The Ripple Equation: Predicting Stock Prices Using Financial News and Graph Models

by

Christopher Dick

Supervised by

Dr Adam Chester

Associate Professor, Department of Computer Science

for the degree of

MSc Computer Science

Department of Computer Science

University of Warwick

September 2019

Abstract

Attempts to predict financial markets have existed for as long as markets have themselves. For many, the appeal is monetary: as the price that financial markets assign to each asset (typically a company stock, a bond or a commodity) changes continuously. If someone could accurately predict how these prices were to change then they could simply buy the assets they expect to rise in value and sell the assets they believe will fall in value. This would unlock a huge amount of profit for the predictor. For others, such as policy makers or business leaders, accurate predictions of financial markets help to reduce uncertainty in decision making and allow for informed actions to be made.

This paper attacks the topic of financial markets prediction by looking through a computer science lens. It takes a novel approach by combining a number of different techniques for data analysis that are common in this academic field. Independently, these topics have been previously applied to the prediction of financial markets but never before have they been combined in such a way. The topics explored include Natural Language Processing (NLP), Graph Theory and Time Series Analysis. The scope of the paper includes a novel approach to modelling the price movements of the stock market. The model chosen is a graph. This market graph is composed of entities as nodes, where one entity is connected to a second entity if the former influences the latter. This description is purposefully vague as it allows the paper to experiment with different definitions of "entities" and "influence". The price changes within this market graph are modelled using the Ripple Equation; a novel equation that this paper presents for the first time.

Additionally the paper includes an NLP price prediction to help assist the graphical model with its prediction. Despite being two pieces of the project, the market graph and the NLP price prediction are intimately connected. As a model of the stock market, the graph informs the NLP price prediction and conversely the information retrieved from the financial news is used to the build the graph structure. Analysis of the results obtained are presented towards the end of the paper.

Keywords: Graph Theory, Natural Language Processing, NLP, Sentiment Analysis, Time Series Analysis, Data Mining, Prediction, Machine Learning, Correlation, Statistics, Finance, Financial Markets, Stocks, Stock Markets

Declarations

I hereby declare that all work contained in this project is my own. While the topic of this project is financial markets trading, none of this should be taken as investment advice.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

A key function of a market is to provide a common venue for buyers and sellers to exchange their goods. As the number of buyers, sellers and goods increases financial markets take on a much more important (and arguably profound) role: to assign each good a price. The price at which a good exchanges hands from the seller to the buyer is driven by a large number of factors, all of which can be understood to affect either the demand or the supply for that good. Markets are excellent tools for resource allocation, specifically for signalling to producers which products are in demand and which products are over-supplied. Processing the vast about of information on each good, deciding how the information effects each of these factors and arriving at a price at which the buyer and seller can both agree on is no small task.

Take, as a non-financial example, a pencil. There are many complex steps involved: the wood must be harvested and transported to the factory; the lead must also be produced and transported to the factory; the materials must be combined and the pencil must be manufactured; the final product needs to be transported to the retail store to be marketed and sold. Each of these steps requires a huge amount of expertise, indeed it conceivable that no single living human fully understands each step involved. Yet the market has no trouble finding the value for this pencil (or even the value of publicly traded stocks that sell pencils, a much more complex problem). In effect the market is valuing how expensive each of the steps involved in production cost and balancing these costs with an estimation of how much the customer is willing to spend. The market, through the mechanisms of supply and demand, is a distributed computation machine, designed to process information and efficiently assign a price to each good being sold. This is true for all types of market: financial or otherwise.

When a market participant (be it an investor, a business, a government entity or an observer) engages in the activity of predicting the future price of an asset, what do they need to do? There are a number of approaches that can be categorised in many ways. One common distinction is fundamental analysis versus technical analysis: the former looks to understand all the components of the asset's price and derive its fair value while the latter looks for historical patterns and supposes that these patterns may repeat themselves. Computational approaches to predicting financial markets have tended to focus on technical analysis, as pattern recognition is a task that comes naturally to statistics and machine learning. This is discussed in the literature review in Chapter 2. In contrast, this paper will focus on fundamental analysis.

Another distinction worth making between typical approaches to financial markets prediction is whether practitioners chose to value an asset from the ground up or not. The ground up approach decomposes a price into the numerous components (such as revenue and costs for a stock, or default rates and interest rates for a bond) and builds a fair value of the asset. A fundamental investor taking this approach can compare their valuation to the current market price and to historical levels to decide if they believe the asset is undervalued or overvalued. In contrast, a practitioner may make the assumption that the current market price takes into account all available information and instead attempt to predict further price movements. This is the approach that we will take in this paper.

How do you go about predicting further price movements of an asset? An approach is to list the factors that influence an asset's price and build a map of the mechanism that causes these factors to change an asset's price. The practitioner must then estimate the probability that a particular factor in this price mechanism will change and then estimate the effect this change will have on your asset. An example may be to value fictional company Goodie's Orange Juice. Goodie's main cost is the price of importing oranges, which it buys from Spain at a price of 50p for enough oranges to make 1 litre of juice. Additional costs for this litre add up to another 50p and Goodie's sells it's juice for £1.20 per litre: giving a 20p profit per litre. Now suppose new trade tariffs with the EU are to be imposed with a 50% chance and if they are imposed they will push the equivalent price of oranges to 60p, causing profit per litre to fall to 10p. After making these calculations you then may estimate that this fall in profit will reduce Goodie's share price by 10%. Incorporating the

50% probability of tariffs leaves you with an expected share price fall of 5%.

In this example, the factor is the trade policy (ie the new tariff on oranges) and the mechanism is the series steps that link the change in trade policy to the change in stock price. The longer the chain of logic required to get from the cause of the shock to the resulting price movement, the more room there is for uncertainty about the prediction. There are three types of uncertainty here:

1. **Factors:** Uncertainty that is priced into the factor prediction (eg tariffs will be introduced with a probability of 50%).
2. **Within Mechanism:** Uncertainty about each stage of the mechanism (Will tariffs really push the price up by 10p? Will a 50% fall in profit really only affect the stock price by 10%? Will consumers perhaps be willing to pay 10p more for their juice?).
3. **Mechanism Construction:** Uncertainty about the mechanism itself (What if Goodie's starts to import oranges from Morocco for 55p?).

In this paper we chose to model the factors that influence the price of a stock by considering the relationships between the stocks in the market.

1. **Factors:** The movement of stocks prices are considered factors that influence the price of other stocks.
2. **Mechanism:** We model the mechanism that connects these stock prices as a graph. This market graph is composed of entities as nodes, where entities are connected if they influence each other.

At this stage, the description is purposefully vague as it allows experimentation with different definitions of "entities" and "influence".

This paper lies at the intersection of numerous topics in computer science including Natural Language Processing (NLP), Graph Theory and Data Science (specifically, Time Series Analysis). While the relationships between stocks are modelled explicitly, any other factors that could influence a stock's price is assumed to be exogenous. We experiment with various NLP techniques to draw signals from financial news that these exogenous factors have changed, but we do not model these exogenous factors explicitly. We then feed these signals into the market graph, model the movement of price changes around the graph and make stock price predictions.

The market graph and the NLP price prediction are intimately connected. The NLP price prediction detects signals from financial news that indicate a stock's price may move. These signals are then combined with the Market Graph, a model of the stock market, to decide which other stocks may also move. In this way, the NLP price prediction depends on the Market Graph. Conversely, the structure of the Market Graph is configured using the information retrieved from the financial news. So the structure of the Market Graph depends on the NLP Trading Strategy.

This paper has a clear goal: to combine numerous topics in Computer Science to form a novel way of modelling the stock market and making price predictions. The motivation of this paper is to understand the mechanics of the stock market using tools from computer science. It is evaluated by considering the directional accuracy of the predictions.

Chapter 2

Literature Review

This chapter reviews the existing literature in order to form a context. It begins with the classic economics papers that are crucial to the topic of trading, following by an introduction to common techniques used in algorithmic trading. Next is a discussion key the key papers in recent years that relate to machine learning applied to algorithmic trading, including the topic of neural networks. This chapter also briefly reviews the academic work on applying both Natural Language Processing and Graph Theory to financial markets trading.

It is important to note that authors who are successful in building algorithmic trading models have little incentive to publish. In other domains it is possible to find patents for the authors' valuable discoveries. Not only are patents unavailable for financial trading strategies but once a market trading strategy is published, it's effectiveness disappears. This is comes indirectly from Modigliani and Miller's 1958 paper [1] that stating that "as investors exploit these arbitrage opportunities, the value of the overpriced shares will fall and that of the underpriced shares will rise, thereby tending to eliminate the discrepancy between the market values of the firms." Any opportunity found is therefore limited, and so any group that builds a successful trading model is heavily incentivised to use their model in secret in order to keep these limited profits to themselves.

2.1 General Trading

Economics, the study of how nations allocate resources to satisfy the needs of their citizens [2], treats the topic of financial markets as a key *resource allocation* tool for capitalist economies. Before exploring common methods in financial markets prediction, it's important to review key aspects of financial economics. Euguene Fama introduced the term "efficient markets" in his seminal paper published in 1969 [3]. In this paper Fama describes markets that fully reflect all available information (i.e. are efficient) as the ultimate ideal for resource allocation so that businesses, governments and individuals can trust that current prices are fair. Fama concludes that "the evidence in support of the efficient markets model is extensive". Preceding this, Bachelier proposed that the prices of assets in financial markets follow Brownian motion [4] also known as a random walk. The implied consequence of both the Efficient Market Hypothesis (EMH) and the Random Walk Theory is that markets are inherently unpredictable. This is heavily debated by both academics and practitioners. In 2003 Malkiel published a review of the criticisms to EMH stating that in recent years some "economists emphasized psychological and behavioral elements of stock-price determination, and they came to believe that future stock prices are somewhat predictable". Practitioners posit that many do profit from financial markets prediction, the counter-argument being that these profits may be at random [5]. It goes without saying that most attempts to predict financial markets assume that the market is in some way predictable.

For reasons detailed above, the existing academic literature on predicting financial markets may appear thin in some areas but this isn't for lack of knowledge. Many results can be found outside of academia. For example, Quantopian is a website that crowdsources algorithmic trading strategies from the general public. The website encourages the use of Machine Learning (ML) and NLP techniques for financial markets analysis and even includes a NLP model for users to measure market sentiment that is pre-trained on the social media website StockTwits [6]. Another example is kaggle.com, a website that hosts data analysis challenges. This website recently posed the question "Can we use the content of news analytics to predict stock price performance?": they hosted a contest inviting users use NLP and ML techniques to predict the stock market [7].

Successful trading strategies are based on deriving strong signals that indicate the future direction of the market. We have briefly introduced the schools of technical analysis and fundamental analysis, but here we add a third: news-driven analysis. In many respects these approaches can be thought of as different philosophies that traders subscribe to; either entirely or in part. Most of the existing research papers focus on technical analysis: they attempt to derive trading signals from historical price movements. In the book titled "Technical analysis and stock market profits", Schabacker and Mack [8] define a "stock chart" as a "pictorial record of the trading history of any stock". They go on to say that they "are not interested... in the company behind the stock" because the stock chart has nothing to do with fundamental factors, is it merely a history of price movements.

In contrast, strategies that leverage fundamental analysis to predict price movements are interested in determining the intrinsic value a company by studying economic forecasts, the company's financial strength, quality of management and business opportunities. From here these strategies "determine the stock fair value... to identify investment opportunities" as described by Wafi et al [9].

The third approach is news-driven analysis. This includes strategies that may consider both historical price movements and the company's fundamentals, but focuses on how the stock price is affected by news events. In their paper "A news-driven model of the stock market" Gusev et al. state that stock "prices change when investors buy or sell securities and it is the flow of information that influences the opinions of investors, according to which they make investment decisions" [10]. The implied consequence of this is that if you can model the flow of information, you can model the price of a stock.

2.2 Machine Learning for Trading

In comparison to the popularity of machine learning learning as a whole, the application of machine learning to algorithmic trading relatively uncommon. As an indication, a Google Scholar search for "Deep Learning" and "Deep Learning Image Recognition" yield around 4 million and 2 million results respectively. In comparison, "Deep Learning Algorithmic Trading" yields 20 thousand results [11], most of which are false positives that do not focus on the topic of algorithmic trading.

There could be a number of reasons for this. Firstly, there are a number of challenges that financial trading poses to machine learning in contrast to other commonly studied domains such as gameplay or robotics. Trading can be thought of as actions between agents: Huang [12] notes "the agent has the potential to alter the financial market if powerful enough", and so for analysis to be of practical use one cannot assume the market is fully exogenous. Moreover Huang also discusses how it is impossible to have a complete view of the market. No matter how large the data set, there may other factors that influence price movement.

Secondly, there is little to build on. In Li's 2017 review of Deep Reinforcement Learning [13], the author notes that despite it being a "natural solution to some finance and economics problems" there has been little development. Li offers the idea that this may be the case because "it is nontrivial for finance and economics academia to accept blackbox methods like neural networks".

A third reason is that there is thought that there might be a trade-off between trading strategy complexity and trading strategy performance. Trader and author Ernest Chan speaks to this in his 2009 book "Quantitative Trading": "we can characterize AI as trying to fit past data points into a function with many, many parameters... With many parameters we can for sure capture small patterns that no human can see. But do these patterns persist? Or are they random noises that will never replay again?" [14].

Approaches to machine learning in algorithmic trading can be categorised as follows:

1. Sample price data from a number of financial products, make a number of assumptions about the independence and randomness of that data, and then fit a Deep Learning model to it. Yao and Tan [15] take this approach, as does Huang [12].

2. Produce artificial time series data to represent price movements of a financial product. For example, Ritter [16] produces data with an exploitable pattern built-in by design and the task is to see if a neural network can be built that detects this pattern. With this approach it is also possible to vary the complexity of the data (or conversely decrease the strength of the pattern) to see if a neural network can still detect the opportunity.

3. Do something in between 1 and 2. Ganesh and Rakheja [17] sample real market data where they know a pattern does exist: they highlight the existence of price reversion to some mean that changes over time. The authors then proceed to build a deep learning model in the hope that their model will uncover this pattern. Sewell [18] takes this approach and speaks to the importance of choosing a model before analysis begins: "Certainly, this thesis recommends using a data-driven approach by employing machine learning. Using both theory and domain knowledge, one must, a priori, select a prediction technique".

It is worth emphasising that this variety of approaches is one way that machine learning applied to trading differs from machine learning applied to other domains, such as gaming. Mnih et al. [19] were novel in that they created a neural network that could solve Atari games without encoding any rules of the games. Yet they knew that the games were solvable, or at least they knew there did exist strategies that were high performing. The authors wanted to see if the neural network could discover such strategies (or even find new ones). In contrast, with financial markets it is usually not known if patterns even exist and so any application of neural networks has an additional hurdle to cover.

The potentially infinite number of factors that affect a market price means a number of assumptions may be made when modelling:

1. **Technical analysis only.** This includes Yao and Tan [15] as well as Ganesh and Rakheja [17].
2. **Include fundamental analysis.** Huang [12] uses technical approaches but suggests their work would be improved by including macroeconomic data such as news or economic statistics.
3. **Signal data.** There have been a number of attempts to predict financial markets using non-financial information such as social media data [20] and Google search queries [21]. The choice of data source is inspired by more than availability alone. The question of what financial data to use is a function of the author's choice of assets, time period and time frequency. Stocks, for example, are heavily influenced by fundamental data. Sewell [18] highlights the importance of domain expertise but only a select few authors chose to include it. Gudelek et al. [22] do not incorporate domain knowledge stating that "extracting features from fundamental analysis might require different machine learning approaches e.g. natural language processing".
4. **Market frictions**. This includes transaction fees and slippage (how much the market moves before it is possible to execute the trade). These frictions differ greatly over time, by product and with liquidity (how easy it is to buy or sell a given product at a given time). Therefore it is common to simplify these frictions as either a static spread that is subtracted from any profit [12] or by providing a threshold by which the stock price must move in order to constitute a significant movement [17]. Huang [12] goes on to experiment with different spread assumptions to show that spread is typically inversely correlated with performance. At some level of spread Huang shows it is impossible to find a profitable strategy.
5. **Price data.** The availability of price data is naturally a concern for researchers in this field. Most stock and bond data is owned by the exchanges and so is expensive to get in the large quantities required for neural networks. Because of this, a number of papers use publicly available FX data such as Huang [12] or Yao and Tan [15]. In the cases where authors have chosen to create their own data that simulates price movements, the data is clearly free [16]. Ganesh and Rakheja [17] looked at high frequency trading and therefore required tick data trade by trade, not summarised over a period of time. To source this they partnered with a financial firm who had access to the data, WealthNet Advisors. Gudelek et al. [22] used data from Exchange Traded Funds (ETFs) that is publicly available from Google Finance.

Analysing multiple assets at once allows for cross-market signals to be extracted. Huang [12] employs this technique by using data from 12 FX pairs together, whereas Yao and Tan [15] chose to analyse each of their FX pairs separately. Both Huang [12] and Ganesh and Rakheja [17] emphasis the importance of "online" learning - allowing the neural network to continue to learn as it makes predictions, beyond just the training data.

2.3 Neural Networks For Trading

Although this paper does not make use of neural networks, as an important area of machine learning it is worth covering the literature on this type of model.

The details of the neural networks used for trading varies widely. Older papers on the topic, such as the one by Yao and Tan [15] published in 2000, use relatively basic forms of neural networks. In this paper the authors do not provide much detail on their model but highlight that they use only one hidden layer (in addition to the input and output layers). The authors vary the number of neurons in the hidden layer to show how this affects the results. For the error function, Yao and Tan use Normalised Mean Squared Error. This was typical at the time, for example it was used in the Santa Fe Time Series competition in the late 1990s [23]. In contrast Ganesh and Rakheja's paper [17], published in 2018, used a much more modern Categorical Cross entropy error function.

Recently, it has been common for authors to use different types of recurrent neural network in order to incorporate some type of memory into the network. This is intuitive, as financial market prices are likely a function of both their recent past, their distant past and also some external factors. To incorporate this memory, most authors use some variation of a multi-layer perceptron (MLP) model. For example Ganesh and Rakheja [17] use three MLP models, with a ReLU activation function for the hidden layers and Softmax for last layer. Gudelek et al. [22] use a convolutional neural network (CNN), a form of MLP model, that is usually applied to image recognition problems. For this innovation, the authors take financial data and reframe it as an image input into their CNN.

Both Huang [12] and Ritter [16] assume their financial markets data follows a Markov decision process, that is that any “action decision must not depend on whole history of states” [16]. To solve these Markov decision problems, both authors apply a Q-learning recurrent neural network. Ritter derives his own reward function to train his model and implements an ε-greedy policy to ensure that every action is attempted many times. Huang [12] takes a different approach here, purposely avoiding an ε-greedy policy. He notes that random experimentation would incur transaction costs in a practical setting, and therefore is a not a good model to use when training the neural network. To combat this, Huang pioneers an “action augmentation” technique that provides the author’s model with feedback at every step of the process (based on whether that action immediately paid off or not) instead of feedback solely at the end of every iteration. While intuitively, this feedback may be misleading as a good action may not always pay off immediately, Huang shows through comparison to an ε-greedy policy that action augmentation provides an additional 6.4% annual return on average.

Worth highlighting is the model choice that Sewell [18] made. The author implements a Q-learning neural network but also runs experiments with the following: support vector machines, Bayes point machine, Fisher Kernel and Difference of Convex (DC) functions.

Unsurprisingly, the majority of published research on the topic of neural networks in trading report positive results. That is, each paper reports that their model achieved a level of accuracy that corresponds to market-beating returns.

For those authors who took the approach of creating artificial data with a known opportunity, such as Ritter [16], the reported results are naturally exceptional. This is almost by design though and cannot be directly compared to other papers. For the papers that using real market data, typical accuracy is well above 50%. For example Gudelek [22] states that their model “can predict the next days prices with 72% accuracy and end up with 5:1 of our initial capital, taking realistic values of transaction costs into account”. Almost all papers present model results as a range where one variable is varied to show the model’s dependence. For example [17] gives accuracy as a function of participant percentage (how confident one requires the model to be before making a trade) and actually shows that model accuracy falls as the model makes more trades. Yao and Tan [15] show model returns with a varied number of neurons in the hidden layer, as well as in comparison to classical algorithmic trading models. Huang [12] varies the transaction costs built into the model and calculates return with ε-greedy policy to show that action augmentation is a preferable model feature.

As with all statistical models, overfitting is a concern. Most papers comment on how they have acted to reduce overfitting however without independent model testing it is impossible to be sure. Because of this, comparing papers via their results is not as revealing as via their assumptions. Put differently, given that many papers report results in a similar range, those papers with stronger assumptions are the papers who’s results are most impressive.

2.4 NLP for Trading

There is some academic literature on algorithmic trading, and even on leveraging NLP to build trading strategies.

The NLP techniques used by the existing literature on trading are relatively standard. The most common tool used is a pre-trained sentiment classifier. Papers that use this method include “Twitter mood predicts the stock market” by Bollen et al. [24], “Twitter as a tool for forecasting stock market movements” by Nisar and Yeung [25], “A news-driven model of the stock market” by Schumaker and Chen [26] and Oh and Sheng’s analysis of the financial social media site StockTwits [27]. Other methods include bag of words [27] [26], noun phrases, named entity recognition and phrasal aggregation [26].

The results achieved by papers in the related literature are varied. Unsurprisingly every paper obtains positive results but there are degrees to their success. As alluded to, most studies have evaluated their performance with accuracy scores. Bollen et al. [24] obtain a 86.7% directional accuracy with their best performing model. Oh and Sheng [27] achieve an F score of 0.853. Nisar and Yeung [25] measure the correlation between sentiment and stock market direction, achieving an R value of 0.923. For this project I will target maximising annualised returns on our trading strategy. Some studies do target such returns, such as Schumaker and Chen [26] who achieved 2.06% return. This is a lower number than we would deem to be successful. Preis et al have published a paper on “Quantifying trading behavior in financial markets using Google Trends” [21] where they achieve an annualised return of 23% over a period of 7 years. However this return is achieved when evaluating their trading strategy on the training data and so is unrealistic.

2.5 Graph Theory for Trading

This paper is by no means the first to apply graph theory to financial markets, indeed there are numerous textbooks that make the connection between graph theory, network theory and financial markets. An example is Easley and Kleinberg's textbook titled "Networks, Crowds, and Markets: Reasoning about a Highly Connected World" [28].

There are also instances in the academic literature where graph theory has been applied to the stock market. Abrams et al. published "Analysis of Equity Markets: A Graph Theory Approach" [29] where they constructed a graph that represented the stock market and applied basic graph analysis techniques including Spearman-Rank correlation, degree distribution and identified sectors within the graph. Konig and Battiston provide a step-by-step walkthrough of the basics of the topic in their paper "From Graph Theory to Models of Economic Networks" [30] where they state "The use of methods from graph theory has allowed economic network theory to improve our understanding of those economic phenomena in which the embeddedness of individuals in their social inter-relations cannot be neglected." There are also a number of undergraduate and graduate level papers on the topic, such as "The Market Graph" by Budai and Jallo [31] as well as "Network analysis of the stock market" by Sun et al. [32]. Both of these examples analyse the characteristics, structure and dynamics of the stock market when modelled as a graph.

2.6 Our Approach

In contrast to the work on machine learning in trading and NLP in trading, for the literature on graph theory in trading each author stops short of using their market model for predictive purposes. This paper focuses on financial markets prediction using graphical modelling methods. Through the introduction of the Ripple Equation, this project builds on the work that has been done to model the market as a graph and combines it with the work on NLP driven trading strategies. In this way, this paper provides a novel contribution and advances the state of the art in this area.

Chapter 3

Project Management

The project management of this paper is presented in this chapter with the aim of describing how the project has evolved since it's inception in its earliest form in January 2019. The project began as an attempt solely to apply Natural Language Processing to stock market price prediction. As data was collected and the literature was reviewed, the project's scope expanded to include a strong graph theory and modelling component. At this point the necessary research was carried out to ensure that the project was moving in an informed direction. The resources used are covered in the literature review but textbooks include Easley and Kleinberg's "Networks, Crowds, and Markets: Reasoning about a Highly Connected World" [28] as well as "A First Course in Network Theory" by Estrada and Knight. By the time of the project presentation in May, an outline of the objectives of the project had developed. Figure 3.1 is the original Gant chart shown in that presentation, which laid out target dates for each section of the project development.

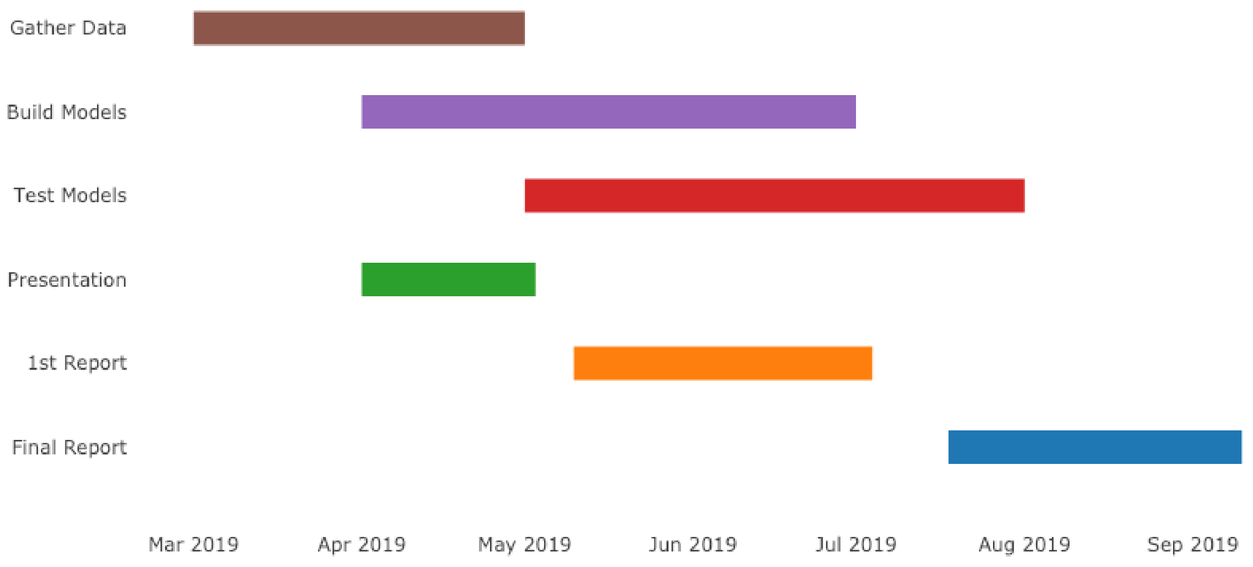


Figure 3.1: The initial project plan as presented on 10th May 2019

The first stage of the initial plan was gathering data. This took less time than anticipated as price data was readily available. Financial news data was not as available and so the scope was narrowed slightly. The building and testing of models began on time as planned, but continued right up until the end of the final report. While this contradicts the initial plan, it is in line with the iterative approach that emphasises the completion of each iteration of the model (including write up) before moving on to the next. All deadlines for the Presentation, Interim and Final Reports were met on time. While most of the objectives of the project remain unchanged since this plan was proposed, it has since grown into an ambitious attempt to define a new model through which to understand the movement of price changes. The central focus of the paper is the Ripple Equation and its implementation as the $Predict()$ function.

The first unforeseen problem was the lack of readily available financial news data. This problem was significant as the whole project hinged on having a large quantity of high quality news data from which to derive insights. All such data sources were extremely expensive or not suitable for the paper's purposes (for example, they did not contain financial information). The solution to this problem was to scrape data directly from news websites to form a unique data set containing exactly the information required for the paper.

3.1 Iterative Approach

This paper takes layered approach to the construction of the trading model. There are numerous connected pieces required for a full functioning model. Instead of completing one piece at a time to full satisfaction, a working but basic version of each stage of the project is first completed. Each stage is then updated for a second iteration of the model. In total there are three iterations of the end-to-end working model. Each model works independently and together they demonstrate completeness of the project overall.

The approach of presenting numerous models that increase in complexity is not seen in any of the academic papers reviewed so far. The reason that this paper took such a unique approach is two-fold. Firstly, approaching the project in iterations minimised the risk of running out of time. Once the first iteration was complete, a fully working model was obtained. Had time run out at this point the paper would still have been written. As it is, there was time for multiple iterations. In this paper each of the three iterations are presented as to compare and contrast the different approaches. Furthermore it offers an insight into the thought process, revealing how the project began with strong assumptions that could be loosened as the project progressed.

Outside of the papers reviewed so far, there are many examples of iterative processes for the development of software. These examples exist in both literature and in industry. Craig Larman, author of "Agile and Iterative Development: A Manager's Guide" [33], emphasises that "most software is not a predictable or mass manufacturing problem. Software development is new product development". He goes on to define each iteration in such processes as a "self-contained mini-project composed of activities such as requirements analysis, design, programming, and test". This is precisely the project management approach employed for this paper.

3.2 Coordination With Supervisor

Regular meetings with the project supervisor were held consistently throughout the life of the project. In the ideation stages in January 2019, brainstorm sessions were held with the supervisor. This inspired the initial project proposal of predicting the stock market with NLP, but also was crucial in the development of the Ripple Equation and the graph theory element of the project. Between April and June, meetings were held bi-weekly with the frequency increasing to every week in July and August. This high frequency of meetings was a crucial part of the iterative approach to this project allowing independent work to be held accountable, ideas to flow freely and the project to develop in multiple directions. The improvements made in the second and third models are indirect results of this meeting cadence.

3.3 Technical Decisions

Decisions regarding technical decisions have been quite standard. The work was developed in a Jupyter Notebook running Python 3.7. This allowed for a flexible and iterative work environment that permitted experimentation. Experimentation is key for this project as it combines numerous topics within the field of Computer Science in such novel way. The project leverages powerful python packages such as Pandas (for data management), SpaCy and TextBlob (for NLP), NetworkX (for graphs) and Plotly (for visualisation). For data collection data was scraped using the Requests and Beautiful Soup models. During development, the work is synced to a private GitHub repository that holds a history of code versions. The benefits of using GitHub were two-fold. Firstly, it provided a secure storage for the development and the data required for the project. This storage included versioning that permitted progress to be rolled back to an earlier stage if required. On a number of occasions this rollback option was used when attempting to develop a later version of the model. The second benefit of GitHub was that it permitted work to be carried out seamlessly from multiple machines. Both the department computers and personal laptops were used, and GitHub made this possible. The three model iterations discussed in this paper will be made available on a public GitHub page [34]. Partial code and pseudocode is presented throughout the paper when it is necessary to illustrate the implementation. In addition to this partial code and in addition to the full code on GitHub, the paper is submitted with a ZIP file containing three folders titled first, second and third that contain the full data and working python scripts for each of the three models respectively.

Chapter 4

Methodology

As alluded to in Chapter 3, this paper presents the design and development of three stock price models. Each model is an attempt to describe and predict market prices. The paper focuses on the third approach for trading strategies that is described in the literature review: news-driven analysis. Despite the claims from Gusev et al. [10] that stock prices are function of information flow, there are not that many machine learning research papers on trading that use news as an input. It is the combination of modelling the stock market graphically and using financial news for insights that this paper focuses on. Each one of the three model iterations requires the following sections to be completed:

1. **Market Graph:** defining an influence measure, building an adjacency matrix, constructing the graph.
2. **Price Data:** defining a stock universe, retrieving price data and collecting news corpora.
3. **News Data:** preprocessing the news, identifying signals and measuring the impact of the signals.
4. **Prediction:** take the news signals, ripple the impact across the graph and output predictions.
5. **Trading Strategy and Evaluation:** measuring the performance of the model, either directly or via a trading strategy.

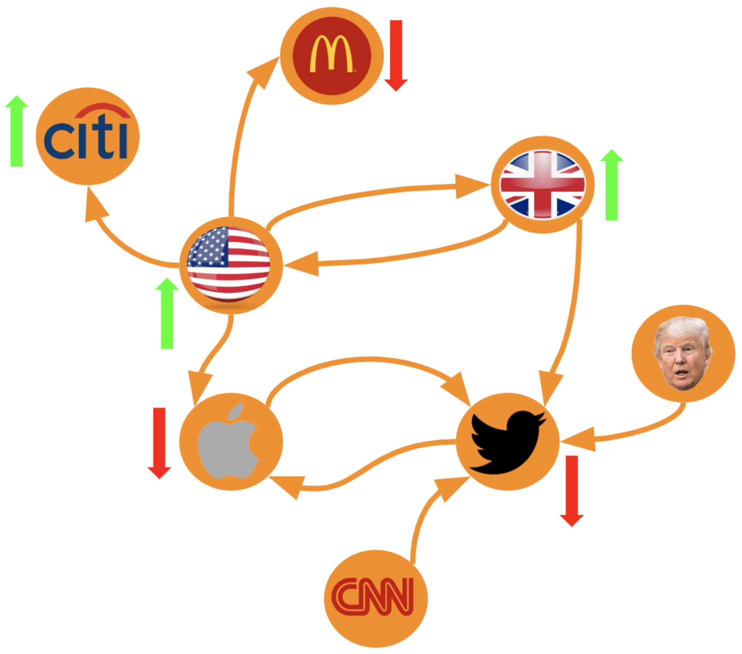


Figure 4.1: A visualisation of the Market Graph

The Market Graph models entities in the stock market as nodes that are connected by edges. Edges represent influence of one entity over another. Price changes are said to ripple across the graph.

The general approach taken for each of the project's five components is outlined here. Figure 4.2 presents the structure of each model. At the centre of each model is the *Predict*() as it is here that the Ripple Equation is implemented. This Equation is detailed later in the chapter. News data, price data and a graph structure form inputs to the *Predict*() function. Additionally, the impact of the news data needs to be measured and fed into the *Predict*() function. The function takes these inputs and calculates which stocks are predicted to have moved through the calculation of the Ripple Equation, so called because it computes how price movements "ripple" through the market graph. These predictions can be measured directly or fed into a trading strategy, which is then evaluated.

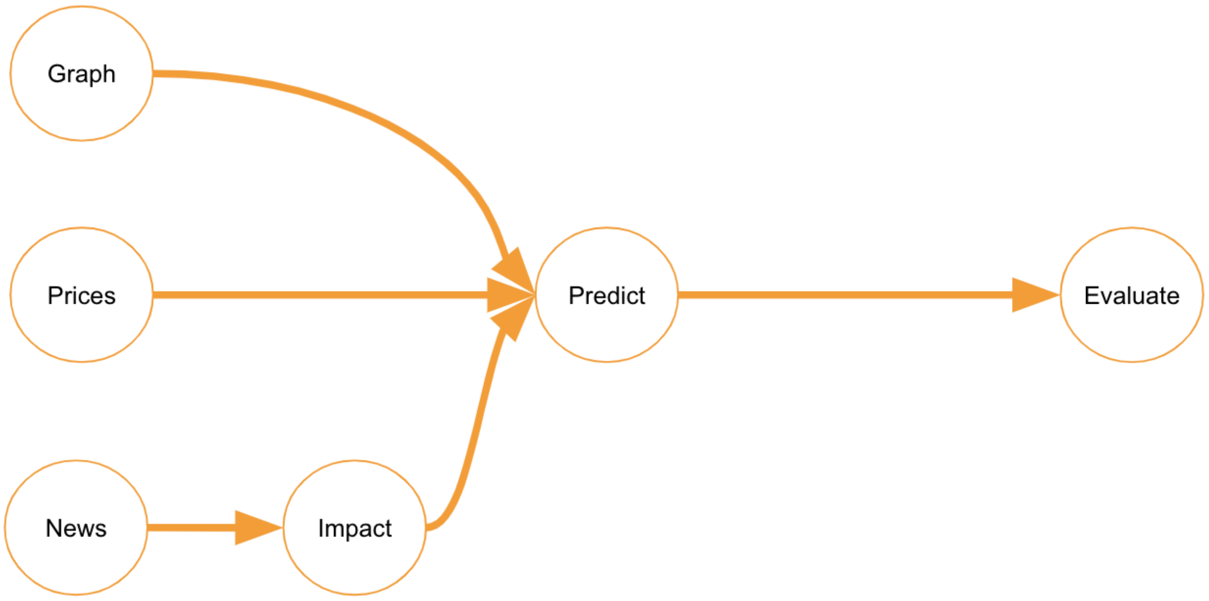


Figure 4.2: A diagram describing each model's structure

The rest of this chapter takes a look at what is required for each step in the general model. It does not go into the implementation of each of the three models, how they differ or their results - this is covered in Chapter 5.

4.1 Market Graph

In the Market Graph, the nodes represent entities in the market and the edges represent whether two nodes influence each other. When it comes to constructing the graph, there is some freedom in choosing which nodes are connected. The aim is to find measures of influence between entities that reflect real-life price movements and accurately model the stock market. Given a stock universe containing n stocks, denote by S_i and S_j two stocks different stocks (i.e. $i, j \in [1, n]$ and $i \neq j$). A measure for the influence of S_i over S_j is a measure that describes the extent to which a move in the price of S_i causes a move in the price of S_j. When looking for a good influence measure there are a couple of key considerations. Firstly we define a directed influence measure as one that is non-symmetric: the influence of S_i over S_j is not necessarily equal to the influence of S_j over S_i. An assumption is that a directed influence measure better models real-life price movements compared to to an undirected one. Secondly an ideal influence measure will capture *causation*: we want to model when S_i causes S_j to move, not just the likelihood that they both move in a given time period. This second consideration is clearly difficult to model but is something to aim towards when experimenting with influence measures. Here are a number of possible approaches:

1. $correlation(S_i, S_j)$ (an undirected measure)
2. $covariance(S_i, S_j)/variance(S_i)$ (a directed measure)
3. co-occurrence of the two stocks together in the news articles as a percentage of each stock's individual occurrence (directed)
4. other time series attribution techniques
5. other NLP relevancy techniques

6. techniques that combine time series analysis and NLP

Once an influence measure is decided on, the market graph is technically straight-forward to to produce, though the preprocessing required is more complex. Details are given in the next chapter.

4.2 Price Data

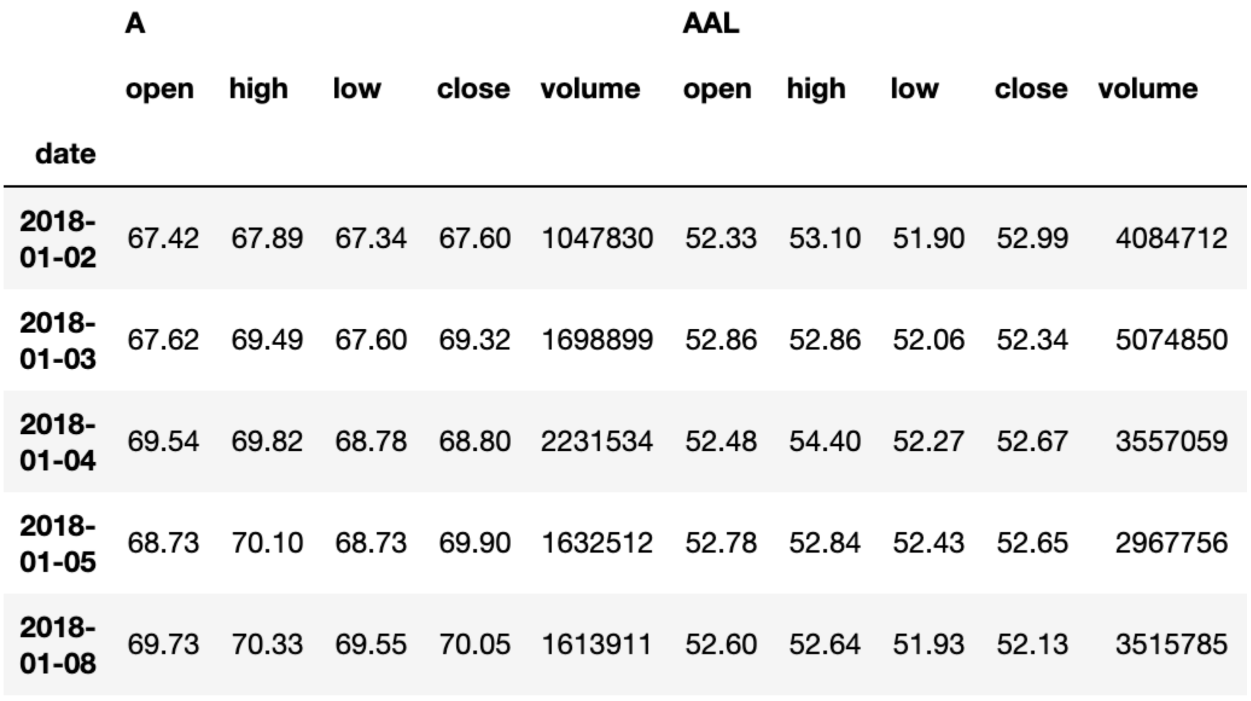
The models detailed in this paper work with time series data from single stocks (as opposed to baskets of stocks or indices). Each stock modelled is a member of the S&P500 index, the index regarded as the best measure of the largest stocks in the US [35]. The number of companies selected from this index ranges between 50 to 500 companies for different iterations of the model. Each stock is prominent and assumed to feature regularly in news articles. Such a large number of stocks provides an opportunity to show that the trading strategy is independent of the company being analysed. For all three models the stocks selected are subset of the S&P500 stock index.

Price data is sourced from the Investor Exchange (IEX) trading API [36]. This API is free for small amounts of data usage, but the quantity needed for this paper required a subscription plan costing $9 per month. Subscribers are given a private token to use when making API calls. A typical call for our purposes is as follows:

```
from iexfinance.stocks import get_historical_data
tf = get_historical_data(top_stocks, start=start_date, end=end_date, output_format="pandas", token
     =token)
tf.to_pickle("stocks.pickle")
```

The variable *top_stocks* is a list of tickers for which price data is to be retrieved. This list varied for each of the three models implemented, as discussed in the next chapter. Similarly, the date range required also varied from 3 months to 12 months as the model grew more complex and the testing more rigorous. The data from each API call was pickled and saved so that calls didn't need to be repeated. In addition to price data a similar call was used to retrieve meta-data on each stock such as market capitalisation and company industry.

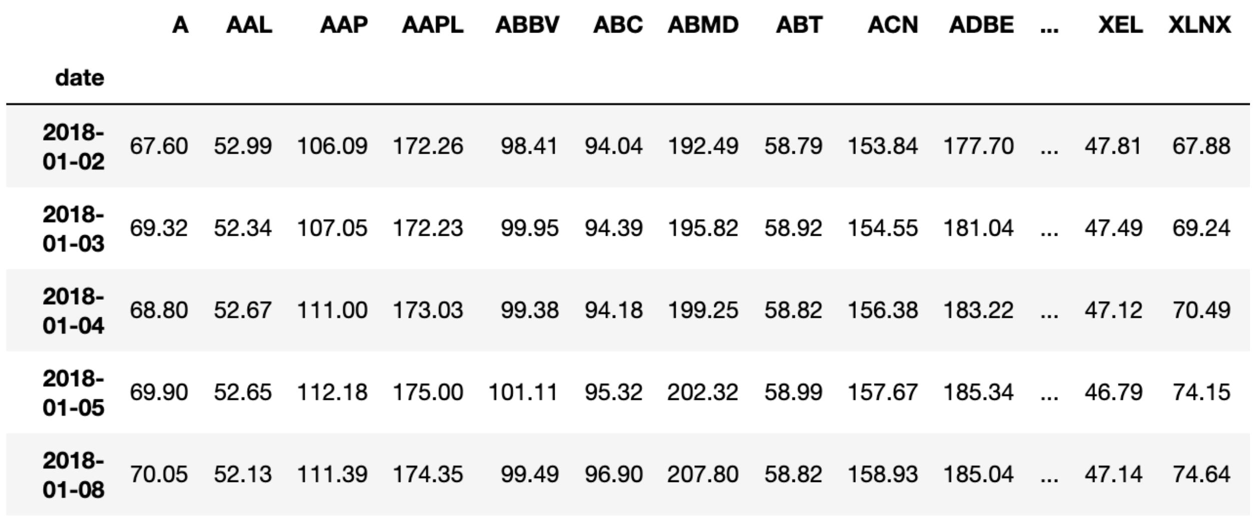
An example of the raw data retrieved in an API call is given in figure 4.3. For this paper only close of day data is required, so the other columns were dropped. The API call only allowed data on 100 stocks to be retrieved in one call, so for the full S&P500 multiple calls were made and the results concatenated.



	A					AAL				
	open	high	low	close	volume	open	high	low	close	volume
date										
2018-01-02	67.42	67.89	67.34	67.60	1047830	52.33	53.10	51.90	52.99	4084712
2018-01-03	67.62	69.49	67.60	69.32	1698899	52.86	52.86	52.06	52.34	5074850
2018-01-04	69.54	69.82	68.78	68.80	2231534	52.48	54.40	52.27	52.67	3557059
2018-01-05	68.73	70.10	68.73	69.90	1632512	52.78	52.84	52.43	52.65	2967756
2018-01-08	69.73	70.33	69.55	70.05	1613911	52.60	52.64	51.93	52.13	3515785

Figure 4.3: Sample price data before preprocessing

Figure 4.4 shows the price data after the preprocessing step, this is part of the full dataframe of price data used in the models.



	A	AAL	AAP	AAPL	ABBV	ABC	ABMD	ABT	ACN	ADBE	...	XEL	XLNX
date													
2018-01-02	67.60	52.99	106.09	172.26	98.41	94.04	192.49	58.79	153.84	177.70	...	47.81	67.88
2018-01-03	69.32	52.34	107.05	172.23	99.95	94.39	195.82	58.92	154.55	181.04	...	47.49	69.24
2018-01-04	68.80	52.67	111.00	173.03	99.38	94.18	199.25	58.82	156.38	183.22	...	47.12	70.49
2018-01-05	69.90	52.65	112.18	175.00	101.11	95.32	202.32	58.99	157.67	185.34	...	46.79	74.15
2018-01-08	70.05	52.13	111.39	174.35	99.49	96.90	207.80	58.82	158.93	185.04	...	47.14	74.64

Figure 4.4: A table containing sample price data

4.3 News Data

An original corpora of news articles was collected for this paper. The initial intention was to to collect three types of news corpora: long-term research reports, medium-term news articles and breaking news from twitter. Time restrictions meant that the focus had to be narrowed to just medium-term news. In total 38,000 financial news articles were collected from Reuters.com [37] for the period August 2018 to February 2019, including 12,332 articles published in 2018.

This corpora contains 12,332 articles relating to 2018, which is 33 articles per day on average. While initial models focus on the headlines of these news articles, the final models explore the body of the text. The corpora contains every financial markets-related news article published by the influential financial news firm, Reuters. Reuters is known as a trustworthy and impartial news source, famous for its Trust Principles that guide its reporters on avoiding bias [38]. The fact-checking website MediaBiasFactCheck.com that measures the credibility of news outlets says that Reuters scores ”Least Biased based on objective reporting and Very High for factual reporting due to proper sourcing of information with minimal bias” [39].

This is by no means a full collection of the news data published in 2018 and this paper does not claim that it is a representative sample. This context is important when we review the results in a later chapter. To collect the corpora, a script was written to crawl Reuters.com and retrieve the URLs for every news articles that could be found. It is important to note that some of the following code was adapted for use from a public Github repository that was used to scrape Boston Marathon results [40].

```
def pull_ids(pagenumber):
    ids_list = []
    url = “https://uk.reuters.com/news/archive/marketsNews?view=page&page=” + str(pagenumber)
        + “&pageSize=10”
    response = requests.get(url)
    page = str(BeautifulSoup(response.content))

    while True:
        url, n = getURL(page)
        page = page[n:]
        if url:
            urllist = url.split(“/”)
            if urllist[1] == ’article’:
                ids_list.append(url)
        else:
            break

    return  list (set( ids_list ))
```

Simply, this function can be set to run on the archive page of Reuters.com and will iterate through each page, grabbing the URLs and returning them as a list. The function *getURL*() is used here but its implementation is omitted. *getURL*() takes a page of HTML and returns the first URL found on that page, along with the position that the URL was found at. After generating a list of URLs from Reuters.com, the following function is used to scrape the articles. Again, some of the code was adapted for use from a public Github repository [40].

```
def pull_article (url):
    try:
        #make call to site
        response = requests.get(url)
        soup = BeautifulSoup(response.text, "lxml")
        page = str(BeautifulSoup(response.content))

        #pull article details
        article = soup.findAll("script", type="application/ld+json")[0]
        article = json.loads(clean_json( article .text))

        #pull article body
        paragraphs = soup.find("div", "StandardArticleBody_body")("p")
        if paragraphs == []:
            paragraphs = soup.find("div", "StandardArticleBody_body")("pre")
        body = [item.text for item in paragraphs]

        #save to file
        filename = str(datetime.now()) + ".json"
        article ["body"] = body
        with open("reuters/" + filename, "w+") as f:
            json.dump(article, f)
        time.sleep (0.1)
    except Exception as e:
        with open("error_log.txt", "a") as g:
            g.write(url + " " + str(e))
```

This function is relatively simple. It takes a URL as an input, calls that page and saves it down to a JSON. Sometimes the pages explored were in the wrong format and had to be investigated manually. In this circumstance the URL of the page throwing the error was saved to an error log file for manual inspection.

After the articles are retrieved we must apply modern NLP techniques to extract features from the text. We are looking for features that imply which stocks the articles will impact and what impact the article will have on those stocks. Techniques available include word embeddings, sentiment analysis, named entity recognition and topic segmentation as well as a number of preprocessing techniques such as lemmatisation and part-of-speech tagging. Details of the individual techniques used for each model are given in the next chapter. Roughly speaking these are the steps involved:

1. Read each previously unseen news article and extract the named entities mentioned.
2. Classify the impact of that article on the price of the stock mentioned.
3. Aggregate this impact over a set time period and feed it into the market graph.

4.4 Prediction

The most important part of each model in this project is the *Predict*() function. This function implements the Ripple Equation, which is crucial part of this paper and a novel contribution to the literature. In summary, the purpose of this function is to understand the signals extracted from the financial news and predict what impact this implies for the other stock prices in the defined stock universe. The *Predict*() function achieves this by exploiting the relationship between nodes as described by the Market Graph.

It is worth including a brief mention of the Page et al's 1999 paper titled "The PageRank citation ranking: Bringing order to the web" [41]. This paper described what was at the time a novel method of ranking websites on the internet through a recursive method that considers how many incoming links each website has. The paper says that "Intuitively, this can be thought of as modelling the behaviour of a random surfer" that clicks on links and executes a "random walk on the graph of the Web". In the PageRank algorithm there exists a vector R with an entry for each website that describes the importance of that website. The algorithm also contains a matrix A that describes the number of links between the websites. The key step in the PageRank algorithm is the update of vector R at timestep i given by:

$$R_{i+1} \leftarrow AR_i$$

The Ripple Equation is inspired by the elegant update found in the PageRank algorithm, though it doesn't claim to be as neat nor as aesthetic. The analogy is that while the PageRank algorithm models the movement of a random surfer across the internet in order to obtain the importance of each website, the Ripple Equation models the price ripples across a graph of the stock market in order to obtain price predictions for each stock. Where the PageRank algorithm updates the vector R representing the importance of each stock, the Ripple equation updates the vector p, the price of each stock. Where the PageRank has a static A for the number of links between each website, the Ripple equation has a static matrix R, describing the influence between stocks. This is where the similarities end, for the Ripple Equation requires an additional matrix D, that describes which stock prices have changed from one timestep to the next.

A formal definition is now presented. The Ripple Equation for n entities is given by:

$$p^{t+1} = diag(RD^t) \odot p^t$$

where

$$p^t := (p_1^t, ..., p_n^t)^T \text{ where } p_i^t \text{ is the price of entity } i \text{ at time } t$$

$$D_{ij}^t := \begin{cases} 1 & \text{if } i = j \\ \frac{p_i^t - p_i^{t-1}}{p_i^{t-1}} & \text{if } S_i \text{ has an edge leading to } S_j \\ 0 & \text{otherwise} \end{cases}$$

R is $n \times n$ matrix such that R_{ij} is influence of S_j over S_i for stocks S_i, S_j

$\odot$ denotes element-wise multiplication

The Ripple Equation is implemented in slightly different ways for each version of the model, but is always part of the $Predict()$ function. This function takes different inputs depending on which version of the model we are considering, and outputs price predictions for the following time period (ie the next day). Details of how the $Predict()$ function is implemented for each model are given in the next chapter. The Ripple Equation is a proprietary equation derived to calculate how a price change moves across the market graph; it is a novel contribution to the literature and inspired by the work of Page et al. on the Page Rank formula [41].

4.5 Trading Strategy and Evaluation

Evaluation of the models in this paper uses standard techniques such as directional accuracy. Additionally, in this domain, it is preferable to demonstrate which models are profitable and which are not. Time restrictions did not permit the use of techniques common in industry, such as backtesting, that can measure profitability of the model.

Instead we limit our evaluation to directional accuracy. Once a stock price change has been predicted, we measure how accurate this prediction was. We do this by simply comparing the predicted change to the actual change over that day and measuring directional accuracy.

An example trading strategy is as follows. For every prediction the model will return a market graph with the same nodes and edges but different price values. Each of these price values will be used to decide how much should be invested in each stock. If the model predicts that 20% of the stocks should rise in the next day by more than a given threshold then we buy these stocks with an arbitrary investment purse of \$1,000,000 split over each stock we want to buy. By observing the actual price changes we can calculate the profit/loss for the day. The ultimate measure of performance would be that the model is profitable over a period of multiple months.

The trading strategy described above (buying stocks that we predict will rise more than a given threshold) is just one example of a possible trading strategy that could be implemented. There is freedom here to implement a whole range of complex (or simple) strategies if time allowed, but it does not. Additionally it would have been powerful to introduce real-life market features such as bid-ask spreads or slippage, which is possible when using a trading strategy and profitability as the evaluation method. Details of proposed further work are outlined in section 6.2.

Chapter 5

Implementation and Results

In this chapter discusses the three models implemented for this project. Each of the models represents both an increase in complexity over the previous model but also an increase in how well the model represents a real-world trading strategy that a practitioner would use. Each section of this chapter presents one model, along with details of its implementation and its results.

5.1 First Model

The first model is the most basic of the three. with this model, the primary aim was to complete one model that was fully working from end to end. Because of this, many assumptions and simplifications were made for this model.

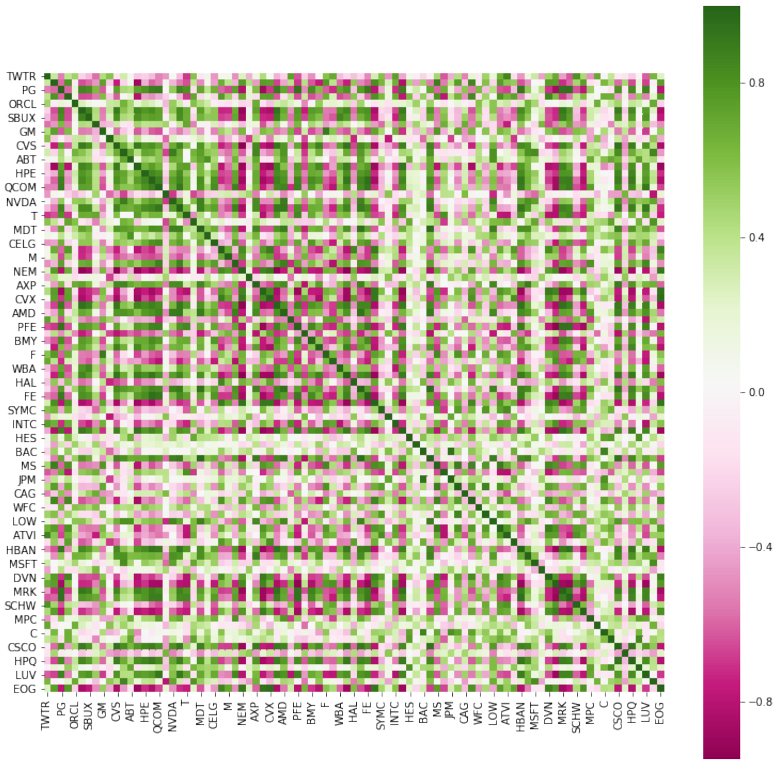


Figure 5.1: Heatmap of the correlations between the stocks used in the first model

Price Data

The first model made use of retrieved price data for 89 stocks for a period of 3 months from 1st June 2018 to 1st September 2018. These 89 stocks formed the stock universe for this model, each is a member of the S&P500. For the full list of stock tickers see the appendix. The price data is held in a dataframe similar to figure 4.4 from the last chapter. From this it is easy to measure the correlations between the stock prices.

For this model, the correlation was measured over a period of 30 days ending September 1st. A heatmap of these correlations is shown in figure 5.1. A detailed analysis of the heatmap is not presented here, but it is worth noting that a number of dark patches stand out in the diagram (representing stocks pairs that have a particularly high or a particularly negative correlation).

To further explore the correlations between the 89 stocks present in this model a number of analytical techniques are applied, beginning with the distribution of pairwise correlations across our 89 stocks. In total there are $89^2 = 7921$ correlations, but for the following charts we have removed the data points where correlation equals exactly 1: the correlations of a given stock with itself. Figure 5.2 shows a Probability Density Function plot of correlation versus number of pairs with that correlation.

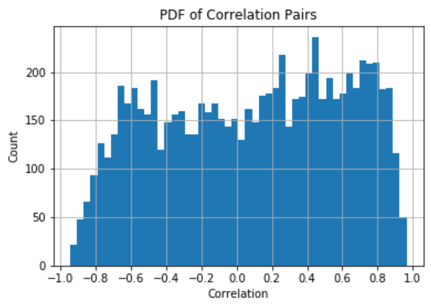


Figure 5.2: PDF of Correlations

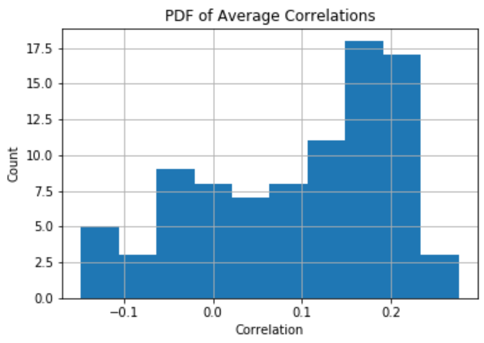


Figure 5.3: PDF of Average Correlations

This plot reveals that the correlations are reasonably uniformly distributed with peaks around -0.6 and 0.8. Summing instead all the correlations for a given stock and take an average, it is possible to see the distribution of average correlations, as shown in figure 5.3. This chart is not particularly insightful, so instead consider taking each stock and summing the absolute values of each of its correlations produces a number for each stock that crudely represents "how correlated it is to everything else". Figure 5.4 shows a chart of the probability density function of these absolute average correlations, and highlights that over 20 of the stocks are "highly correlated to everything else".

The PDF of Absolute Average correlations in figure 5.4 shows that there are are lot of stocks that have a reasonably high (positive or negative) correlation to a lot of other stocks: their absolute average correlation is around 0.6. Considering which entities/ nodes in our graph should have influence/ edges to other nodes, the question is raised of where to draw the line to distinguish which entities are connected and which are not.

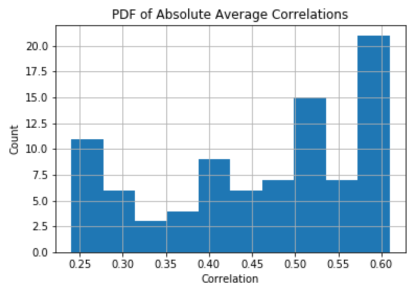


Figure 5.4: PDF of Absolute Average Correlations

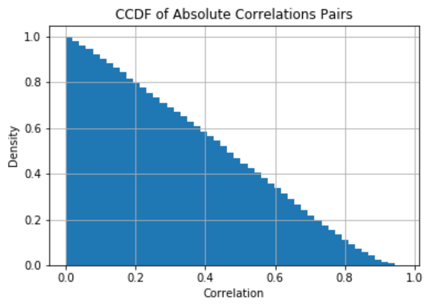


Figure 5.5: CCDF of Absolute Correlation Pairs

Market Graph

For the first model, this paper keeps the graph construction relatively simple. This means keeping the influence measure relatively simple too. For this model we chose an undirected influence measure, simple correlation as discussed in the previous section. The key question is where to draw the cutoff to distinguish which entities are connected by an edge and which are not. To answer this question consider the complementary cumulative distribution function of the absolute value of the correlation pairs, as seen in figure 5.5.

Figure 5.5 shows that around 10% of the correlation pairs have a correlation of less than -0.8 or above 0.8. This is the cutoff point is selected for the first model. To build the market graph the python module networkX is and a function called *build_graph*() is defined as follows:

```
def build_graph(date, thresh = 0.8, timeseries_data = tf, adjacency_func = weighted_a):
    idx              = timeseries_data.index.get_loc(pd.to_datetime(date))
    correlations     = timeseries_data.iloc[-1:-30].corr()
    adjacency_matrix = correlations.applymap(lambda value : adjacency_func(value, thresh))
    G                = nx.from_numpy_matrix(adjacency_matrix.values)
    G                = nx.relabel_nodes(G , dict(zip(G.nodes() , adjacency_matrix.index)))
    return G
```

This function took 3 inputs: the date on which the graph is to be build (date), the correlation cutoff threshold for the edge construction (thresh), the price data to use (timeseries_data) and the function that would be used to define the influence relationships between nodes (adjacency_func). These inputs are used to calculate the correlations between the stocks and compute an adjacency matrix. Finally the function outputs a graph built with NetworkX. The following function is applied to the timeseries data for the 89 stocks, alongside a look closer at the resulting graph structure.

```
def simple_a(x, thresh):
    return (abs(x) >= thresh) * 1 & (abs(x - 1.0) > 0.00001) * 1

G = build_graph(date ="2018-08-31", thresh = 0.8, timeseries_data = tf, adjacency_func = simple_a)
for c in nx.connected_components(G):
    print(c, len(c))

OUTPUT:
{'EOG', 'SYMC', 'HPQ', 'PYPL', 'GE', 'CMCSA', 'NVDA', 'VZ', 'CSCO', 'WMT', 'BK', 'COTY', 'FCX',
    'DAL', 'XOM', 'HBAN', 'SBUX', 'CNC', 'T', 'RF', 'CVX', 'SLB', 'TWTR', 'KEY', 'EA', 'HPE', '
   CTL', 'APC', 'LUV', 'BA', 'MO', 'FE', 'NEM', 'QCOM', 'MS', 'CVS', 'CELG', 'MDT', 'DVN', 'F', '
   GM', 'V', 'FISV', 'PG', 'BMY', 'LOW', 'EBAY', 'FB', 'AXP', 'MU', 'WDC', 'AAPL', 'TJX', 'AMD', '
   EXC', 'INTC', 'TGT', 'WBA', 'MRK', 'PFE', 'ATVI', 'SCHW', 'WU'} 63

{'JPM', 'BAC', 'C'} 3

{'COP', 'MRO'} 2
```

In this example there are 3 connected components with size larger than 1: one of size 63, one of size 3 and one of size 2. The rest of the stocks are disconnected from the graph. The adjacency_func called simple_a returns 1 if the correlation between two stocks is greater than the threshold (0.8) and returns 0 otherwise. The next section shows how the first model incorporates financial news. The section after will show how the predict function will take the impact of the financial news and ripple an impact across the graph we have built. Figure 5.6 is a visualisation of how a price impact would ripple across the graph.

News Data

The first model was built with only 10,000 articles so that the time required to develop each step of the process was reduced. Each article was preprocessed before using SpaCy's pre-trained Named Entity Extraction method to extract all entities with the tags "ORG", "GPE" or "NORP" (i.e. entities that related to an organisation). Each entity was then further cleaned to reduce the number of duplicates. The most 1000 commonly mentioned entities were extracted and every entity that was not a stock in the 89 stocks was manually removed. The tickers of those stocks were manually matched to the entities extracted from the corpora. The result is a dictionary where the keys are the ticker symbols for the 89 stocks and the corresponding values are the names of those stocks written in the way they are referred to in the news.

For this model we focused on the headline of each article, classifying its sentiment using TextBlob's built in sentiment classifier

```
TextBlob(headline, analyzer = PatternAnalyzer()).sentiment[0]}
```

This function outputs a score between -1 and 1, with -1 for text with negative sentiment and +1 for text with positive sentiment. The simplifying assumption was made that any article with a headline that scored

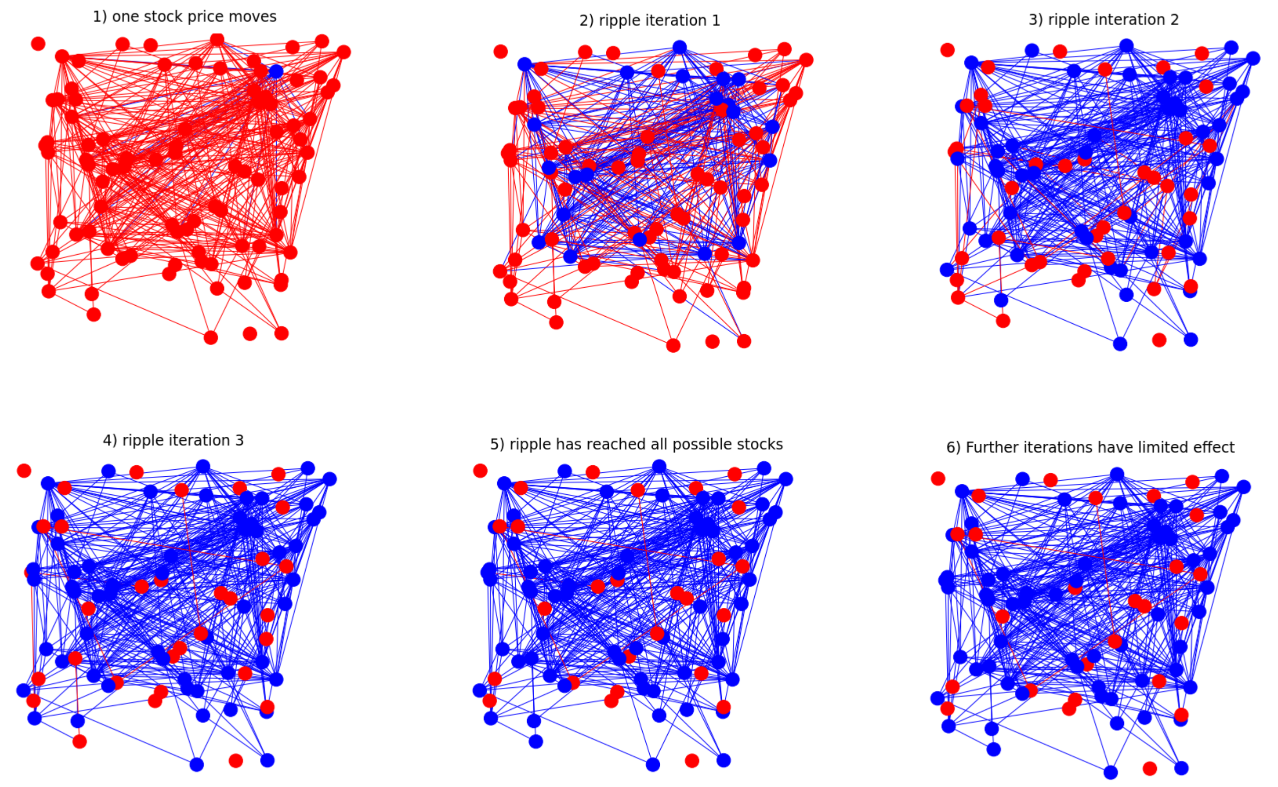


Figure 5.6: Visualisation of a price ripple

less than -0.1 had a negative (-1) impact on the price of any stock mentioned in that article. Similarly we assumed +0.1 or higher had a positive (+1) impact and the impact was negligible (0) otherwise.

News articles were aggregated over a set time period of 1 day. This is an assumption that can be relaxed in a later iteration of the project but feels natural as an initial step as the price data obtained is for a daily frequency. The resulting dataframe is 65 rows long, one row for each trading day in the three month period from the beginning of June 2018 to the beginning of September 2018. The first 89 columns of this dataframe contain price data for each stock. The final column contains a dictionary for every row (trading day) where the keys are the tickers that were mentioned in an article that day and the values are the impact that article had on the stock (+1, 0 or -1).

Prediction

The aim of the model was to take price and news information relating to one trading day, and predict what the prices will be on the next day. For the first model we built a relatively simple *Predict*() function that implemented the Ripple Equation from section 4.4. Before this was completed, there was a preliminary version of the *Predict*() function that we will talk about briefly. For the preliminary version of the *Predict*() function, inputs were predict_ticker, predict_date, orig_graph, moved_ticker, num_ripples. Crucially, this function requires that the user specifies which ticker the function should return a prediction for and which ticker has moved.

The function begins by defining the key variables required for the ripple function: the price vectors, the change matrix and the influence matrix. It then computes a pre-specified number of iterations (or ripples) and returns the predicted price for the pre-specified ticker. To make evaluation more straight-forward the function returns not only the predicted price, but also the original price and the actual price observed at the following timestep (i.e. the next day). This preliminary version of the *Predict*() function was used to test the implementation of the Ripple Equation, but the major drawback was that it didn't take the news data as an input and required that any user specified the exogenous stock moves.

We move now to the full *Predict*() function for the first model. The inputs to this function are similar to the preliminary function with the addition of an input of time series data. This time series data is produced by combining the price data and news data discussed in earlier sections of this chapter. The function *config_timeseries*() presented below is used to produce the time series data in the right format.

```
def config_timeseries(news_data, price_data, start_date, end_date, sentiment_func):
    mf = news_data.copy()
     preprocess_entities ()
    mf["date"] = pd.to_datetime(mf.dateCreated).apply(convert_date)
    mf = mf[(mf.date >= start_date) & (mf.date < end_date) ]
    mf["my_ents"] = mf.entities.apply(search_entities).apply(lookup_name)
    mf["sentiment"] = mf.headline.apply(lambda x : [x]) + mf.my_ents
    mf["sentiment"] = mf.sentiment.apply(sentiment_func)

    #combining news and price data into tf
    date_news = {}
    for d in mf.date:
        date_news[d] = []
    for i in range(mf.shape[0]): #combine sentiment of news articles on day
        date_news[mf.date.iloc[i]].append(mf.sentiment.iloc[i])
    for date in date_news.keys(): #combine all dicts per day into one
        result = {}
        for dictionary in date_news[date]:
            result.update(dictionary) #merging dicts can be improved later
        date_news[date] = result

    price_data          = price_data.loc[start_date:end_date].copy()
    price_data ["news"] = pd.Series(date_news)

    return price_data
```

This function takes the news data, price data, a start date, an end date and a function that extracts sentiment from the news articles. It returns a dataframe where the rows are the dates between the start and end date. There is a column for each stock ticker than contains the stock's price on that data. The final column of the dataframe contains a dictionary for each date, where the keys of the dictionary are the tickers talked about in the news that day and the values are the impact that the news article had on that ticker.

Algorithm 1 is the pseudocode for the *Predict*() function for the first model. The six inputs include the optional moved_ticker. When the user does not specify which stock has moved the algorithm looks in the inputted time series data for the stocks that have news about them that day, and what impact that news is expected to have.

The rest of the function's implementation is similar to the preliminary version of the *Predict*() function: first by defining key variables and then by applying the Ripple Equation from section 4.4. Now the algorithm must take into account that more than one stock may cause the initial ripple, but this is simple to incorporate as the algorithm just needs to alter multiple values in the change matrix.

```
Input  : predict_ticker: Stock to be predicted
         predict_date: Date to make a prediction
         orig_graph: Influence graph of stocks
         num_ripples: Iterations of the ripple equation
         timeseries_data: Dataframe containing prices and news
         moved_ticker Stock that has moved (optional):
Output: p_predict: Predicted price
         p_orig: Original price
         p_target: Actual price move
initialise:
  p_orig = price on predict_date
  p_target = price on day following predict_date
  R = adjacency matrix of orig_graph with 1s on diagonal
  D = a matrix of zeros
  p_t_minus_1 = 1d vector of prices from timeseries_data
  p_t_minus_2 = same as p_t_minus_1
  ;
if moved_ticker == None then
  | moved_ticker = a list of stocks that had news on predict_date ;
else
  | moved_ticker = moved_ticker as given in Input;
end
for ticker in moved_ticker do
  | Calculate change in share price;
  | Update p_t_minus_1 to for changed prices;
end
for i in 0 ... num_ripples do
  | Build change matrix D for iteration i;
  | p_t = ripple equation for iteration i;
  | p_t_minus_2 = copy(p_t_minus_1);
  | p_t_minus_2 = copy(p_t);
end
p_predict = entry of p_t that corresponds to predict_ticker;
```

Algorithm 1: First Prediction Algorithm

This first implementation of the *Predict*() function is basic but functional. In the next section we evaluate the first model.

Results

The purpose of first model was primarily to test the concept and reveal where improvements could be made for the second and third models. The following script was used to get an idea of the accuracy that the model was capible of. It is important to note that this is not a rigorous test of the model's performance, but more a test that the model is functional and the outputs are sensible.

```
 predict_ticker   = “TWTR”
orig_graph        = G
num_ripples       = 5
timeseries_data   = tf
acc_count         = 0
for date in timeseries_data.index:
    predict_date    = str(date)[:10]
    prediction, original, actual = predict(predict_ticker, predict_date, orig_graph, num_ripples,
         timeseries_data, moved_ticker = None)
    if (prediction > original) == (actual > original) : acc_count += 1

print (“ Accuracy”, acc_count/ len(timeseries_data.index))
```

This script simply iterates through each date in the time series data and makes a prediction for the Twitter stock. The direction of this predicted price move is compared to the direction of the actual price move for each date. This script demonstrates a directional accuracy of 78.46%.

5.2 Second Model

The second model takes the complexity up a level. In doing this, the aim is that some of the assumptions can be relaxed and the model will be a better representation of real-life market dynamics. Improvements have been made to the construction of the market graph, the measurement of the impact in from the financial news data, the implementation of the *Predict*() function and the model evaluation.

Market Graph

From the first model to the second model, the market graph has undergone relatively incremental improvements. Recall that in the first model, two entities in the graph were connected with an edge if their correlation was over a threshold of 0.8. In the first model, this correlation was static regardless of the prediction date. Of course this is an unreasonable assumption: the correlation of two stocks will vary throughout time. To combat this, the second model uses a rolling correlation that measures the correlation between each pair of stocks over the previous 30-days before the prediction date. In addition, in the first model the adjacency matrix for the market graph contained only 1s (if correlation was above threshold) or 0s (otherwise). The second model improves on this by including the actual correlation numbers in the adjacency matrix, if above the threshold, and 0 otherwise. Specifically, the adjacency function in the *build*() function above was updated from *simple_a*() to *weighted_a*() given by:

```
def weighted_a(x, thresh):
    if abs(x) <= thresh or abs(x) == 1: return 0.0
    else: return x
```

Price Data

The stock universe is unchanged between the first and second models, keeping the same 89 stocks. The timeline for the data has increased however, from the 3-month period used in the first model to a six month period from the 1st March 2018 to 1st September 2018. The intention here is to strengthen the validity of any results obtained.

News Data

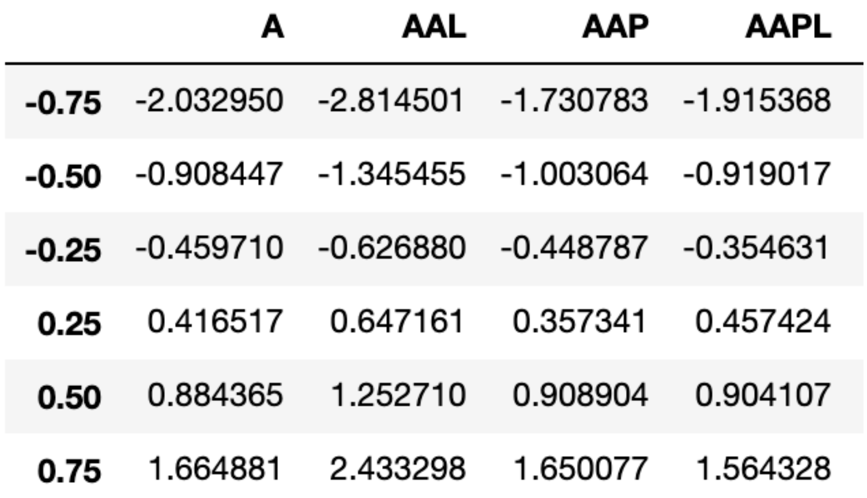
As with the first model, the second model uses a 10,000 article sample of the full 38,000 article corpus. What does change is the method of calculating the impact that each article has on the stocks in the stock universe. In the first model, the TextBlob NLP module is used to analyse the headline of each article. If the sentiment score is above 0.1 the impact to all stocks mentioned in that article is classified as +1, if the sentiment score is below -0.1 the impact is classified as -1, otherwise the impact is classified as 0. This is computed in a function called *calc_sentiment*(*headline*).

This function was replaced with *calc_sentiment_precise*(*headline*) that makes a minor improvement by changing the sentiment buckets from [-1.0, -0.1, 0.0, 0.1, 1.0] to [-1.0, -0.5, -0.1, 0.0, 0.1, 0.5, 1.0]. In turn, *calc_sentiment_precise*(*headline*) was replaced with *calc_sentiment_returns*(*headline*). This function is powered by a dataframe containing typical daily returns for each stock. This dataframe was built by computing daily returns for each stock throughout 2018 and then saving the quartiles for each stock. *calc_sentiment_returns*(*headline*) then computes a sentiment for each headline and converts this to an impact for each stock, based on that stock's typical returns. For example, if an article is deemed to have impacted AAPL by a sentiment of +0.25, then this quartile is looked up in the returns dataframe to see what return this corresponds to, in this case it would be +0.46%. A sample of this returns dataframe can be seen in figure 5.7.

Prediction

It is the *Predict*() function that has undergone the most change between the first and second model. Recall that the inputs for the function in the first model were predict_ticker, predict_date, orig_graph, num_ripples, timeseries_data, moved_ticker. For the second model the market graph was removed (variable name orig_graph) as an input and instead call the *build_graph*() function inside the *Predict*() function. This changes requires that the *Predict*() function has thresh as an input. The function also has two additional inputs use_sentiment and dampening. The second model no longer requires predict_ticker as an input.

Calling *build_graph*() inside the *Predict*() function enables a unique graph to be constructed each time the *Predict*() function is called. This enables the inclusion of rolling correlation as described above.



	A	AAL	AAP	AAPL
-0.75	-2.032950	-2.814501	-1.730783	-1.915368
-0.50	-0.908447	-1.345455	-1.003064	-0.919017
-0.25	-0.459710	-0.626880	-0.448787	-0.354631
0.25	0.416517	0.647161	0.357341	0.457424
0.50	0.884365	1.252710	0.908904	0.904107
0.75	1.664881	2.433298	1.650077	1.564328

Figure 5.7: The returns dataframe containing typical daily returns split into quartiles

The variable use_sentiment was included for debugging purposes and can largely be ignore. Simply, this variable is set to False by default but if set to True it finds stock price impacts by looking ahead 1 day to see the actual change in the stock price. Of course this is not a reasonable assumption at all, and in all testing this variable is set to False so that the price impact is obtained by calculating sentiment using the financial news data.

The dampening variable is included in this model is a fundamental difference to how to function computes the ripple equations. The Ripple Equation is the central equation of this paper. It the governs how a stock's price impact spreads to the price of other stocks in the market. The equation is applied iteratively using matrix multiplication, and can be pictured as an price movement spreading to adjacent stocks on each iteration until the impact has spread to the entire connected component. Intuitively, one would expect that on each iteration the impact is reduced somewhat and that the price movements are less powerful the further away you get from the "source". The dampening variable is the implementation of this affect. By default, the variable is set to 0, indicating no dampening. If S_j is adjacent to S_i and S_i experienced a price movement on a given iteration, the change in price of S_j is given by D[i][j] in the following:

```
dampening = 1 / (1 + iteration * dampening)
D[i][j]   = dampening * (p_t_minus_1[i] - p_t_minus_2[i]) / p_t_minus_2[i]
```

That is, a dampening factor is included. This dampening factor is equal to 1 when the input variable is set to 0 and the dampening factor decreases as the input variable increases. The dampening factor also decreases with each iteration, as consistent with intuition.

The final change to the *Predict*() function is that the input variable predict_ticker has been removed. This is because the *Predict*() function no longer returns only the prediction for a specified ticker. Instead, it returns predictions for all tickers in the market graph. For the second model, that is the 89 stocks in the stock universe.

The full code for this predict function is given in the appendix for this second model. It can be seen that the general structure of the function is unchanged, with the only changes being the incremental improvements described above. This change requires no additional computation, merely the format of the output needs to be changed so that the entire price vector is outputted from the Ripple Equation. As in the first model, the predictions are outputted along side the original price and actual price the following day.

Results

The evaluation of the second model is significantly more in depth than of the first model. As the updated *Predict*() function allows for the prices of all stocks to be predicted at once, this is what can be evaluated. In addition, a parameter search of the three key input parameters (dampening, threshold and number of ripples) is conducted. We present pseudocode for the evaluation section of the second model in algorithm 2:

```
Input   : end_date: Date marking beginning of data
          start_date: Date marking end of data
          timeseries_data: Dataframe containing prices and news
Output: results.txt: File containing one line per result, where each line contains parameters used and
          output of Predict() function
for dampening between 0 and 3 do
    for thresh between 0 and 1 do
        for num_ripples between 1 and 10 do
            for date between start_date and end_date do
                Call the predict function with these parameters;
                Of the stocks that moved, count the number that moved the same direction as predicted;
                Open results.txt file and save results for this combination of parameters;
            end
        end
    end
end
```

Algorithm 2: Script for Evaluation of Second Model

The full code for this evaluation is in the appendix for the second model. We now analyse the results. In total the *Predict*() function was called for every one of the 129 trading days in the six month period to 1st September 2018. The dampening, thresh and num_ripples variables iterated through 15, 10 and 9 different values respectively. This amounts to 1,350 different experiments run. With 129 days per experiment this amounts to 174,150 calls of the *Predict*() function in total. Each call of the function makes a prediction for each of the 89 stocks in the stock universe. In total there $174,150 \times 89 = 15.5$ million predictions made for the evaluation of the second model. When computing accuracy, this paper considers only the stock price predictions that are different to the original stock price, that is only the prices that are predicted to move. The direction of the prediction (is the price predicted to rise or fall?) is compared to the direction of the actual move between the prediction date and the price on the next trading day.

The mean accuracy over these directional predictions was 52.4%. We emphasise that the accuracy is computed by looking at the direction of price movement predicted for each stock on each day in the six month period. The number of correctly predicted price moves is compared to the total number of price moves to calculate an accurate number with that particular triplet of parameters (dampening, threshold, number of ripples). While the mean accuracy was 52.4%, the standard deviation accuracy was 1.21%. The lowest accuracy was 49.91% (dampening = 2.6, thresh = 0.8, num_ripples = 9.0) and the highest accuracy was 59.86% (dampening = 1.4, thresh = 0.9, num_ripples = 1.0). A surface plot of the parameter search is given in figure 5.8.

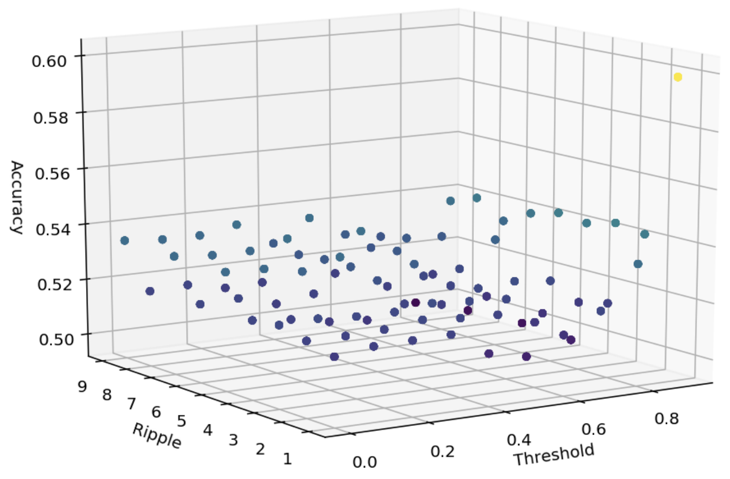


Figure 5.8: A surface plot of the parameter search for evaluating the second model

It is worth noting that threshold and number of ripples had a minimal effect on the results. The reason that dampening does not affect the results is that this parameter affects the magnitude of a stock's predicted price movement, but not the direction. Therefore it has no impact on the accuracy. Increasing the number of ripples that the model applies to the graph means that more nodes will be affected by a price impact, and so may include more nodes in the prediction. The consequence is that changing the num_ripples parameter will affect accuracy measurement, though its affect is limited as at some point the ripple has spread to the entire graph. This affect can be seen in the results in figure 5.8.

5.3 Third Model

The third and final model included in this paper brings two major additions to the second model. Firstly, the stock universe was expanded from the 89 stocks that were included in the first two models to the full 500 stocks that constitute the S&P500. Secondly, the *Predict*() function was updated to receive market impact signals from the full body of each news article. This contrasts with the first and second models that only measure the sentiment of the news headlines.

Market Graph

The third model constructs the market graph using the same *build_graph*() function described in first model section. The parameters used are the same as in the second model, specifically *weighted_a*() is the adjacency function used to decide which entities are connected and which are not. However, as the stock universe has increased to 500 stocks the resulting graph is quite different. The following is a short analysis of the components within the market graph, $\mathcal{G}$, that is formed when the threshold is set to 0.8 with market data as of 21st May 2018.

The market graph, $\mathcal{G}$, has 36 distinct connected component subgraphs. Its largest connected component subgraph, $\mathcal{G}_1$, contains 459 nodes. Aside from $\mathcal{G}_1$ there is one connected component subgraph containing three nodes, and three connected component subgraphs that each contain two nodes. The remaining 31 connected component subgraphs each contain one disconnected node. Notable disconnected nodes include F (Facebook), which was experiencing significant price volatility from the ongoing Cambridge Analytica scandal and the resulting fallout [42]. It is likely that this stock is not highly correlated to any other stock because it traded so idiosyncratically in this period. The largest connected component subgraph, $\mathcal{G}_1$, is fairly sparse amongst its nodes, with a minimum eccentricity of 5 and a maximum eccentricity of 10.

Price Data

The third model leverages price data from the IEX Finance API [36] in the same way as the second model. That is, time series data on a daily frequency over a six month period from 1st March 2018 to 1st September 2018. Retrieving price data for 500 stocks is functionally very similar to retrieving price data for 89 stocks. The only difference with the third model is that the price data has to be retrieved in five separate API calls and the results concatenated. This is simply due to a restriction in the IEX Finance API. The full list of S&P500 stock tickers was obtained from wikipedia [43] on 10th August 2019. This is relevant because S&P500 index is somewhat dynamic and the list of tickers included in the list changes slightly over time. For this paper we a static stock universe was used.

News Data

As with the first and second model, the third model analysed the Reuters corpus of financial news articles and extracted potential impacts on the stock price. For the third model, the full corpus of 38,507 articles was loaded. 6,740 of these were published in the six month period analysed by this model, roughly 35 articles per day.

The method of extracting the impact of each article was improved significantly for this model. While the first and second model found named entities in the entire article, they only analysed the sentiment of the article's headline. In contrast, the third model analysed sentiment of each sentence within the body of the article. This is a significantly more realistic, as practitioners would read entire article bodies. Furthermore the headline is often not representative of the sentiment on every stock mentioned in the article.

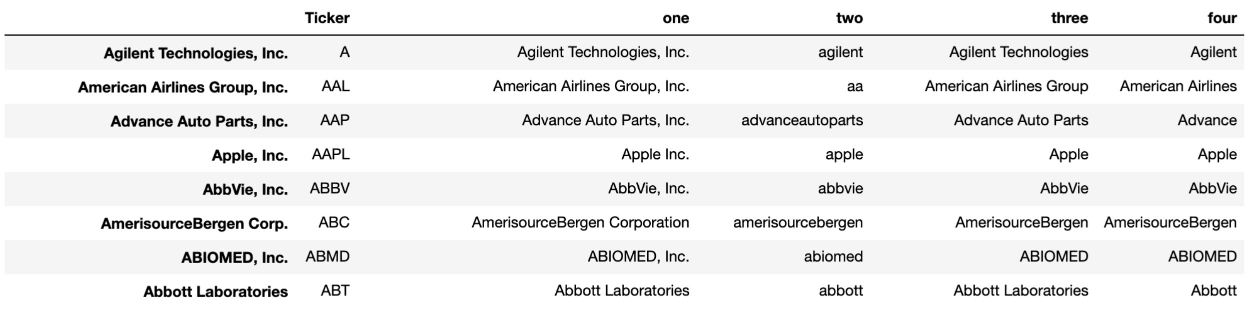
Consider the following excerpts from an article published on 9th May 2018 at 8:54am, just after the UK stock market opens. The headline of the article is *"Imperial Brands, energy stocks lift the FTSE, Burberry and Greggs sink"* and implies an overall neural sentiment. Within the article there are sentences such as *"Belgian billionaire sells stake in Burberry"*, implying a negative sentiment for Burberry; *"Vodafone shares stable after $21.8 bln deal"*, implying a neural sentiment for Vodafone; and *"..sanctions U.S. President Trump announced against Iran. Royal Dutch Shell and BP, up 1.9 percent and 1.6 percent respectively"*, implying a positive sentiment for BP and Shell.

Analysing each article on a per-sentence basis adds an extract dimension to the impact extracted. The implementation of this takes place in the function named *sentiment_body*(*body*) that is presented below.

```
1  def sentiment_body(body):
2      sentiment = Counter({})
3      count     = {}
4      if type(body) == float: return {}
5      sentences = nlp(body).sents
6      for sentence in sentences:
7          e = clean_entities ( extract_entities (sentence), with_type = False)
8          e = [lookup[ent] for ent in e if ent in lookup.keys()]
9          if len(e) > 0:
10             sentence_sentiment =
11                 Counter(calc_sentiment_returns([sentence.text] + e))
12             for ent in sentence_sentiment.keys():
13                 if ent in sentiment.keys():
14                     if abs(sentence_sentiment[ent])) == max(abs(sentiment[ent]), abs(
                           sentence_sentiment[ent])):
15                         sentiment[ent] = sentence_sentiment[ent]
16                 elif sentence_sentiment[ent] > 0.0:
17                     sentiment[ent] = sentence_sentiment[ent]
18     return sentiment
```

This function takes the body of an article as an input and returns a dictionary that contains the tickers present in that article body, along with a sentiment score for each ticker. On line 5 the python module SpaCy is applied to the body of the text to split it into sentences. The main section of the function iterates through each sentence. On line 7 the entities are extracted and preprocessed. On line 10 the sentiment function *calc_sentiment_returns*() (as described in the second model) is applied to each sentence. If the sentiment for that sentence is the most extreme (absolute value is higher) then assign it as the sentiment of that article for corresponding ticker. It is this function that computes the impact of an article's body on each stock mentioned in the article.

The price data is retrieved by ticker, whereas the impact of the news data is measured for each named entity in the article. To interface between the two a lookup table was created. For the first and second models this lookup table was made manually, mapping tickers to their numerous names (e.g. F was mapped to facebook, fb, instagram). This is practical for 89 stocks, but not for 500 stocks. For the third model a lookup table was generated by obtaining the official stock name for each ticker from the IEX Finance API and applying a rules based mapping to approximate a list of names for each stock. Figure 5.9 shows an example of the dataframe that was generated using this rule based method.



	Ticker	one	two	three	four
Agilent Technologies, Inc.	A	Agilent Technologies, Inc.	agilent	Agilent Technologies	Agilent
American Airlines Group, Inc.	AAL	American Airlines Group, Inc.	aa	American Airlines Group	American Airlines
Advance Auto Parts, Inc.	AAP	Advance Auto Parts, Inc.	advanceautoparts	Advance Auto Parts	Advance
Apple, Inc.	AAPL	Apple Inc.	apple	Apple	Apple
AbbVie, Inc.	ABBV	AbbVie, Inc.	abbvie	AbbVie	AbbVie
AmerisourceBergen Corp.	ABC	AmerisourceBergen Corporation	amerisourcebergen	AmerisourceBergen	AmerisourceBergen
ABIOMED, Inc.	ABMD	ABIOMED, Inc.	abiomed	ABIOMED	ABIOMED
Abbott Laboratories	ABT	Abbott Laboratories	abbott	Abbott Laboratories	Abbott

Figure 5.9: A sample of the lookup table used to map tickers to names

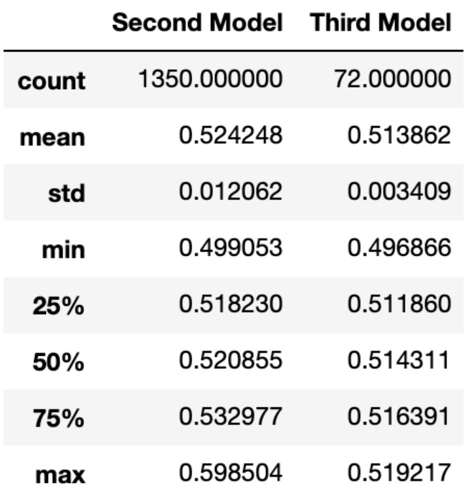
These mappings are far from perfect, but give a good approximation for the paper's purposes.

Prediction

The implementation of the *Predict*() function remains unchanged between the second and third model.

Results

The third model is analysed in a very similar way to the second model. The primary difference is that the dampening parameter is fixed at 1 (i.e. no dampening) as this parameter does not affect the directional accuracy of the model.



	Second Model	Third Model
count	1350.000000	72.000000
mean	0.524248	0.513862
std	0.012062	0.003409
min	0.499053	0.496866
25%	0.518230	0.511860
50%	0.520855	0.514311
75%	0.532977	0.516391
max	0.598504	0.519217

Figure 5.10: Directional Accuracy of Second Model and Third Model

Figure 5.10 presents the evaluation results of the second and third models. Note that these results are not directly comparable as a different stock universe is used for each model, but it is instructive to place the results side by side to see the contrast. The first row, count, shows how many experiments took place for each model. The difference is primarily the fixed dampening parameter for the third model. The next striking thing about the results is that the second model had both a higher mean accuracy with 52.4% versus 51.3% in the third model, but more importantly the maximum accuracy for the second model was 59.9% versus 51.9% in the third model. The standard deviation of the two models shows that the results of the third model were much more consistent, though this isn't necessarily a good thing. In practise a user could take the parameters that achieved the highest results and trade using the model configured in this way.

The results raise an important question: why did the third model produce a lower directional accuracy score compared to the second model? The expanded stock universe is the most likely reason, as the news data on the smaller stocks in the full S&P500 is not as exhaustive as it is on the larger 89 stocks included in the second model. This, and potential model improvements, is covered in detail in the next chapter.

Chapter 6

Discussion

So far this paper has outlined the motivation for the project, the existing literature, the inspiration and how the model was iterated through it's three different versions. This chapter puts that work into context, discussing how the project and its results compare to similar work. This chapter also explores the directions that the project may take beyond this paper, potential improvements and ideas for the validity of the results.

6.1 Evaluation

From the outset, the ambition of this project has been to combine techniques from Economics, Time Series Analysis, Graph Theory and Natural Language Processing. These techniques are combined using the Ripple Equation, as described in Section 4.4. It is right to judge this novel contribution by the strength of its results in comparison to price prediction attempts in related work but also it is important to take into context the assumptions that each paper made. What matters to a practitioner is finding a model that will work in a real trading scenario.

Having described the implementation of the three models in the previous chapters, it is now possible to discuss the model's type. From one perspective, the model is unsupervised. It does not require any training: a user simply loads historical price and news data into the model for it to form predictions. There is no manually labelled data from which the model learns patterns before applying these patterns to unseen data. Despite this, the model utilises a pretrained sentiment classifier from TextBlob and a pretrained named entity classifier from SpaCy. These models are both supervised machine learning models. The model presented in this paper combines the two pretrained classifiers with the returns table to produce a predicted daily return for the stock being predicted. This returns table, and also the lookup table to match named entities to tickers, are both made before the model makes stock predictions. All this considered, the model presented in this paper is by no means a wholly unsupervised model.

This context allows the paper's model to be compared to models in the existing literature. The best results from the first, second and third models are 78.5%, 59.9% and 51.9% respectively. These scores are for directional accuracy, which amounts to a binary classification problem, and therefore random guessing would produce a result of 50%. Clearly the second and third models score only slightly above this, but any model that predicts prices correctly more often than incorrectly can still be helpful to practitioners. In "Twitter mood predicts the stock market" Bollen et al use social media data to predict the Dow Jones Industrial Average (DJIA) stock index [24]. Their best model achieves 86.7% directional accuracy predicting daily returns over a period of 15 trading days in December 2008. Not only is this a very small number of samples compared to the six-month window explored in this paper but Bollen et al are also attempting to predict the movement of just one price, the DJIA, in comparison to the 89 or 500 stocks predicted by the models in this paper.

Schumaker et al's 2009 paper titled "Textual Analysis of Stock Market Prediction Using Breaking Financial News" [26] uses a variety of machine learning techniques to predict the individual stock prices in the S&P500. In this way it is comparable to the model in this paper. How it differs is in the time frame that the predictions are made: over minutes, not days. Schumaker et al's best model produces a directional accuracy of 58.2%, a number that is in line with our results. Their paper also presents a "simulated trading" returns value, which this paper does not. From this perspective Schumaker et al achieved 2.84% annualised return.

As a final point of comparison for evaluating this paper's results, consider Preis et al's "Quantifying Trading Behavior in Financial Markets Using Google Trends" [21]. Preis et al analyse a dataset of Google Trends search terms along side the performance of the DJIA. The paper uses a vast time period, seven years from 2004 to 2011, to find which search terms coincide best with DJIA price movements. The aim of the Google Trends paper is to demonstrate the relationship between certain search terms and price performance.

For their "Google Trends strategy" they report a cumulative profit of 326% by trading the DJIA between 2004 to 2011. This equates to an average annualised return of 23% - a fantastic result for any trading strategy of an extended period of time. The major caveat is that the evaluation period is the same period as the period of time used to find the pattern. This does not detract from the aims of the paper, to find patterns in Google search terms that correspond to stock price movements, but it is a serious example of data snooping and broadly invalidates the practicality of the results.

6.2 Further Work

This paper achieved its stated goals of combining numerous topics in Computer Science to form a novel way of modelling the stock market and making price predictions. That said, there are many potential improvements that could be added to the model to develop it's performance and practical applicability. This section discusses those those potential improvements and presents how this paper's models have laid the foundations for further work.

Improved Influence Measures

Section 4.1 outlined a number of potential influence measures that could be used to construct the market graph. Out of all of these, only the first measure was implemented: the correlation between two entities in the graph. This influence measure is undirected and does not begin to capture the intuition that some stocks in the market influence other stocks but are not themselves influenced by those same stocks. Given more time experimentation with other influences measures could prove valuable. In particular the $covariance(S_i, S_j)/variance(S_i)$ for two stocks S_i and S_j could be a method for quantifying what percentage of the volatility of S_i is attributable to S_j. Implementation of news-based influence measures or hybrid news-price correlation influence measures would add significant value to the work. These include counts of entity name co-occurrence within articles or the application of other natural language processing techniques to measure the relative importance of stocks.

Advanced Graphical Model Techniques

There is significant literature that covers the construction of graph models using advanced techniques. These techniques include learning covariance matrices using distributions such as the Wishart distribution. Markov random fields, Bayesian models and Probabilistic Graphical Models are all worth considering when looking to build a model that learns the adjacency matrix representation of a graph given input data. Denev's 2015 book titled "Probabilistic Graphical Models: A New Way of Thinking in Financial Modelling" [44] outlines an innovative method to incorporate human expert intuition and market dependencies into asset price models. Applying this approach, or one of the other advanced techniques, to stock market prediction would be an interesting avenue to explore.

Improved Ripple Equation Implementation

The Ripple Equation in its current form, as described in section 4.4, computes the movement of price changes across the market graph in an efficient manner. That said, its implementation could benefit from state-of-the art computational optimisation such as parallelisation or execution in another language besides Python. As it stands, one experiment using the third model to predict S&P500 prices over a six month period took around four minutes to complete. This time lag may be acceptable for trades executed on a daily basis but nowhere near fast enough for traders who operate in a low latency environment. In their 2013 paper "Low-Latency Trading", Hasbrouck and Saar [45] define "low-latency activity as strategies that respond to market events in the millisecond environment, the hallmark of proprietary trading by high-frequency traders".

Non-stock entities

The nodes of the market graph constructed in this paper are stocks. The graph models the relationship between these stocks to understand what influenced the prices movements of each stock. It stands to reason that prices are influences by more entities that just other stocks in the stock market. It would certainly be interesting to expand the scope to include other entities. It is possible that the entities could represent a number of different influencers in the stock market: regulators, politicians, central banks, stocks or even non-stock assets such as crude oil or the EURUSD exchange rate. This would give rise to two tiers of entities within the graph: those that emit asset prices to be predicted (e.g. stocks) and those that exist just to inform price movements of other entities (e.g. central bank interest rates).

Modelling the market as a graph captures the interacting relationships that entities have on other entities in the market. This captures an important component of a stock price's movement. There is arguably another component that this model does not capture: the overall market trend or momentum. In the model described in this paper, if a price ripple does not reach a particular stock node then that stock price is predicted to be unchanged. This does not stand up to intuition. Instead it would be reasonable to predict that a stock unaffected by the news drifts somewhat in the same direction as the overall market, industry or with a cluster of stocks that it is loosely correlated to. Including a node that represents overall market or industry trends would serve as a way of incorporate this into the market graph.

Trading Strategy

Ideally this paper would have employed a more holistic method of measuring model performance. This would consist of a trading strategy that takes the predictions from the model and makes paper trades (i.e. trades executed without real cash) to build a portfolio. By comparing the initial investment and size of this portfolio after a set time period, the profitability of the portfolio can then be obtained. There is significant freedom in developing a trading strategy that could be used. This can be an advantage but also could lead to biases: picking a trading strategy that works without truly testing the model's validity. This trading strategy can be seen as the evaluation of a model or an addition component to the model itself, essentially increasing the model's complexity by adding an extra moving part. In this paper's development a trading strategy was tested but subsequently removed for this reason: it is hard enough to attribute performance to the different components of the model as it is without introducing a trading strategy. If this was to be included as a method of evaluation, it would be recommended to implement a simple, transparent trading strategy such as buying all the stocks that are predicted to rise or are predicted to rise within a certain confidence interval.

Focused stock universe

Other improvements that could constitute a basis for further work include narrowing the stock universe to a particular industry or stocks in a particular market. This might improve the signal to noise ratio in the news and the relationship between stocks. Another improvement is that the pretrained sentiment analyser used in this paper was trained on general text articles. Instead it would interesting to build a supervised machine learning model that predicts the impact on a stock price given a day's news articles. This alone would take significant work but has potential to improve the results meaningfully. In a similar vein, the named entity recognition tool used in this paper is not trained specifically to look for market entities. It would be interesting to build such a classifier and apply it to a further iteration of this paper. Finally, as with all machine learning models, this one would benefit from more data - particularly more news data. It is not possible to have complete access to all information in existence, but a representative sample would likely improve model performance. This would need to include multiple daily news sources, short term news such as Twitter data and long term text data such as annual reports or 10-K stock filings. An example of a successful attempt to use 10-K filings to predict stock movement can be seen in Microsoft's article titled "Stock Market Predictions with Natural Language Deep Learning" [46].

Chapter 7

Conclusion

In conclusion, this paper has introduced a novel way of modelling and predicting price movements in the stock market. This has been achieved through the introduction of the Ripple Equation. The paper's stated goals have been achieved and demonstrated. The evaluation of this model shows that results are comparable to the existing literature. However, further work is required to truly reveal the model's capability. The ultimate evaluation of the model's performance will be implementation of an advanced version of this paper's methodology into a real-life trading environment, tested on live data with real investments.

Bibliography

[1] F. Modigliani and M. H. Miller, "The cost of capital, corporation finance and the theory of investment," *The American economic review*, vol. 48, no. 3, pp. 261–297, 1958.

[2] "Definition of economics," Jun 2019. [Online]. Available: https://www.investopedia.com/terms/e/economics.asp

[3] E. F. Fama, "Efficient capital markets: A review of theory and empirical work," *The Journal of Finance*, vol. 25, no. 2, pp. 383–417, 1970. [Online]. Available: http://www.jstor.org/stable/2325486

[4] L. Bachelier, "Theorie de la speculation," *Annales scientifiques de l'Ecole Normale Superieure*, vol. 3e serie, 17, pp. 21–86, 1900. [Online]. Available: http://www.numdam.org/item/ASENS$_1900_{31}7_{21}0$

[5] B. G. Malkiel, "The efficient market hypothesis and its critics," *Journal of economic perspectives*, vol. 17, no. 1, pp. 59–82, 2003.

[6] "Stocktwits trader mood data." [Online]. Available: https://www.quantopian.com/data/psychsignal/stocktwits

[7] "Two sigma: Using news to predict stock movements — kaggle." [Online]. Available: https://www.kaggle.com/c/two-sigma-financial-news

[8] R. W. Schabacker and D. Mack, *Technical analysis and stock market profits.* Vision Books, 2008.

[9] A. S. Wafi, H. Hassan, and A. Mabrouk, "Fundamental analysis models in financial markets–review study," *Procedia economics and finance*, vol. 30, pp. 939–947, 2015.

[10] M. Gusev, D. Kroujiline, B. Govorkov, S. V. Sharov, D. Ushanov, and M. Zhilyaev, "Predictable markets? a news-driven model of the stock market," *Algorithmic Finance*, vol. 4, no. 1-2, pp. 5–51, 2015.

[11] "Google scholar." [Online]. Available: https://scholar.google.co.uk/scholar

[12] C. Y. Huang, "Financial trading as a game: A deep reinforcement learning approach," *arXiv preprint arXiv:1807.02787*, 2018.

[13] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[14] E. Chan, *Quantitative Trading: How to Build Your Own Algorithmic Trading Business*, ser. Wiley Trading. Wiley, 2009. [Online]. Available: https://books.google.co.uk/books?id=NZlV0M5Ije4C

[15] J. Yao and C. L. Tan, "A case study on using neural networks to perform technical forecasting of forex." *Neurocomputing*, vol. 34, pp. 79 – 98, 2000.

[16] G. Ritter, "Machine learning for trading," 2017.

[17] P. Ganesh and P. Rakheja, "Deep reinforcement learning in high frequency trading," *arXiv preprint arXiv:1809.01506*, 2018.

[18] M. Sewell, "Application of machine learning to financial time series analysis," 2017.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[20] T. M. Nisar and M. Yeung, "Twitter as a tool for forecasting stock market movements: A short-window event study," *The Journal of Finance and Data Science*, vol. 4, no. 2, pp. 101–119, 2018.

[21] T. Preis, H. S. Moat, and H. E. Stanley, "Quantifying trading behavior in financial markets using google trends," *Scientific reports*, vol. 3, p. 1684, 2013.

[22] M. U. Gudelek, S. A. Boluk, and A. M. Ozbayoglu, "A deep learning based stock trading model with 2-d cnn trend detection," pp. 1–8, 2017.

[23] A. S. Weigend, *Time series prediction: forecasting the future and understanding the past.* Routledge, 2018.

[24] J. Bollen, H. Mao, and X. Zeng, "Twitter mood predicts the stock market," *Journal of computational science*, vol. 2, no. 1, pp. 1–8, 2011.

[25] T. M. Nisar and M. Yeung, "Twitter as a tool for forecasting stock market movements: A short-window event study," *The Journal of Finance and Data Science*, vol. 4, no. 2, pp. 101–119, 2018.

[26] R. P. Schumaker and H. Chen, "Textual analysis of stock market prediction using breaking financial news: The azfin text system," *ACM Transactions on Information Systems (TOIS)*, vol. 27, no. 2, p. 12, 2009.

[27] C. Oh and O. Sheng, "Investigating predictive power of stock micro blog sentiment in forecasting future stock price directional movement." Citeseer.

[28] D. Easley and J. Kleinberg, *Networks, Crowds, and Markets: Reasoning about a Highly Connected World.* Cambridge University Press, 2010. [Online]. Available: https://books.google.com/books?id=atfCl2agdi8C

[29] J. R. Abrams, J. Celaya-Alcalá, D. Baldwin, R. Gonda, and Z. Chen, "Analysis of equity markets: A graph theory approach."

[30] M. D. Koenig and S. Battiston, "From graph theory to models of economic networks. a tutorial," *Networks, topology and dynamics*, vol. 613, pp. 23–63.

[31] D. Budai and D. Jallo, "The market graph: A study of its characteristics, structure & dynamics," 2011.

[32] W. Sun, C. Tian, and G. Yang, "Network analysis of the stock market."

[33] C. Larman, *Agile and Iterative Development: A Manager's Guide*, ser. Agile software development series. Addison-Wesley, 2004. [Online]. Available: https://books.google.co.uk/books?id=76rnV5Exs50C

[34] 2019. [Online]. Available: https://github.com/ChrisTDick/rippleequation.git

[35] "Sp dow jones indices." [Online]. Available: https://us.spindices.com/indices/equity/sp-500

[36] "Iex developer platform." [Online]. Available: https://iextrading.com/developer/

[37] R. Editorial, "Breaking news, business news, financial and investing news more — reuters.co.uk," 2019. [Online]. Available: https://www.reuters.com/

[38] 2019. [Online]. Available: https://www.thomsonreuters.com/en/about-us/trust-principles.html

[39] 2019. [Online]. Available: https://mediabiasfactcheck.com/reuters/

[40] 2019. [Online]. Available: https://github.com/rojour/boston$_results$

[41] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[42] S. Rodriguez, "Here are the scandals and other incidents that have sent facebook's share price tanking in 2018," 2019. [Online]. Available: https://www.cnbc.com/2018/11/20/facebooks-scandals-in-2018-effect-on-stock.html

[43] 2019. [Online]. Available: https://en.wikipedia.org/wiki/List$_of_S$%26$P_500_companies$

[44] J. Czerlinski Whitmore, "Probabilistic graphical models: A new way of thinking in financial modelling," 2017.

[45] J. Hasbrouck and G. Saar, "Low-latency trading," *Journal of Financial Markets*, vol. 16, no. 4, pp. 646–679, 2013.

[46] 2019. [Online]. Available: https://www.microsoft.com/developerblog/2017/12/04/predicting-stock-performance-deep-learning/

Appendix A

Appendix for First Model

List of stock tickers used in the first model
[”GE” , ”AMD” , ”T” , ”BAC” , ”AAPL” , ”MU” , ”F” , ”FCX” , ”BA” , ”WFC” , ”BMY” , ”MSFT” , ”CSCO” , ”NVDA” , ”SLB” , ”TWTR” , ”PFE” , ”DIS” , ”INTC” , ”MS” , ”SYMC”, ”C” , ”HBAN” , ”MRO” , ”CELG” , ”VZ” , ”WU” , ”CMCSA” , ”RF” , ”ORCL” , ”PG” , ”KMI” , ”QCOM” , ”CNC” , ”PYPL” , ”KEY” , ”FB” , ”CVS” , ”JPM” , ”XOM” , ”KHC” , ”BK” , ”MPC” , ”CTL” , ”M” , ”MRK” , ”KO” , ”HPQ” , ”AMAT” , ”FISV” , ”HAL” , ”DAL”, ”WBA” , ”APC” , ”MDLZ” , ”SCHW” , ”LUV” , ”MDT” , ”AAL” , ”HPE” , ”MO” , ”DVN” , ”BHGE” , ”COP” , ”AXP” , ”NFLX” , ”COTY” , ”FITB” , ”TGT” , ”EBAY” , ”COG” , ”NEM” , ”ABT” , ”SBUX” , ”WMT” , ”LOW” , ”EOG” , ”TJX” , ”GM” , ”FE” , ”EA” , ”HES” , ”CAG” , ”WDC” , ”V” , ”CVX” , ”NLSN” , EXC”, ”ATVI”]

Predict Algo for first model

```
def predict( predict_ticker , predict_date, orig_graph, num_ripples, timeseries_data, moved_ticker = None
    ):
    # initialise variables
    p_orig    = timeseries_data[ predict_ticker ][ str(predict_date) ] #original price
    p_target = timeseries_data[ predict_ticker ]. shift (−1)[str(predict_date) ] #price we are trying to
        predict
    R         = np.asarray(nx.adjacency_matrix(orig_graph).todense()) + np.diag(np.array([1]*len(
        orig_graph.nodes))) #influence matrix
    D         = np.zeros(R.shape) #change matrix

    #setting prices in graph
    for node in orig_graph.nodes:
        orig_graph.node[node]["px"] = timeseries_data[node][str(predict_date) ]

    #define original price variables
    G_t_minus_2 = orig_graph.copy()
    p_t_minus_2 = np.array([G_t_minus_2.node[x]["px"] for x in G_t_minus_2.nodes])
    G_t_minus_1 = orig_graph.copy()

    #what has moved?
    if moved_ticker == None:
        moved_ticker = timeseries_data.news[predict_date]. keys()
    else:
        moved_ticker = [moved_ticker]

    for ticker in moved_ticker:
        moved_p_orig = timeseries_data[ticker ][ str(predict_date) ] #original price
        moved_p_next = timeseries_data[ticker]. shift (−1)[str(predict_date) ] #next day price
        moved_change = (moved_p_next − moved_p_orig) / moved_p_orig
        G_t_minus_1.node[ticker ][ "px"] *= moved_change + 1

    p_t_minus_1 = np.array([G_t_minus_1.node[x]["px"] for x in G_t_minus_1.nodes])

    #apply the ripple equations
    for iteration in range(num_ripples):

        #create change matrix D for this iteration
        for i in range(D.shape[0]):
            for j in range(D.shape[1]):
                if i == j:
                    D[i][j] = 1
                elif R[i][j] == 1:
                    D[i][j] = (p_t_minus_1[i] - p_t_minus_2[i]) / p_t_minus_2[i]
                else:
                    0

        #run one iteration of ripple equations
        p_t = np.diag(np.matmul(R,D)) * p_t_minus_1

        #define new price variables
        p_t_minus_2 = p_t_minus_1.copy()
        p_t_minus_1 = p_t.copy()

    predict_ticker_index = list(G.nodes).index(predict_ticker)
    p_predict = p_t[predict_ticker_index]

    return p_predict, p_orig, p_target
```

Appendix B

Appendix for Second Model

Predict Algo for second model

```
def predict(predict_date, num_ripples, thresh, adjacency_func, timeseries_data, moved_ticker = None,
            use_sentiment = False, dampening = 0):

    # initialise variables
    orig_graph        = build_graph(date = predict_date, thresh = thresh, timeseries_data =
        timeseries_data,
                                   adjacency_func = adjacency_func)
    timeseries_data  = timeseries_data[list(orig_graph.nodes) + ["news"]]
    p_orig            = timeseries_data.loc[str(predict_date)] #original prices
    p_target          = timeseries_data.shift(-1).loc[str(predict_date)] #prices we are trying to predict
    R                 = np.asarray(nx.adjacency_matrix(orig_graph).todense())                    + np.
        diag(np.array([1]*len(orig_graph.nodes))) #influence matrix
    D                 = np.zeros(R.shape) #change matrix

    #setting prices in graph
    for node in orig_graph.nodes:
        orig_graph.node[node]["px"] = timeseries_data[node][str(predict_date)]

    #define original price variables
    G_t_minus_2 = orig_graph.copy()
    p_t_minus_2 = np.array([G_t_minus_2.node[x]["px"] for x in G_t_minus_2.nodes])
    G_t_minus_1 = orig_graph.copy()

    #what has moved?
    if moved_ticker == None:
        moved_ticker = timeseries_data.news[predict_date].keys()
    else:
        moved_ticker = [moved_ticker]

    #how much has it moved?
    moved_ticker = [ticker for ticker in moved_ticker if ticker in orig_graph.nodes()]
    for ticker in moved_ticker:
        moved_p_orig = timeseries_data[ticker][str(predict_date)] #original price
        if use_sentiment:
            moved_change = timeseries_data.news[predict_date][ticker]
            moved_p_next = moved_p_orig * (moved_change + 1)
        else:
            moved_p_next = timeseries_data[ticker].shift(-1)[str(predict_date)] #next day price
            moved_change = (moved_p_next - moved_p_orig) / moved_p_orig

        G_t_minus_1.node[ticker]["px"] *= moved_change + 1

    p_t_minus_1 = np.array([G_t_minus_1.node[x]["px"] for x in G_t_minus_1.nodes])

    #apply the ripple equations
    for iteration in range(num_ripples):
        #create change matrix D for this iteration
        for i in range(D.shape[0]):
            for j in range(D.shape[1]):
                if i == j:
                    D[i][j] = 1
                elif R[i][j] != 0: #careful, might need to change when you change the adjacency matrix!
                    #dealing with divide by 0 errors
                    if abs(p_t_minus_2[i]) < 0.00001:
                        D[i][j] = 0.0
                    else:
                        dampening = 1 / (1 + iteration * dampening)
                        D[i][j]   = dampening * (p_t_minus_1[i] - p_t_minus_2[i]) / p_t_minus_2[i]
                else:
                    0

        #run one iteration of ripple equations
        p_t = np.diag(np.matmul(R,D)) * p_t_minus_1

        #define new price variables
        p_t_minus_2 = p_t_minus_1.copy()
        p_t_minus_1 = p_t.copy()

    zipped = zip(p_t, p_orig, p_target)
    p_predict = dict(zip(orig_graph.nodes, zipped))

    return p_predict
```

Evaluation of second model

```
end_date       = pd.to_datetime("1st September 2018")
start_date     = end_date - pd.DateOffset(months = 6)
start = datetime.now()
print("Starting at:", start)
timeseries_data = config_timeseries(news_data = df[cols], price_data = tf, start_date = start_date,
    end_date = end_date, sentiment_func = calc_sentiment_returns)
adjacency_func = weighted_a


print("Time to build variables:", datetime.now() - start)

print("\n", "****** BEGINNING TEST ******")
middle = datetime.now()

for dampening in [x/10.0 for x in range(0, 30, 2)]:
    for thresh in [x/10.0 for x in range(0, 10, 1)]:
        for num_ripples in range(1, 10, 1):
            total_acc    = 0
            total_moved = 0

            for date in timeseries_data.index[:]:
                acc_count    = 0
                moved_count = 0
                predict_date = str(date)[:10]
                predictions  = predict(predict_date = predict_date, num_ripples = num_ripples,
                    thresh = thresh,
                            adjacency_func = adjacency_func, timeseries_data =
                                timeseries_data,
                            dampening = dampening, moved_ticker = None, use_sentiment
                                = True)
            #print(predictions)
            for prediction in predictions.values():
                predicted    = prediction[0]
                original     = prediction[1]
                actual       = prediction[2]
                pred_change = (predicted - original) / original
                act_change  = (actual    - original) / original
                if abs(pred_change) > 0.0001:
                    moved_count += 1
                    if (pred_change > 0) == (act_change > 0):
                        acc_count += 1
            total_acc    += acc_count
            total_moved += moved_count
            if moved_count == 0: acc = "n/a"
            else:
                acc = "{:.2f}".format(acc_count / moved_count)
                #print("accuracy", acc_count/moved_count)



        print("num_ripples:", num_ripples, "counts:", total_acc, total_moved,
              "accuracy:", "{:.2f}".format(total_acc / total_moved), "\n")

        result = (dampening, thresh, num_ripples, total_acc, total_moved, total_acc /
            total_moved)

        with open(filename, "a") as f:
            f.write(str(result) + "\n")
```

Appendix C

Appendix for Third Model

Evaluation of third model

```

print("\n", "****** BEGINNING TEST ******")
middle = datetime.now()
for thresh in [x/10.0 for x in range(2,10,1)]:
    for num_ripples in range(1,10):
        total_acc    = 0
        total_moved = 0

        for date in timeseries_data.index[1:]:
            acc_count   = 0
            moved_count = 0
            predict_date = str(date)[:10]
            predictions  = predict(predict_date = predict_date, num_ripples = num_ripples, thresh =
                 thresh,
                                adjacency_func = adjacency_func, timeseries_data =
                                    timeseries_data,
                                dampening = dampening, moved_ticker = None, use_sentiment =
                                    True)
            #print(predictions)
            for prediction in predictions.values():
                predicted    = prediction[0]
                original     = prediction[1]
                actual       = prediction[2]
                pred_change = (predicted - original) / original
                act_change  = (actual    - original) / original
                if abs(pred_change) > 0.0001:
                    moved_count += 1
                    if (pred_change > 0) == (act_change > 0):
                        acc_count += 1
            total_acc    += acc_count
            total_moved += moved_count

        try:
            accuracy = total_acc / total_moved
        except:
            accuracy = 0.0

        result  = (dampening, thresh, num_ripples, total_acc, total_moved, accuracy)

        print("num_ripples:", num_ripples, "counts:", total_acc, total_moved)
        print("accuracy:", "{:.2f}".format(accuracy), "\n")

```

```
            with open(filename, "a") as f:
                f.write(str(result) + "\n")

        print("Time to predict and evaluate:", datetime.now() - middle)
```