         NGram.Sentiment ← NGram.incOcc/NGram.totalOcc
```

3.4 Portfolio Optimisation

3.4.1 Calculation of Risk and Reward

In order to generate a profitable portfolio, the expected reward of a stock needs to be calculated, which considers the *price history of stock within a given time-frame* and a *hold period* (how long a stock is kept in the portfolio before selling) to determine an average return percentage of any given investment. This can be used to tailor the portfolio for long-term or short-term investment by considering only recent prices or, for example, the entire price history of a stock.

Using this return information, the volatility/variance of the stock must be calculated and used as a measurement of how much investment risk is associated with a stock. This is done by obtaining the *average* return of a stock and then calculating the dispersion of each return away from that mean - the *variance*. A stock with high variance will exhibit large fluctuations away from its mean value $\bar{\boldsymbol{x}}$, meaning that it *could* result in a sudden high-percentage return but could also result in a loss of equal magnitude. Stocks with low variance will increase or decrease steadily over time with very few fluctuations - this is a safer method of investing but is only suitable for long-term investment strategies or short-term investments where low but more reliable returns are desired. Since we are considering *multiple* stocks, the performance between the stocks must also be considered by calculating the *covariance* of *Stock A* and *Stock B*.

Return and Average Return

$$R_i = \frac{x_i}{x_{i-d}} - 1 \tag{3.3}$$

$$r_i = ln(\frac{x_i}{x_{i-d}}) \tag{3.4}$$

$$\text{Average Return: } \mu = \bar{\boldsymbol{r}} = \frac{\sum_{i=d}^{n} r_i}{n-d} \tag{3.5}$$

Where

R_i = Arithmetic return of day i
r_i = Logarithmic return of day i
n = Total number of days
d = Hold period

Arithmetic return refers to the exact percentage returned from an investment and is useful for interpreting the financial gain of a return. *Logarithmic return* gives similar results to arithmetic return (which can be used as approximations) but has the benefit of symmetry when considering sequential price changes. For example, starting with £100, an increase of 20% and then a decrease of 20%:

$$R_i = (\tfrac{120}{100} - 1) + (\tfrac{100}{120} - 1) = 0.03333333333$$

$$r_i = ln(\tfrac{120}{100}) + ln(\tfrac{100}{120}) = 0$$

Variance and Covariance

$$\text{Mean value: } \bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3.6}$$

$$\text{Variance: } var(\boldsymbol{x}) = \frac{\sum_{i=1}^{n} (x_i - \bar{\boldsymbol{x}})^2}{n-1} \tag{3.7}$$

$$\text{Covariance: } cov(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{n} (x_i - \bar{\boldsymbol{x}})(y_i - \bar{\boldsymbol{y}})}{n-1} \tag{3.8}$$

Averages of variance and covariance are calculated using division by *n-1* due to using the *sample distribution* of a stock's price rather than the *total population*.

Covariance Matrix

$$\begin{bmatrix} var(\mathbf{r}_0) & cov(\mathbf{r}_0, \mathbf{r}_1) & \dots & cov(\mathbf{r}_0, \mathbf{r}_n) \\ cov(\mathbf{r}_1, \mathbf{r}_0) & var(\mathbf{r}_1) & \dots & cov(\mathbf{r}_1, \mathbf{r}_n) \\ \vdots & \vdots & \ddots & \vdots \\ cov(\mathbf{r}_n, \mathbf{r}_0) & cov(\mathbf{r}_n, \mathbf{r}_1) & \dots & var(\mathbf{r}_n) \end{bmatrix} \tag{3.9}$$

Where $\mathbf{r}_i$ represents the vector of returns for stock i

N.B. $var(\mathbf{r}_x) = cov(\mathbf{r}_x, \mathbf{r}_x)$ and $cov(\mathbf{r}_x, \mathbf{r}_y) = cov(\mathbf{r}_y, \mathbf{r}_x)$ $\therefore$ matrix is square and symmetric.

The covariance matrix is used to determine the risk of two stocks being in a portfolio together, whereby the joint risk is calculated as:

$$w_x w_y cov(\mathbf{r}_x, \mathbf{r}_y) \tag{3.10}$$

Where w_x and w_y are the weights associated with stocks x and y, and $\mathbf{r}_x$ and $\mathbf{r}_y$ are their respective arrays of returns.

Expected Return and Expected Risk

The Expected Risk and Return refers to the return and risk of the portfolio as a whole, rather than considering each stock individually and is used as a measurement of a portfolio's performance. The Expected Return requires a vector $\mathbf{w}$ of weights and a vector of average returns $\boldsymbol{\mu}$, one for each stock. The value is the summation of each weight multiplied by its respective average return value, as such:

$$\sum_{i=1}^{|\mathbf{w}|} w_i \mu_i \tag{3.11}$$

The Expected Risk similarly requires the weight vector $\mathbf{w}$ but uses the covariance matrix described earlier:

$$\sum_{i=1}^{|\mathbf{w}|} \sum_{j=1}^{|\mathbf{w}|} w_i w_j cov(\mathbf{r}_i, \mathbf{r}_j) \tag{3.12}$$

Optimisation Performance Function

There are multiple ways in which a Mean-Variance optimisation problem can be evaluated, depending on the requirements of the user. For example a greedy approach, whereby the goal is to simply receive the greatest amount of potential profits, would require the *maximisation of the reward* with no consideration of the risk. Similarly, if the goal were to take as little investment risk as possible, the problem would require *minimisation of the portfolio's variance*. In this project however, we are considering the balancing of reward vs. risk in order to provide a safer investment strategy. This can be done through maximising the ratio of reward-to-risk as such:

$$\max \frac{expectedReturn}{expectedRisk} \tag{3.13}$$

Considerations of the Mean-Variance Approach

According to Roudier[24], the Mean-Variance approach favours assets with a higher expected return, which can negatively impact the diversification of a portfolio. Roudier also discusses the commonly encountered problem with Mean-Variance optimisation, in which transactions costs are not taken into account - a concept which can lead to falsely optimistic results. Remedying this issue can be as simple as including *constraints* within the evaluation function, penalising solutions based on their costs (commission, transaction costs etc.).

3.4.2 Optimisation Constraints

Weighting of Stocks

Given a maximum investment amount, the goal is to assign a certain percentage of this amount to each stock. This is represented through a vector $\mathbf{w}$ of stock *weights* (with each stock having one weight), which determines the fraction of the total amount that is available for investment in a given stock. As this is a *percentage* representation, the sum of the weights must be constrained to 1, as such:

$$\sum_{i=1}^{|\mathbf{w}|} w_i = 1 \tag{3.14}$$

During the process of optimisation, should this constraint be violated, the solution can be penalised via negatively impacting the *fitness/objective function* or the weights can be scaled up or down, until the constraint is satisfied. Negative portfolio weights represent *short-selling* - the concept of 'borrowing' a stock from a lender; our implementation will set negative weights to zero, disallowing such borrowing. For evaluation function simplicity, we will scale weights at each iteration, resulting in zero-weighted stocks being excluded from the final portfolio.

Transaction Costs

Transaction costs refer to the fees involved with the purchase and sale of shares, including commission, bid/ask prices, price impact costs (large transactions can affect the price of the stock) and the cost of transferring the stock to the new owner. The formulation for this, as stated by KV Fernando[12] is:

$$\text{Cost of buying: } C_i^B = \begin{cases} (w_i' - w_i)b_i, & \text{if } w_i' > w_i \\ 0, & \text{otherwise} \end{cases} \tag{3.15}$$

$$\text{Cost of selling: } C_i^S = \begin{cases} (w_i - w_i')s_i, & \text{if } w_i' < w_i \\ 0, & \text{otherwise} \end{cases} \tag{3.16}$$

$$\text{Total Transaction Cost: } \sum_{i=1}^{n}(C_i^B + C_i^S) \tag{3.17}$$

Where b_i is the cost for buying stock i and s_i is the cost for selling stock i. This can be a constant value or a percentage of the stock's value (the approach considered in this project).

Fernando's method considers a minimisation approach for integration of this constraint however, this project considers a maximisation approach and therefore the formula to maximise becomes:

$$\max(\frac{expectedReturn}{expectedRisk} - \sum_{i=1}^{n}(C_i^B + C_i^S)) \tag{3.18}$$

Where n is the number of stocks and i is the index of the stock.

3.4.3 Genetic Algorithm

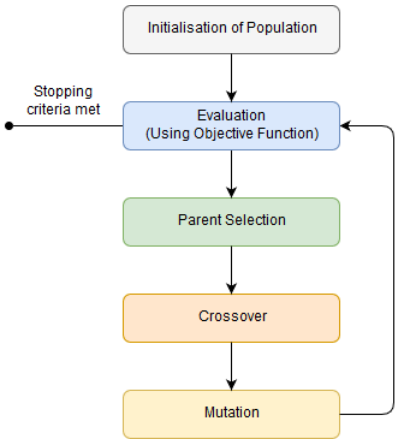

Figure 3.4: Overview of a Genetic Algorithm

A Genetic Algorithm (**GA**) is a method of global optimisation that mimics the process of evolution, through acting on a *chromosome* of data which represents some form of information. This includes *selection* of suitable parent chromosomes, *crossover* or *reproduction* of the two parent chromosomes and *mutation*, which perturbs the chromosome away from the rest of the population, potentially finding more optimal solutions. They are often preferable to other Artificial Intelligence (**AI**) methods as they can handle *noisy* environments well; as discussed earlier, this is the case with stock market data.

As we are optimising the weighting of the stocks in a portfolio, to minimise risk and maximise return, the information within each gene will consist of the percentage weight values from the vector $\mathbf{w}$.

Selection

Within this project, selection of parent chromosomes will be achieved using *Roulette Wheel Selection* or *Fitness Proportionate Selection*, where the selection process is biased towards greater performing chromosomes, whilst still having the chance of selecting poorer performing chromosomes. Each chromosome is assigned a percentage, based on its fitness and the total fitness of the population, which acts as its probability of being chosen.

Algorithm 3 Roulette Wheel Selection

```
1:  procedure RWSelection(population)                    ▷ Input chromosomes
2:      totalFitness ← 0
3:      selectionPoint ← rand()
4:      currentPos ← 0
5:      probabilities[] ← []
6:      for current = 1 to length(population) do
7:          totalFitness = totalFitness + evaluate(population[current])
8:      for current = 1 to length(population) do
9:          probabilities[current] = evaluate(population[current])/totalFitness
10:     while currentPos < selectionPoint do
11:         currentPos = currentPos + probabilities[selectedChromosome++]
        return population[selectedChromosome]
```

Crossover

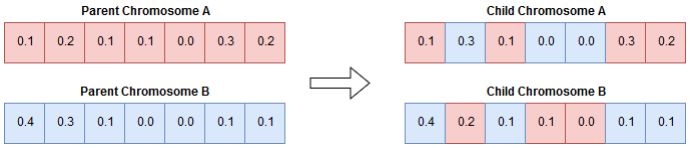

Figure 3.5: Uniform Crossover

Crossover refers to the taking of values from two parent chromosomes to form child chromosomes with features inherited from both of the parents. Here, *Uniform Crossover* is used, where interchange of data is based at the single gene level rather than swapping whole sub-sections of genes. A *crossover rate* is used to determine how likely it is that two genes will swap places and is usually a high value such as 0.5, meaning 50% chance.

Algorithm 4 Uniform Crossover

```
procedure CROSSOVER(parent1, parent2, rate)        ▷ Input parent chromosomes
    if length(parent1) ≠ length(parent2) then return
    child1 ← parent1
    child2 ← parent2
    for current = 1 to length(parent1) do
        if rand() < rate then
            swap(child1[current], child2[current])
    return [child1, child 2]
```

Mutation

Mutation is a method of perturbing a chromosome away from the rest of the population, to potentially find a more optimal solution. This is done by altering the value of a gene based on some mutation method and a *mutation rate*, which is usually fairly low, for example 0.1 or 10% chance. The method used here simply increments or decrements mutated genes with a Gaussian value and then scales the chromosome to be within the constraint of genes summing to 1.

Algorithm 5 Mutation

```
procedure MUTATE(chromosome, rate)        ▷ Input chromosome and mutation rate
    for current = 1 to length(chromosome) do
        if rand() < rate then
            chromosome[current] ← chromosome[current] + Gaussian()
    return chromosome
```

Note: *Gaussian()* refers to a function that generates a number between -1 and 1 with a normal distribution with *mean 0* and *standard deviation 1*

3.4.4 Simulated Annealing

Simulated Annealing (**SA**) is another global optimisation method, inspired by the controlled heating and cooling of materials in order to result in an optimal formation. The algorithm accepts any improving solutions as the new *best solution* but can also accept worse solutions based on the current *temperature* value, giving it an edge on other search algorithms by allowing exploration of the search space, not just exploitation[31]. Given an initial temperature $t \leq 1$, a minimum temperature $t_0 > 0$ and a cooling rate $1 < c < 0$, the algorithm is given as:

Algorithm 6 Simulated Annealing

```
1: procedure SAOptimiser(initTemp, minTemp, coolRate, iteration)
2:     currTemp ← initTemp
3:     solution, currSolution ← initSolution()
4:     bestFitness, currFitness ← evaluate(solution)
5:     while currTemp > minTemp do
6:         for i = 0 to iterations do
7:             candSolution ← mutate(currSolution)
8:             candFitness ← evaluate(candSolution)
9:             if candFitness > currFitness then
10:                currFitness ← candFitness
11:                currSolution ← candSolution
12:                if candFitness > bestFitness then
13:                    bestFitness ← candFitness
14:                    solution ← candSolution
15:            else if e^((currFitness − candFitness)/currTemp) > rand() then
16:                currFitness ← candFitness
17:                currSolution ← candSolution
18:        currTemp ← currTemp ∗ coolRate
       return solution
```

3.5 Preliminary Experiments

Before implementation of the Automated Trading System, a suitable model must be determined along with effective input features, hyper-parameters and output format. In this section we will consider the input features and Machine Learning models available to us and determine their usability within the application empirically.

3.5.1 Determining Appropriate Input Features

Applying models straight to pure stock price data can have contrasting results. Using *Google*'s stock history (GOOGL) to predict future prices results in an increasingly low accuracy as the prediction time-frame increases, in comparison to *Microsoft* (MSFT), well known for having a steady stock price, which also returns an lowering accuracy but at a slower gradient. Various models are capable of improving such predictions, as the Stage 1 results show.

Stage 1. Default Models & Price Data

Using model suggestions based on *Auto-WEKA*, experiments were performed to determine the most accurate model across multiple stocks (using stocks from the NASDAQ 100 index, of which a list can be found within the appendix) using default parameters. At this stage, input features are **OHLCV** values and output values are either *0 or 1* for classification, to represent *decreasing and maintaining/increasing* prices respectively. Accuracy is calculated for *n days* where $n = [1, 30, 200]$ as a representation of short, medium and long-term prediction accuracies. This stage of experimentation is used as a baseline to measure the level of improvement caused by adding features, cleaning data and model parameter alteration.

Model	Hyper-parameters	Accuracy		
		1-Day	*30-Day*	*200-Day*
Logistic Regression	N/A	53.1527%	58.4285%	66.638%
SVM	**Kernel**: Poly	53.1501%	58.5032%	63.4354%
	Kernel: Normalised Poly	54.963%	62.6858%	75.3418%
	Kernel: Radial Basis Function	54.963%	62.6858%	75.3418%
Random Forests	**Max. Depth**: 30	52.4488%	**64.453%**	**77.0516%**
MLP	**LR**: 0.3, **Hidden Layers**: 1	53.1495%	58.3978%	66.5574%
	LR: 0.3, **Hidden Layers**: 3	53.1495%	58.3978%	66.5574%
	LR: 0.1, **Hidden Layers**: 1	53.291%	58.3978%	66.5574%
	LR: 0.1, **Hidden Layers**: 3	53.1474%	58.3978%	66.5574%

Table 3.1: Stage 1 Experiments: Classification Accuracy Results

As can be seen from the above results, arbitrary selection of models can have a large but not necessarily useful effect on output accuracy. It can be seen that some models are more suitable for this problem, given basic data, although with such high error values, using these models with this data will not be ideal in the final system. In the next experiment, inclusion of additional technical indicators as features will be evaluated based on how they affect these accuracy values.

For classification of price direction, identical accuracies (for example - four predictions of 58.3978% and 66.5574% across MLP) are caused by *always* predicting up **or** down; since the problem is binary classification, we already have a 50% or '*coin flip*' average accuracy without using a prediction model. Results such as 62.6858% are also due to predicting the same output for each instance, however at 30-day prediction they have a greater bias due to there being a medium-term increasing trend, leading to a misleading increase in accuracy.

To improve upon these scores, the data can be exponentially smoothed. Exponentially smoothing the previously used data will allow for any variances within the values to be reduced, allowing for greater prediction accuracy. Here, the intensity of smoothing is determined based on a smoothing rate $0 \leq \alpha \leq 1$ whereby a value of 1 results in no smoothing and values approaching 0 result in an increased degree of smoothing.

Due to excessive time requirements during training of Normalised Polynomial and Radial Basis Function SVMs, only the Polynomial SVM is considered from now on.

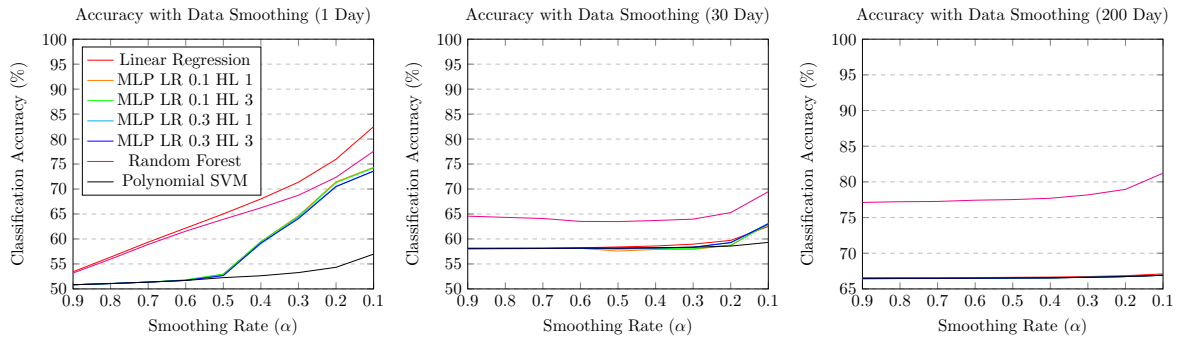

Figure 3.6: Stage 1 Accuracy Improvement of Prediction Models by Smoothing Price Data

Stage 2. Price Data & Technical Indicators

Using the previous pure OHLCV data as well as Technical Indicator information, the experiments were repeated.

Model	**Hyper-parameters**	**Accuracy**		
		1-Day	*30-Day*	*200-Day*
Logistic Regression	N/A	53.1465%	58.5938%	67.0181%
SVM	**Kernel**: Poly	53.2004%	58.2614%	66.5463%
Random Forests	**Max. Depth**: 30	53.2653%	89.9474%	96.787%
MLP	**LR**: 0.3, **Hidden Layers**: 1	53.2081%	58.5604%	67.0029%
	LR: 0.3, **Hidden Layers**: 3	53.1905%	58.5604%	67.0029%
	LR: 0.1, **Hidden Layers**: 1	53.1536%	58.5604%	67.0029%
	LR: 0.1, **Hidden Layers**: 3	53.3944%	58.5604%	67.0029%

Table 3.2: Stage 2 Experiments: Classification Accuracy Results

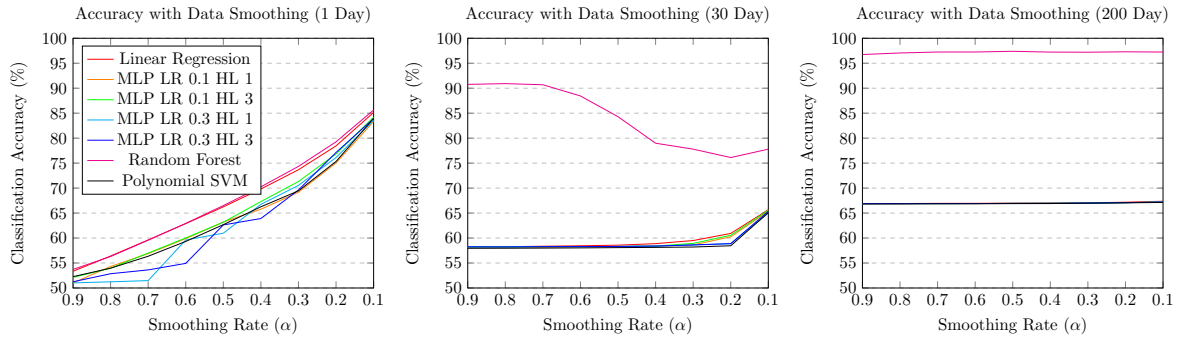

Figure 3.7: Stage 2 Accuracy Improvement of Prediction Models by Smoothing Price Data

Observing the *Accuracy Increase by Exponential Smoothing* graphs (figure 3.7), it can be seen that inclusion of technical indicators greatly increases the accuracy of prediction models, with *Random Forests* taking a substantial lead. More interestingly, in the medium-term (here, 30 days), the greater the degree of smoothing introduced, the worse the performance of the Random Forest becomes. At 200 days, the accuracy improvement is minimal, as any trends at this point are fairly trivial to spot, even without a Machine Learning model.

Stage 3. Price Data, Technical Indicators & News Article Sentiment

Using the same baseline models from the previous stage, Sentiment Analysis is performed on relevant news articles and input alongside price data.

Interestingly, news sentiment data can negatively impact the classification accuracy of the models. This is likely due to the lack of available news articles dating back to the earlier stock price records. This could be remedied in future work through the procurement of a historic news API (mostly likely at some financial cost) and then processing that sentiment of those articles. In conjunction with this, other text-based information sources could be incorporated into the sentimental value of a stock, such as social networks (Twitter, Facebook etc.), financial forums or investment chat-rooms. Inclusion and consideration of text data can be more difficult than price data, as there is no guarantee on any given day that text-based data will be published.

Model	Hyper-parameters	Accuracy		
		1-Day	*30-Day*	*200-Day*
Logistic Regression	N/A	53.1492%	58.5965%	67.0148%
SVM	**Kernel**: Poly	53.2004%	58.2614%	66.5463%
Random Forests	**Max. Depth**: 30	53.1694%	89.7415%	96.7497%
MLP	**LR**: 0.3, **Hidden Layers**: 1	53.2037%	58.5604%	67.0029%
	LR: 0.3, **Hidden Layers**: 3	53.1988%	58.5604%	67.0029%
	LR: 0.1, **Hidden Layers**: 1	53.158%	58.5604%	67.0029%
	LR: 0.1, **Hidden Layers**: 3	53.158%	58.5604%	67.0029%

Table 3.3: Stage 3 Experiments: Classification Accuracy Results

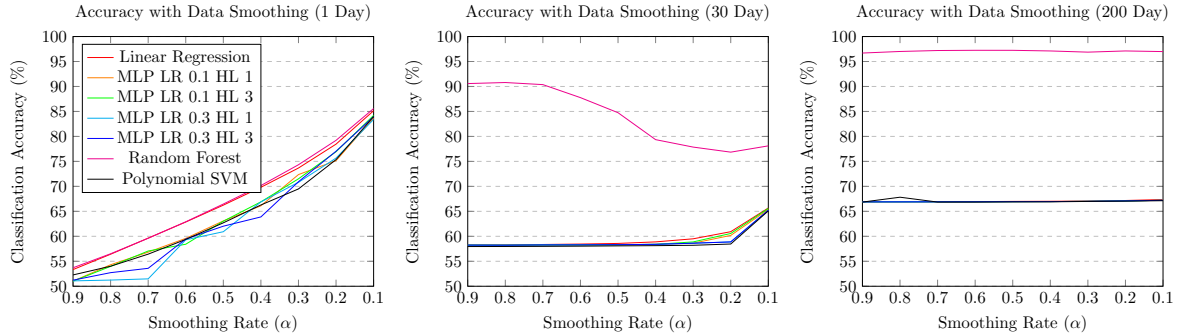

Figure 3.8: Stage 3 Accuracy Improvement of Prediction Models by Smoothing Price Data

3.5.2 Machine Learning Model Selection

Based on the results from section 3.5.1, *Random Forests* are clearly perform the best out of the considered set of potential models; the application will therefore utilise Random Forests when making stock direction predictions. Further reading of the functionality and properties of Random Forests can be found in Breiman's documentation of the model [3].

3.6 Trading Automata

The Automated Trader will use a simple decision system, based on the output of the prediction model. Given a set of time periods to predict price changes for, the Automated Trader will count how many of the predictions suggest a price *rise* and use this to divide the total remaining allocation between the corresponding days. This will then be used to invest in the stocks, whereby the time period to hold the stock is kept in the database until the time period has elapsed. Once this period has passed, the corresponding stock will be predicted again for the same time period. If the prediction results in a *rise*, the stock will be kept in the portfolio and the elapsed time is reset; if *fall*, the stock will be sold.

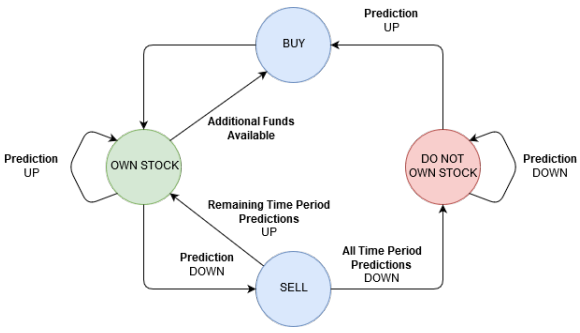

Figure 3.9: Automated Trader Decision System

Implementation

The application described herein is written in Java to allow for rapid prototyping and cross-compatibility over several systems. Similarly, *JavaFX* is used for producing the Graphical User Interface (**GUI**) due to its wide array of existing graphical components such as line graphs, pie charts and progress indicators, and compatibility across several window management systems and Operating Systems (**OS**). Data is stored in and retrieved from a *MariaDB* or *MySQL* database using standard *Structured Query Language* (**SQL**) commands. Using the Agile Methodology and considering the desire for OS cross-compatibility, the project code is managed using *Git*, to allow efficient sharing of code and cascading of updates between multiple systems. For easy integration of required dependencies without inflating the file size of the project directory, any necessary Java ARchive (**JAR**) files are downloaded via *Maven* during the build process and included within the final executable.

4.1 First Time Set-Up

4.1.1 Creation of the Database

Upon the first execution of the application, the database does not yet exist and must be created. This can be done through entering the login information of a user with administrative rights on the database. Using a series of SQL commands, the tables required are created and additional functionalities such as INSERT/DELETE triggers are enabled to migrate processing away from the application, to the Database Management System (**DBMS**), where possible. Linking the Java application to the MySQL server is done using the Java Database Connectivity (**JDBC**) MySQL Connector.

4.1.2 Initial Population of the Database

To reduce the number API calls required to acquire the large backlog of data when first running the application, stock price information can be manually imported into the database by downloading stock history CSV files from *Yahoo! Finance* and saving them in the */res/historicstocks/* directory. Before any API calls are made, this directory will be checked for files and any found will be read and transformed into a *batch SQL command* to allow for efficient mass upload of data to the MySQL server. Reduction of API calls is achieved by querying the date of the last stock record available in the database, for each stock ticker and then requesting data for all dates later than this.

4.2 Downloading Historic Data

4.2.1 Stock Price Data

Price data is downloaded using a URL API request and Java's *HttpURLConnection* class:

```
String request = "https://www.alphavantage.co/query?function=
    TIME_SERIES_" + interval + "&symbol=" + stock + "&outputsize=" +
    outputSize + "&apikey=" + apiKey + "&interval=1min&datatype=csv";

HttpURLConnection conn =
        (HttpURLConnection) new URL(request).openConnection();
Reader reader = new InputStreamReader(conn.getInputStream());
final char[] buf = new char[10240];
final StringBuilder sb = new StringBuilder();
while ((read = reader.read(buf,0,buf.length))>0) sb.append(buf,0,read);
```

The application first requests a bulk download of any historic price data missing from the database and then polls the API every minute for the latest data. Whilst the service covers a large amount of historic data, the application is still initialised using Yahoo! data, as AlphaVantage is missing large amounts of data before the year 2000.

Barchart[1] is a free-to-use service that provides historic daily and minute stock data with greater accuracy over a greater period than *AlphaVantage*. *Barchart* is used interchangeably with *AlphaVantage* within the application; while data from *AlphaVantage* can be unreliable/incomplete and has frequency restrictions, it does not have a daily usage limit. On the other hand, *Barchart* has daily limits but no call restrictions and so can execute requests faster, offer better accuracy and a more complete dataset with data dating back to the Initial Public Offering of each stock. Unfortunately, the free *Barchart* API does not provide real-time data and so is used to update the database only upon initialisation.

Utilising Price Data

Both *Barchart* and *AlphaVantage* are formatted in a similar manner, making it simple to use both APIs with the same code used for parsing requests. Data is retrieved in the following CSV format:

Timestamp, OpenPrice, HighPrice, LowPrice, ClosePrice, TradeVolume

which can be split and entered into the database using simple SQL commands. For example, to submit an Apple Inc. stock quote to the database, the SQL command is:

```
INSERT INTO dailystockprices(Symbol, TradeDate, OpenPrice, HighPrice,
    LowPrice, ClosePrice, TradeVolume)
VALUES('AAPL',"1980-12-12", 0.513393, 0.515625, 0.513393, 0.513393);
```

4.2.2 News Articles

Using the *INTRINIO*[2] API, CSV news information is provided in batches of 10,000 records, therefore multiple calls must be made for stocks having greater than 10,000 articles. The API is first queried for the its metadata of a stock, containing the number of news articles and the amount of pages these records span (i.e. $\lceil \text{number of articles}/10,000 \rceil = \text{number of pages}$).

[1] https://www.barchart.com/
[2] https://intrinio.com/

The application then uses this information to determine how many articles are missing from its database, calculate the page to start requests from based on this value and then download the missing articles with the following API call:

```
String INTRINIO_CSV_CALL = "https://api.intrinio.com/news.csv?page_size
    =10000&ticker=" + stock + "&page_number=" + page;
```

It was found during implementation, that nearly half of the records available from INTRINIO are malformed or do not match the same formatting as the rest of the records. For this reason, it is necessary to explicitly search for values such as *Published Date* using *Regular Expressions*:

```
\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2}\s.\d{4}
```

Once all article information has been downloaded (which can be verified by counting the amount of records in the database and comparing it to the news metadata of the stock), the content of each article is downloaded and processed for sentiment.

4.3 Processing of Downloaded Data

4.3.1 Exponential Smoothing of Price Data

Price data is first Exponentially Smoothed, using the following snippet of code, where **a** is the *Smoothing Rate (α)*:

```
smooth[0] = closePrices[0];                                    (Equation 3.1)
for (int i = 1; i < smoothed.length; i++)
        smooth[i] = a*closePrices[i]+(1-a)*smooth[i-1];        (Equation 3.2)
```

4.3.2 Technical Indicator Extraction

Technical Indicators are extracted from the smoothed price history of each stock, using the **TA-Lib**[3] library. TA-Lib takes an array of OHLCV data and outputs an array of the corresponding indicator, for example obtaining the Exponential Moving Average of a stock:

```
ta.ema(0, closePrices.length-1, closePrices, days, begin, length, out);
```

Where *begin* and *length* manage the bounds/size of the output array '*out*' .

As well as this, the percentage of price change from one day to the next is calculated using:

```
for(int i = 1; i < percentChangeArray.length; i++){
        double currPrice = records.get(dates.get(i));

        percentChangeArray[i] = (currPrice - prevPrice)/prevPrice;
        prevPrice = currPrice;
}
```

4.3.3 News Article Cleaning & Sentiment Calculation

The document is first cleaned and split into sentences by removing any punctuation or *blacklisted sentences* (sentences deemed to be spam or irrelevant, based on excessive frequency, for example *"INHERENT IN ANY INVESTMENT IS THE POTENTIAL FOR LOSS"*). Logical operators such as negation of text can be encoded with certain words, for example *"not okay"* becomes

[3] https://github.com/BYVoid/TA-Lib

"!okay", so that the sentiment of negated text can be processed separately to the non-negated version of the text.

Sentences are then further split into *n-grams* (word collections of n size), which are then saved in the database. Once all n-grams have been collected, their sentiment is decided based on counting how many documents they were found in, the number of documents that were published on days with a *rising* or *falling* stock price and then dividing the number of increasing occurrences, by the total number of n-gram occurrences. N-Gram data '`result`' is retrieved from the database using the SQL query:

```
SELECT Increase, Decrease FROM ngrams WHERE Hash=MD5(word);
```

Sentiment is determined using the following code:

```
double increase  =  Double.parseDouble(result.split(",")[0]),
       decrease  =  Double.parseDouble(result.split(",")[1]);
double sentiment =  increase / (increase + decrease);
```

4.4 Live Mode

Live Mode involves the polling of APIs and processing of their data on a minute-by-minute basis. The processed data is then sent to the prediction model to acquire price direction predictions for making trading decisions. In order to better secure investments, *automatic stock trading* considers only a subset of the total collection of stock tickers, selected based on Portfolio Optimisation which also limits the funds that a stock can have invested. When trading manually, these restrictions are not imposed. Portfolio Optimisation can be performed manually or automatically, based on calculated rebalancing bounds.

4.4.1 Portfolio Optimisation

Shared Functions consist of methods that are common between multiple methods of Portfolio Optimisation, such as perturbation and constraint mechanisms.

Mutation Perturbation of a feasible solution is done by adding a Gaussian value between -1 and 1 to each asset weight in order to move away from an area of the search space. The degree to which an asset weight array is mutated can be set via a *mutation rate*, which determines the probability of each individual weight being altered.

```
for (String stock : weights.keySet())
  if (Math.random() < rate) weights.put(stock, Math.max(weights.get(
     stock) + rng.nextGaussian(),0));
return weights;
```

Constraint: Sum-to-One (Equation 3.14) Given that a portfolio is a collection of percentage allocations, it must be ensured that the total of these weightings is 100%, or 1.0. Using *Java 8 Streams* this can be achieved with simple and elegant code, where x is the sum of weights and w is the array of weights:

```
double x =w.entrySet().stream().mapToDouble(Map.Entry::getValue).sum();
return x >= 0.99 || x <= 1.01;
```

Since the code uses *double* values when calculating the sum, it is possible that precision errors will occur, meaning the summation of the weight array will likely never be *exactly* 1. Therefore, a slight grace buffer of $0.99 \leq x \leq 1.01$ is used to determine if the *Sum-to-One* constraint has been violated.

Constraint Rectification: Weight Scaling If it is found that the *Sum-to-One* constraint *has* been violated, then the weight array can be scaled to have a sum of 1, as follows:

```
weights.entrySet().forEach(x -> x.setValue(x.getValue() / sum));
```

Constraint: Transaction Cost (Equation 3.17) Reduction of buying and selling costs in trading is an important issue to factor in to any optimisation algorithms. Inclusion of this constraint ensures that alteration of a portfolio is discouraged unless it results in a greater profit than cost. Buying costs are considered where a weight at time t for a stock is greater than the weight for the same stock at time *t-1*. Selling costs, inversely, are considered where a weight at time t is less than at time *t-1*. Weights that are equal between times t and *t-1* represent no change and therefore do not incur any fees.

```
double sellCost = 0, buyCost = 0, currCost;
for (String stock : originalWeights.keySet()) {
        double originalWeight = originalWeights.get(stock),
               newWeight      = newWeights.get(stock);
        if ((currCost = (originalWeight - newWeight)) > 0)
                sellCost += currCost * sellCosts.get(stock);
        else if (currCost < 0)
                buyCost += Math.abs(currCost) * buyCosts.get(stock);
}
return sellCost + buyCost;
```

Evaluation Function: Risk (Equation 3.10) Calculation of a portfolio's risk can be done by multiplying the weight of two chosen stocks by each other and their respective entry in the covariance matrix (Equation 3.9), like so:

```
double portRisk = 0;
for (String stock1 : weights.keySet())
        for (String stock2 : weights.keySet())
                portRisk += weights.get(stock1) * weights.get(stock2) *
                    riskCovarianceMatrix.get(stock1).get(stock2);
return portRisk;
```

Evaluation Function: Return (Equation 3.11) Calculation of a portfolio's expected return can by done by multiplying the weight of the stock by the average (expected) return of its price history, like so:

```
double portReturn = 0;
for (String stock : expReturns.keySet())
        portReturn += weights.get(stock) * expReturns.get(stock);
return portReturn;
```

Evaluation Function: Return to Risk Ratio (Equation 3.13) The *Return to Risk Ratio* is calculated using the values returned by the two previous *Evaluation Functions* `portReturn` and `portRisk` in a *ternary operator*, to allow checking for zero values and therefore prevent division by zero errors:

```
return (portReturn == 0 || portRisk == 0) ? 0 : portReturn / portRisk;
```

Constraints can then be incorporated into this through simple subtraction (for maximisation) or addition (for minimisation), for example *Transaction Cost*:

```
fitness -= getTransactionCosts(currPort, newPort, buyCosts, sellCosts);
```

Genetic Algorithm (Section 3.4.3)

Based on the pseudo-code listed in section 3.4.3, the algorithm was implemented using a series of *Java Collections* objects. During the first implementation, *primitive Java arrays* were used, which caused issues when copying values due to incorrect use of the `.clone()` function. Considering future maintenance of the software, it was decided that object arrays and maps were a more suitable and developer-friendly method, which allowed more reliable access to arrays through key/value pair searching, rather than array index access. This in turn uncovered issues where the indices of the array being accessed did not link to the stock that they were expected to be connected to. This was also resolved through the migration to *Java Collections*.

As an addition, *Elitism* was introduced into the implementation, to ensure populations can not be worse than their predecessors, by copying the *best performing* solutions of a population.

```
for (int i = 1; i <= numOfGenerations; i++) {
  bestOfPop = getBestOfPop(Math.round(pop.size() / 10), pop);
  ArrayList<Genome> parent1 = selection(pop), parent2 = selection(pop);
  pop = crossover(parent1, parent2, 0.8);
  pop = mutate(population, 0.1);
  pop = replaceWorstOfPop(bestOfPop, pop);
  currFitness = evaluate(pop,currPort,stockPrices,expReturns,riskMat);
  if (currFitness >= bestFitness) bestFitness = currFitness;
  if (bestOfPop[0].getFitness()>=(prevFitness=bestGenome.getFitness())
      bestGenome = bestOfPop[0];
}
return bestGenome.getGenes();
```

Simulated Annealing (Section 3.4.4)

Due to the standard formulation of the Simulated Annealing algorithm, as outlined in section 3.4.4, being a *minimisation* approach, the implementation had to be altered to allow for the representation of a *maximisation* problem, which was as simple as inverting the fitness calculations. The algorithm was further simplified by combining the checks for *improving score* and *probabilistic worsening movement acceptance* into one `if` statement and calculating the change in fitness only once and storing it in `delta`. This allows for greater readability and understanding of the code and reduces the footprint of the Simulated Annealing optimisation method.

```
double temp = initialTemp;
while(temp > minimumTemperature){
   for(int i = 0; i < iterations; i++) {
     candSolution = mutate(currSolution, 0.5);
     double fitness = evaluate(candSolution, expReturns, riskCovarMat),
            delta   = currentFitness - fitness;
     if (delta < 0 || Math.exp(-delta / t) > Math.random()) {
        currentFitness = fitness;
        currentSolution = new TreeMap<>(candidateSolution);
        if (fitness >= bestFitness) {
           bestFitness = fitness;
           solution = new TreeMap<>(candidateSolution);
        }
     }
   }
   temp *= coolRate;
}
return solution;
```

4.4.2 The Automated Trader

Enabling the Automated Trader

By default the application starts in *Manual mode*, meaning no stocks will be automatically trading (to prevent unauthorised buying or selling of stock); the application can be set to *Semi-Autonomous*, meaning the user can explicitly set their own profit/loss bounds for which the portfolio will be rebalanced, or *Fully-Autonomous*, which sets these bounds automatically based on the *expected return* of the portfolio.

Automated Trading Logic: Buying Stock

To determine how many days to split the total remaining allocation between, predictions must be made for all days and the split amount becomes the sum of days with a *RISE/MAINTAIN* prediction.

```
for (int currDay : dayArray)
        if (StockPredictor.predictStock(stocks, symbol, currDay))
            splitAmount++;
if (splitAmount == 0) return;
```

This value is then used to divide the remaining allocated funds available for purchasing stocks. The `buyAmount` variable represents the rounded integer amount of stocks to purchase, based on the cost of the stock, the remaining funds available and the degree to which this amount is split. If the buy amount is 0, the trade is aborted.

```
double allocRemaining = allocation - investment, value;
int buyAmount = (int) Math.floor(allocRemaining/currPrice/splitAmount);
if (buyAmount == 0) return;
```

Finally, the list of time periods is once again traversed and used to predict the stock direction on multiple days. If the stock is predicted to rise or hold its value, a number of shares of the stock are purchased, based on `buyAmount` and the portfolio allocation/investment values are updated.

```
for (int day : dayArray) {
        if((value  = buyAmount * currentPrice) <= balance
        ||  value <= allocRemaining) return;
        if (StockPredictor.predictStock(stocks, symbol, day)) {
                balance -= value;
                allocRemaining -= value;
                TradingUtils.buyStock(symbol, buyAmount, day, true);
        }
}
```

Automated Trading Logic: Selling Stock

Expired investments (those that have reached or exceeded their holding period) are first retrieved from the database by submitting the following query:

```
SELECT * FROM investments WHERE EndDate <= CURRENT_DATE;
```

Results are stored in an array `expiredInvestments` in the format `[id, symbol, period, amount]`. Reinitialising an investment is done using the following SQL command '`resetInvestment`':

```
UPDATE investments
SET EndDate = DATE_ADD(CURRENT_DATE,INTERVAL period DAY) WHERE ID = id;
```

Selling an investment is done using this SQL command '`removeInvestment`':

```
DELETE FROM investments WHERE ID = id;
```

For each of the investment records, the corresponding stock is once again predicted for the same *investment period* as the current investment record. If the stock is predicted to rise or hold its value, the *elapsed time* of the record is reset to the *investment period*. If the stock is predicted to fall, the stock is sold, selling all shares corresponding to the current investment, as defined by the `amount` variable.

```
for (String investment : expiredInvestments)
        if (StockPredictor.predictStock(stocks, symbol, period))
                databaseHandler.executeCommand(resetInvestment);
        else {
                databaseHandler.executeCommand(removeInvestment);
                TradingUtils.sellStock(symbol, amount), true);
        }
```

4.5 Stock Direction Prediction using Machine Learning

4.5.1 Machine Learning Library

Apache Spark MLLib[20] is used within the application due to its range of configuration options and model scalability. Although this project/application is intended for a home PC, it can be run on large-scale servers using this library. The *Random Forest* family of packages is used here, although (potentially for future works) it offers other classification models such as *Naïve Bayes*, Clustering Methods such as *K-Means* and Regression methods such as *Survival Regression*. The framework also provides a range of tools such as *Feature Selection*, *Data Normalisation* and *Model Evaluation*.

4.5.2 Data Preparation

In order to train and utilise prediction models, a suitable feature vector is constructed using the previously gathered/calculated information and is given as:

[Stock ID[4], Prediction Period, Open, High, Low, Close, Volume, Percentage Change, Smoothed Close Price, 5-Day SMA, 10-Day SMA, 20-Day SMA, 200-Day SMA, 5-Day EMA, 10-Day EMA, 20-Day EMA, 200-Day EMA, MACD, MACD Signal, MACD Histogram, RSI, 10-Day ADX, CCI, AD, OBV, StoOsc K, StoOsc D, William's %R, Sentiment]

To allow usage of this vector by the *Apache Spark MLLib* library, the data must be converted from CSV format to the LibSVM standard[6]. This involves taking the *index/position* i and *value* x_i of each feature in the vector $\mathbf{x}$ and the resultant prediction value y and arranging them in the format:

```
y       0:x[0] 1:x[1] 2:x[2] ... n:x[n]
```

[4]Only used for the Multi-Stock model

4.5.3 Training a Machine Learning Model

Utilising the feature vector files created in Section 4.5.2, the data must be read-in and converted to the specialised data type *JavaRDD*[5], which allows explicit setting of data storage methods such as `MEMORY`, `DISK`, `MEMORY_AND_DISK` etc. which specify where the array of data is kept. This is necessary due to the large nature of the data sets (often in excess of 200MB), so that the *Apache Spark* environment can continue to run efficiently. The dataset can then be split into *training1* and *test* sets, for initialising and evaluating the model.

```
JavaRDD<LabeledPoint> data = MLUtils.loadLibSVMFile(jsc.sc(),
    libSVMFilePath).toJavaRDD().persist(StorageLevel.MEMORY_AND_DISK());

final JavaRDD<LabeledPoint>[] trainTestSplits = data.randomSplit(new
    double[]{0.7, 0.3});
```

Apache Spark has provisions for configuring models to allow for a high level of customisability. Here, since the problem is a *binary classification problem*, we are only considering a result of class 0 or 1 (*RISE/MAINTAIN* or *FALL*), therefore `classes = 2`. The *number of trees* included in the Random Forest, as well as their *Maximum Depth* can greatly affect the accuracy of the model, but also the time and resources required for training trees. This can be dependent on the hardware specification of the machine that the application runs on, however in most cases the process of tuning these settings are a matter of trial and error to balance *performance vs. resource requirements*. With an *i7-8700K 12 Thread 6 Core 3.7GHz* CPU and 16GB of RAM, the optimal settings were found to be `trees = 100` and `maxDepth = 15`, resulting in average model accuracies of around 82% for Single-Stock models. Multi-Stock model accuracies plateau at around 69% with `trees = 500` and `maxDepth = 6`; increasing these values beyond this point results in *Apache Spark Executor Heartbeat Timeout* exceptions.

```
int classes = 2, trees = 100, maxDepth = 15, seed = 12345;

RandomForestModel model = RandomForest.trainClassifier(trainTestSplits
    [0], classes, new HashMap<>(), trees, "auto", "gini", maxDepth, 32,
    seed);
```

4.5.4 Running a Model on Unseen Data

Making predictions based on new data can be done by first formatting the raw data into a *MLLib* accepted format. The format used here is a *Dense* Vector[6], which requires explicit definition of each array element, as opposed to a *Sparse* vector[7] which only requires definition of non-zero values, by giving an array of indices and their corresponding feature values.

```
double featVec[] = new double[splitString.length];

featVec[0] = numberOfDays;
for (int i = 2; i < splitString.length; i++)
        featVec[i-1] = Double.parseDouble(splitString[i]);
featVec[featVec.length-1] = newsSentiment;

return StockPredictor.predictDirection(new DenseVector(featVec),stock);
```

[5]https://spark.apache.org/docs/1.6.0/api/java/org/apache/spark/api/java/JavaRDD.html
[6]https://spark.apache.org/docs/2.0.2/api/java/org/apache/spark/ml/linalg/DenseVector.html
[7]https://spark.apache.org/docs/2.1.1/api/java/org/apache/spark/mllib/linalg/SparseVector.html

4.6 Accelerated Time-Frame Trading Simulation

Due to the time constraints imposed on the project, it would not be possible to evaluate the application on unseen trading days over an extended time-period. Therefore, a non-real-time simulation function was implemented, utilising a subsection of historic stock/news records for training the model and portfolio optimisation and the remaining n days for testing/simulating the automated trader. Using an in-memory representation of the main database as a temporary store for trading information and separate *simulation-only* models, simulations can be performed in complete isolation from the main system.

Separate trading models can be added to the simulation, such as *random trading*, *automated trading*, *buy-and-hold* portfolios, etc. and compared against each other. The functionality of the simulator is exactly the same as the *Live System* but runs in a shorter time-frame, which can be useful for tuning optimisation and prediction parameters.

4.7 Problems Encountered

- **Serial Retrieval and Processing Time**: During tasks that require downloading data from the Internet and/or passing such data through some processing pipeline, for example *stock data requests and their following technical indicator extraction*, it was noted that execution time was not viable for an effective real-time trading solution, often requiring multiple *hours* if there was a backlog of data to process. In order to rectify this issue, concurrent processing and downloading was introduced, as well as performing separate tasks in parallel where possible, having 'blocking' of threads only where necessary, for example sentiment analysis can only be performed when both *News Article Retrieval* and *Stock Data Retrieval* have completed. This greatly improved performance, requiring a fraction of the time of the previous approach, completing jobs in a matter of minutes.

- **Machine Learning Model Resource Requirements**: Whilst training the Random Forests on previous stock data, the application would often fail, giving the error *"Executor heartbeat timed out"*. This was remedied through increasing the amount of memory available to the Java Virtual Machine (**JVM**) and altering the resources available to Apache Spark in the SparkConf variable, where CPU core quantity and RAM-per-core values can be set:

```
private static final SparkConf sparkConf = new SparkConf()
        .setMaster("local[*]")//[number_of_cores], *=All Cores
        .set("spark.executor.cores", "6")//Number of cores
        .set("spark.driver.memory", "2g")//Driver Allocated RAM
        .set("spark.executor.memory", "2g")//RAM per core
```

  Should the issue remain unresolved on some systems, the *heartbeat timeout* value can be increased by increasing the value of *spark.executor.timeout* within the *SparkConf* variable.

- **API Call Limits**: During the time-frame of the project, some API functionalities became degraded due to increasing popularity of the services. As such, API calls became limited by IP address, which degraded the call rate of the API. A workaround for this, was to use a proxy service such as The Onion Router (**TOR**)[8] which allowed the use of the same API key but from alternate IP addresses. This is particularly useful in situations where multiple people may be utilising the API behind the same public IP address but all have their own respective API keys.

[8] https://www.torproject.org/

Evaluation

5.1 Prediction Model Performance

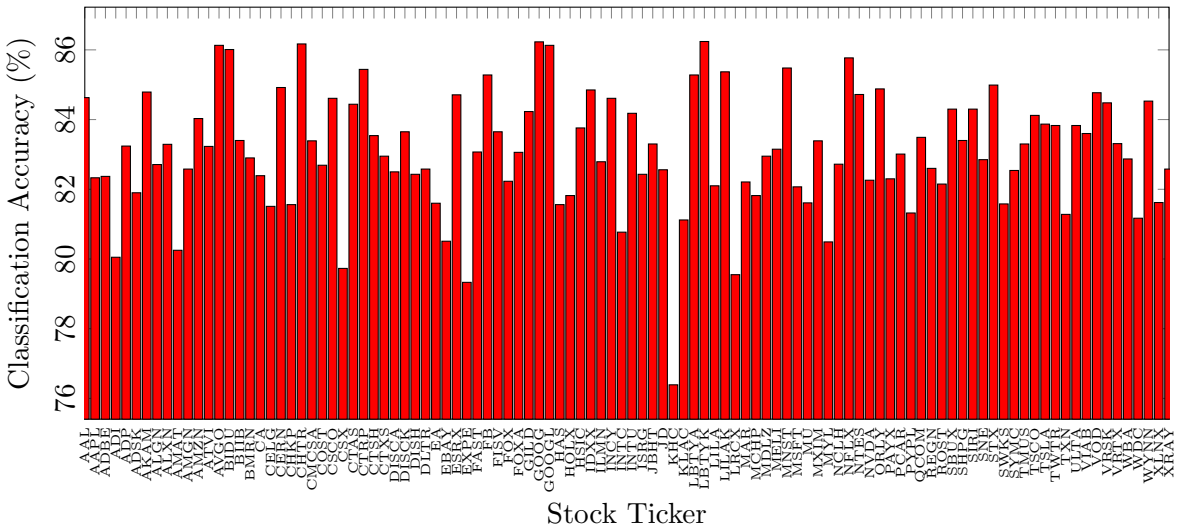

Figure 5.1: Classification Accuracy of Single-Stock Random Forests

During the creation and training prediction models, two approaches were considered:

- *Single-Stock Random Forests* comprising of a separate Random Forest for each stock, allowing for each model to be better trained towards the exact values of a stock's price history without requiring normalisation. Introduction of a new stock requires training a new model and more memory.
- *Multi-Stock Random Forests* which provides a more generalised model for better performance with unseen stocks, but overall poorer accuracy.

After experimenting with the hyper-parameters involved with creating each model, the best found configuration for the Multi-Stock model produced an accuracy of 69% and Single-Stock an average of 82%. Whilst Single-Stock models provided greater accuracy, they required a much larger amount of memory during use. Whilst the results of Random Forests in the *Preliminary Experiments* section of the project had higher accuracies, they considered each time period separately, whereas these models integrate all time periods into the model, by passing it in as a feature in the feature vector.

During training, Multi-Stock models required a longer period of time and greater memory sacrifice, but are faster to evaluate data points and require less memory during use than Single-Stock models. The accuracy of the Single-Stock model, however, was more favourable than speed and memory availability, since the memory usage of the other aspects of the application are negligible compared to the **MLLib** requirements. For this reason, Single-Stock models were used for the final application, however this can be altered within the *settings database*.

5.2 Automated Trading Simulation

Given an initial balance equal to the NASDAQ 100 index value at the start of the simulation, the application simulates automated trading over an extended period of time, by selecting the latest 200 days of price/indicator/sentiment data for each stock as the test/simulation set. The remaining data is used as a training set and as historic return/risk data for the Portfolio Optimiser. For comparison, multiple benchmarks/Automated Trading variants are also simulated, including:

- A *Random Trader*, which randomly decides to buy or sell a stock on each day.
- An *Automated Trader using only an initialised optimal portfolio.*
- An *Automated Trader that rebalances when necessary.*
- An *Automated Trader that starts with equal fund allocation for each stock.*

In addition to this, the latest 200 days of NASDAQ 100[1] price data is retrieved and used for comparison. All functionality of the simulation directly mimics that of *Live Mode* trading, enforcing all constraints (fund allocation limits, investment expiry etc.), Portfolio Optimisation methods and allocation division over multiple time periods.

It is necessary to compare our results to a *Random Trading Strategy* as past research has shown that technical methods of trading have performed no better than a randomised method in the long term[2].

5.2.1 Genetic Algorithm Portfolio Optimisation (GAPO)

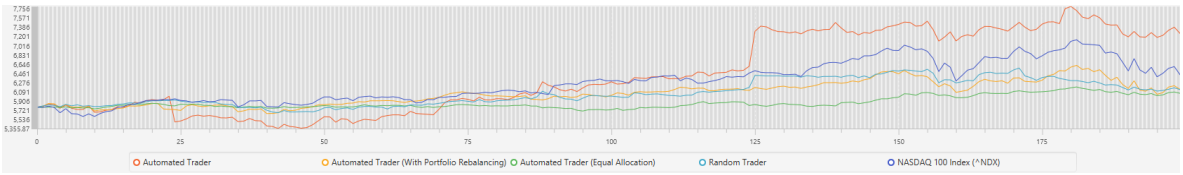

Figure 5.2: Trading Simulation with Genetic Algorithm Portfolio Allocation

Trading Method	Final Balance	Return Value
AUTOMATED TRADER (*Initial Portfolio Optimisation Only*)	7259.18	25.539476158829157%
AUTOMATED TRADER (*Portfolio Rebalancing*)	6140.23	6.188498921925072%
RANDOM TRADER	6134.81	6.0947159203250285%
AUTOMATED TRADER (*Equal Allocation*)	6061.06	4.819200195434618%

Table 5.1: Automated Trader vs. NASDAQ 100 Index: **GAPO** Performance Comparison

As can be seen in figure 5.2, it is possible for the performance of the Automated Trader to first make a substantial loss, however after time it corrects itself and inverts the performance to result in a substantial profit. This would suggest that the total investment time of the Automated Trading application should be carefully contemplated during real-world use. Considering the jump in profit at *125 days*, it is likely that the *Automated Trader* invested heavily in a single

[1]https://www.nasdaq.com/markets/indices/nasdaq-100.aspx

stock at this point, in order to gain a significant improvement over the NASDAQ 100 market performance, a trade of which can also be seen with the *Random Trader* at a less prosperous amount. It should be noted that in this case, portfolio rebalancing has not proved effective in boosting profits but offers a more stable rise in income. The exact outcome of each method can be viewed in table 5.1, showing that the *Automated Trader* made a profit of over 25%, greatly exceeding that of the *Random Trader* and NASDAQ 100 market.

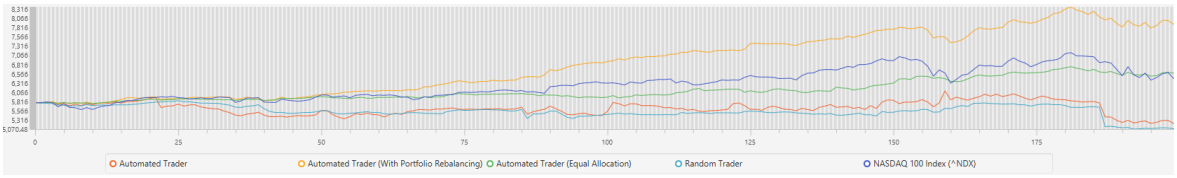

Figure 5.3: Rebalancing Automated Trader outperforms Standard Automated Trader

Running the simulation multiple times has shown that this return is highly dependent on the portfolio that the Automated Trader can trade upon, with some simulations resulting in the *Rebalanced Automated Trader* outperforming the *Standard Automated Trader*, as can be seen in figure 5.3. Optimal performance of the Automated Trading methods can be achieved through tuning of the *look-back period* used to calibrate the Portfolio Optimiser and the *hold period* in which returns are calculated. A higher holding period, calibrated over a greater length of look-back, allows for good long-term results, meaning the *Initialised Portfolio Automated Trader* is likely to outperform the *Rebalancing Automated Trader*. Conversely, using shorter periods will result in better performance during rebalancing, as the portfolio will reflect the latest best performing stocks.

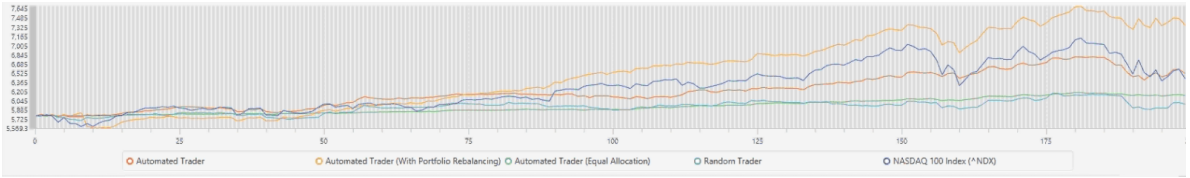

Figure 5.4: Rebalancing and Initialised Automated Traders outperform the NASDAQ 100 index

Careful alteration of these values can result in a more stable income, whereby *both* the Initialised Portfolio Automated Trader and Rebalancing Portfolio Automated Trader outperform the NASDAQ 100 index, as shown in figure 5.4.

Comparison to Random Trading

Analysing the results in table 5.1, it can be noted that the Random Trader outperforms the *Equal Allocation Automated Trader* and is only just outperformed by the *Portfolio Rebalancing Automated Trader*. In this case, it may be true this method is no better than a Random Trader in the long-term, however the *Initial Portfolio Optimisation Automated Trader* successfully outperforms the Random Trader by a significant margin. Considering figure 5.3, the *Random Trader* behaves very similarly to the *Initial Portfolio Optimisation Automated Trader*, with our method only just outperforming the random strategy, suggesting the inclusion of *portfolio rebalancing* may be a viable solution for beating random trading methods. Figure 5.4 is the result of tuning the Portfolio Optimisation module's parameters and shows that, given the correct configuration, all three automated trading strategies can outperform a random trading strategy.

5.2.2 Simulated Annealing Portfolio Optimisation (SAPO)

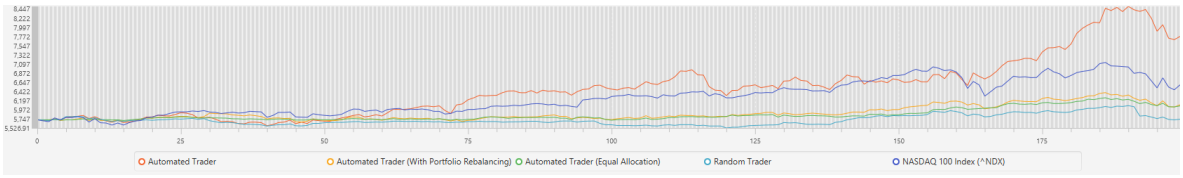

Figure 5.5: Trading Simulation with Simulated Annealing Portfolio Allocation

Trading Method	Final Balance	Return Value
AUTOMATED TRADER (*Initial Portfolio Optimisation Only*)	7768.70	35.64877065056883%
AUTOMATED TRADER (*Portfolio Rebalancing*)	6094.41	6.414100802472945%
AUTOMATED TRADER (*Equal Allocation*)	6066.58	5.928162705472688%
RANDOM TRADER	5740.67	0.23747246803254288%

Table 5.2: Automated Trader vs. NASDAQ 100 Index: **SAPO** Performance Comparison

Use of the Simulated Annealing method results in a more diversified portfolio than the Genetic Algorithm, likely due to the possibility of accepting worsening moves which are more subject to *exploration* than *exploitation*. Whereby the Genetic Algorithm makes its profits from investing heavily in a certain set of stocks, Simulated Annealing has shown here to gain profits from investing in safe and risky stocks in roughly equal measure. This may explain how the SA Automated Trader achieved a greater return of ∼35% using an initial portfolio. Having a more diversified portfolio in the long-term gives more of a chance of good returns by balancing out profits and losses, compared to investing too heavily in few stocks or even a single stock in the long-term which may result in a heavy loss.

Comparison to Random Trading

In this simulation, as shown in figure 5.5, the random trading strategy has performed the worst. The *Rebalancing Portfolio* and *Equal Allocation Automated Traders* behaved very similarly during the simulation and finished only slightly above the *Random Trader*. Whereas portfolio rebalancing for the *Genetic Algorithm* results in changing which stocks are most heavily invested in, *Simulated Annealing* rebalancing results in re-diversification, which in the short term can be harmful to profits, reducing the performance of these methods to almost random trading level. Juxtaposition to this, the *Initial Portfolio Optimisation Automated Trader* greatly outperforms all other methods due to long-term diversification and shows that in the long-term, it is possible to create a strategy that is better than random trading.

Conclusion

6.1 Project Management

The project as a whole went fairly smoothly; due to the wide array of possible features that could be implemented into the system, it wasn't uncommon for a new feature to be suggested and then consequently implemented. Whilst this would take some time out of production of the main features, the additions contributed to a better final system, taking important factors into consideration that were previously ignored (such as trading fees in Portfolio Optimisation). All intended functionality is available within the application, as well as additional utilities such as visualisation tools. The combination of all these implemented features resulted in promising empirical results and a good start to a consumer-level software package.

Project Management using the Agile methodology was a good choice given that the project was executed by a one person team, however if the project were to be continued by a multi-person team then a more structured approach should be taken - at the very least incorporate *Scrum* meetings and *Sprint Cycles* to maximise productivity when performing multiple tasks in parallel. Maintaining the project on *GitLab/GitHub* proved extremely effective, allowing portability of code across multiple systems and geographical locations, as well as having automatic back-up provisions to allow for any issues to be rolled-back. Similarly, storage of project documents on *Microsoft OneDrive* allowed for editing of their content from any device at any time, greatly increasing productivity and reducing any project *time-lags* that cropped up due to unforeseen circumstances.

Time Management was well kept, keeping milestone completion within the expected timelines, despite additional features being introduced into the system. Analysing the Gantt chart from the start of the project (see Appendix A), time allocations to tasks were generous, as previous projects have shown that it's highly likely that certain events will interrupt the schedule, which was beneficial for staying on-track. Many tasks that were not the focus of the project were not included in the Gantt chart (such as code refactoring,) were able to be completed with minimal impact on other tasks.

6.2 Contributions and Reflections

In this project, existing research concerning effective methods of stock prediction has been reviewed and implemented into a usable software package, whereby most implementations of said research were mainly for experimental purposes and were not necessarily implemented as a practical application. The research contained in this project has shown promising results for producing an effective Artificially Intelligent Automated Trading application that can outperform standard methods such as holding a long-term portfolio or using *Algo-Trading*. As can be seen in results table 5.2, the Automated Trader achieved profits of $\sim$35%, which outperforms Busetti's method[4] of $\sim$26% return.

The novelty of the project is within the real-world applicability of the resultant software and the combination of multiple methods of return improvement. Whereas some currently available software may utilise Machine Learning models and Artificial Intelligence in some form, they are often solitary. Within this application, Machine Leaning, Artificial Intelligence and Natural Language Processing has been fused together to create a more superior application. It has also made available the latest research in stock prediction and Portfolio Optimisation, in a tangible, publicly-available package, allowing for commercial use of academic findings.

6.2.1 Revisiting the Research Question

Using current research and considering historic price/technical indicator based prediction, sentimental information and an initial optimal portfolio, can an effective automated stock-trading application be developed that can significantly improve returns over traditional trading?

Given the results from the simulation, whereby the Automated Traders generated returns of between 24% and 35%, exceeding the performance of the NASDAQ 100 index, it can be seen that *it is* possible to produce an effective automated trading application, providing significantly improved returns. That's not to say that the application can't be greatly improved through inclusion of additional models, features and alternate methods for Portfolio Optimisation and Natural Language Processing, but it is certainly a promising start to a commercial application that takes *empirical research* and provides a consumer-level implementation, allowing for practical, real-world use.

6.2.2 Personal Reflection

Producing a complex piece of software without the accompaniment of a development team has certainly allowed for great improvement in programming skills, including refactoring of code to make it read better and run more efficiently. The research phase of the project required careful, critical analysis of academic papers, bringing the necessity of comparing results from various sources and evaluating their usability within this project, improving investigatory and analysis skills. Time management in a scenario where there are few deadlines in between the initial project proposal and final delivery can prove taxing, but from this challenge emerged the demand for self-sufficiency and accountability, providing a source of motivation for completing the project.

Admittedly, at the beginning of the project, the vast mount of features that had to be implemented was quite daunting, giving the feeling of having overestimated ability and available time. Nevertheless, this resulted in a more thorough research process, a more complete application and a more invaluable learning experience.

6.3 Future Work

6.3.1 Expanding to Other Markets

Given the potential application of this system, it would be worth investigating the other markets in which such tools would be useful, such as in the commodities, foreign exchange and cryptocurrency exchanges. The inclusion of such functionality is potentially trivial, given that the *AlphaVantage API* used within this project also provides feeds for these data sources[1][2].

6.3.2 Introducing New Models and Methods

The introduction of novel fusions of AI and Machine Learning approaches such as Ant Colony[5] and Particle Swarm Optimisation[25][1] in stock trading has proven promising for improving prediction accuracy and therefore increase profits.

The application can be extended through a multitude of methods, whether it be the inclusion of more machine learning models, improvement of sentiment analysis methods or considering other sources of data such as crowd-sourced opinion via direct voting or Twitter sentiment mining.

6.3.3 Porting & Mobilisation of Code

The project was implemented in Java due to its inherent cross-compatibility between multiple Operating Systems and ease of GUI creation. It is well known, however, that Java does not best utilise system resources and can result in slower execution and more excessive memory usage than, for example, C++. Whilst implementation of the project in C++ would require OS dependent considerations for OS calls and GUIs, it will greatly increase the performance of the application.

It has become popular in recent times to make applications available on mobile devices (sometimes more-so than on standard computers), due to their portability and ease of use. Whilst it would be infeasible to migrate the entirety of the code onto iOS or Android due to the processing power and battery limitations, it would be worth considering the implementation of the project on a back-end server, accessible by a client app, delegating the extensive processing requirements onto a 3rd party machine.

6.3.4 Integration with Stock Brokers & Quant Services

Stock trading performance has so far only be simulated using a *virtual bank account*. Upon further expansion of the application, perhaps resulting in a more reliable trading strategy, it would be beneficial to integrate the software with various stock brokerage APIs, such as *Interactive Brokers' IB API*[3].

Quant services such as *CloudQuant*[4] allow for testing for trading algorithms, without necessarily requiring monetary investment from developers. Such services often fund the investments made by the algorithms and pay a portion of any profits back to the developer, protecting software writers from losses.

[1] Crypto Currency API call: https://www.alphavantage.co/documentation/#digital-currency
[2] Foreign Exchange API call: https://www.alphavantage.co/documentation/#fx
[3] https://www.interactivebrokers.com/en/?f=%2Fen%2Fsoftware%2Fibapi.php
[4] https://info.cloudquant.com/

Bibliography

[1] Aboueldahab, T., and Fakhreldin, M. Prediction of Stock Market indices using Hybrid Genetic Algorithm/Particle Swarm Optimization with perturbation term.

[2] Biondo, A. E., Pluchino, A., Rapisarda, A., and Helbing, D. Are random trading strategies more successful than technical ones? *PloS one 8*, 7 (2013), e68344.

[3] Breiman, L. Random Forests. *Machine learning 45*, 1 (2001), 5–32.

[4] Busetti, F. Metaheuristic approaches to realistic Portfolio Optimization. *arXiv preprint cond-mat/0501057* (2005).

[5] Cai, Q., Zhang, D., Zheng, W., and Leung, S. C. A new fuzzy time series forecasting model combined with Ant Colony Optimization and Auto-Regression. *Knowledge-Based Systems 74* (2015), 61–68.

[6] Chang, C.-C., and Lin, C.-J. LIBSVM: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST) 2*, 3 (2011), 27.

[7] Chen, K., Zhou, Y., and Dai, F. A LSTM-based method for stock returns prediction: A case study of China stock market. In *Big Data (Big Data), 2015 IEEE International Conference on* (2015), IEEE, pp. 2823–2824.

[8] Cheng, C., Xu, W., and Wang, J. A Comparison of Ensemble Methods in Financial Market Prediction. IEEE, pp. 755–759.

[9] Clarke, J., Jandik, T., and Mandelker, G. The Efficient Markets Hypothesis.

[10] Dunne, M., Bridge, D., and Provan, G. On Stock Market Prediction and Machine Learning, 2015.

[11] Dzikevičius, A., and Šaranda, S. Smoothing Techniques for Market Fluctuation Signals. *Business: Theory and Practice 12* (2011), 63.

[12] Fernando, K. V. Practical Portfolio Optimization.

[13] Forslund, G., and Åkesson, D. Predicting share price by using Multiple Linear Regression., 2013.

[14] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The WEKA data mining software: an update. *SIGKDD Explorations 11*, 1 (2009), 10–18.

[15] Kapiamba, J. N. Simulated Annealing vs. Genetic Algorithm to Portfolio Selection.

[16] Khaidem, L., Saha, S., and Dey, S. R. Predicting the direction of stock market prices using Random Forest. *arXiv preprint arXiv:1605.00003* (2016).

[17] Madge, S. Predicting Stock Price Direction using Support Vector Machines. *Independent Work Report Spring* (2015).

[18] Markowitz, H. Portfolio Selection. *The journal of finance 7*, 1 (1952), 77–91.

[19] Meesad, P., and Rasel, R. I. Predicting stock market price using Support Vector Regression. In *2013 International Conference on Informatics, Electronics and Vision (ICIEV)* (May 2013), pp. 1–6.

[20] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al. Mllib: Machine Learning in Apache Spark. *The Journal of Machine Learning Research 17*, 1 (2016), 1235–1241.

[21] Milosevic, N. Equity forecast: Predicting long term stock price movement using Machine Learning. *arXiv preprint arXiv:1603.00751* (2016).

[22] Mittal, A., and Goel, A. Stock prediction using Twitter Sentiment Analysis.

[23] Pham, H., Chien, A., and Lim, Y. A Framework for Stock Prediction, 2009.

[24] Roudier, F., Sornette, D., and Lapeyre, B. Portfolio Optimization and Genetic Algorithms.

[25] Seidy, E. E. A New Particle Swarm Optimization Based Stock Market Prediction Technique. *International Journal of Advanced Computer Science and Applications 7*, 2 (2016).

[26] Siripurapu, A. Convolutional Networks for Stock Trading, 2015.

[27] Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (2013), ACM, pp. 847–855.

[28] van Witteloostuijn, A., and Muehlfeld, K. Trader personality and trading performance: A framework and financial market experiment. *Discussion Paper Series/Tjalling C. Koopmans Research Institute 8*, 28 (2008).

[29] Wang, Y. Stock price direction prediction by directly using prices data: an empirical study on the KOSPI and HSI. *International Journal of Business Intelligence and Data Mining 9*, 2 (2014), 145–160.

[30] Wanjawa, B. W., and Muchemi, L. ANN Model to Predict Stock Prices at Stock Exchange Markets. *arXiv preprint arXiv:1502.06434* (2014).

[31] Zhang, Y. Transaction Cost Function Minimization Using Simulated Annealing and Smoothing.