

Mapping Sentiment Data derived from text onto Financial Market Data

Chukwuka I. Ajeh
991129

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Bachelor of Science



Swansea University
Prifysgol Abertawe

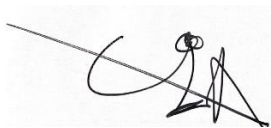
Department of Computer Science
Swansea University

April 28, 2021

Declaration

This work has not been previously accepted in substance for any degree and is not being con- currently submitted in candidature for any degree.

Signed



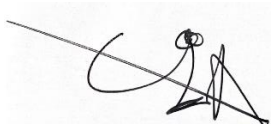
(candidate)

Date 27/04/2021

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed



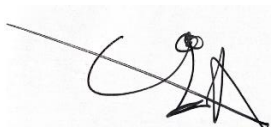
(candidate)

Date 27/04/2021

Statement 2

I hereby give my consent for my thesis, if accepted, to be made available for photocopying and inter-library loan, and for the title and summary to be made available to outside organisations.

Signed



(candidate)

Date 27/04/2021

I would like to dedicate this work to God, who has continually given me strength and courage to complete this degree.

Abstract

For the past few decades, financial institutions have increasingly begun to use algorithms and mathematical models to automate trading in the stock market. With the recent rise in popularity of research in Deep Learning and Big Data, along with increased processing speed, financial firms are now capable of developing highly sophisticated systems that can learn market trends in real-time using a diverse array of sources. However, it has not been without difficulty, stock markets are notoriously noisy due to their volatility, and any sudden news can break any naturally occurring trend. Nevertheless, with a proven ability to take in a vast amount of data, learn from it, and apply these learned techniques to new situations, attempts at using machine learning to find correlations and predict future price movements will only continue to increase.

Recent works have used Recurrent Neural Networks to try to better predict price movements due to their ability to hold context within text and sentences (Xin Du, 2020) by looking at the sentiment within data. This paper presents a custom recurrent neural network model that will try to derive sentiment from the dataset given, with the results then being mapped to financial graphs to briefly discuss the relationship between the sentiment data and subsequent price movement.

Acknowledgements

I would like to thank my family for their support and love. To my parents especially who have been a source of inspiration as the kind of man I want to be in life. I will be forever thankful. I would like to thank my uncle Kingsley Eweka. Without whom I would not have come into this field if he had not introduced and inspired me to become a Software Engineer. I would like to thank my friends for their support and a wonderful university experience. Finally, I would like to thank my supervisor Dr JingJing Deng for his help in creating this paper and supporting me in this field. May God Bless you all.

Table of Contents

Declaration	3
Abstract	7
Acknowledgements	9
Table of Contents	11
Glossary	13
Chapter 1 Introduction	15
1.1 Motivations	15
1.2 Project Aims	16
1.3 Paper Overview	17
Chapter 2 Background	18
2.1 Natural Language Processing: An Introduction	18
2.2 Lexicons	19
2.3 Neural Networks: An Introduction	20
2.4 Recurrent Neural Networks: An Overview	20
2.5 Long Short-Term Memory Networks	22
2.6 Word Embedding - Bag of Words	24
2.7 A brief overview of Financial Markets	24
Chapter 3 Related Work and Methodology	26
3.1 Leveraging Financial News for Stock Trend Prediction with Attention-Based Recurrent Neural Network	26
3.2 Stock Prediction using Twitter Sentiment Analysis	27
3.3 Related Work Summary	27
3.4 Methodology	27
3.4.1 Implementation	27
3.4.1.1 Dataset	28
3.4.1.2 Flair Model	28
3.4.1.3 Vader Model	29
3.4.1.4 Textblob Model	30
3.4.1.5 Custom Model	30
3.4.2 Challenges	32
4.2.1 Computational Limitations	33

Chapter 4 Experiments and Results	34
4.1 Experiments Outline	34
4.2 Experiment 1: Dataset Size	34
4.3 Experiment 2: Architecture Change	36
4.4 Experiment 3: Batch Size Variance	37
4.5 Final Custom Model and Comparisons	37
Chapter 5 Conclusion and Future Work	42
5.1 Comments on results	42
5.2 Future Work	42
5.3 Conclusion	43
Bibliography	44
Appendix A Batch Size Model Results	48

Glossary

Machine Learning – A sub-field of Artificial Intelligence, it is a discipline that focuses on algorithms that learn through experience.

Big Data – A field which aims to deal with extracting information from data sets that are too large or complex to be extracted from using traditional methods.

Dovish – Associated with statements and press releases from financial institutions. Generally, statements of this nature mean a firm will be taking soft or passive decisions. Usually when in negative economic situations statements are of a 'dovish' nature.

Hawkish - Associated with statements and press releases from financial institutions. Generally, statements of this nature mean a firm will strong or aggressive decisions. Usually, when in favourable economic situations, statements are 'hawkish'.

Neural Network – System inspired mainly by biological neural networks. Based on the concept of neurons and are used to train a system to learn a particular task.

Deep Learning – A sub-field within Machine Learning, focuses on using algorithms to construct artificial neural networks.

Natural Language Processing – Using machine learning techniques to analyse text data and derive sentiment and context from the inputted text.

Python – Python is a high-level, interpreted language, that emphasises user-friendly code. It has an extensive collection of machine learning libraries that have made it the go-to language for machine learning.

TensorFlow – TensorFlow is an open-source machine learning library created by Google Brain Team. Their focus is mostly on training deep neural networks.

Keras – Keras is an open-source library that acts as an interface for TensorFlow when making artificial networks.

API – Application Program Interface is an interface that facilitates interactions between multiple pieces of software—usually done via API calls.

Long Short-Term Memory (LSTM) – It is an Architecture for Recurrent Neural Networks, a variant of neural networks. LSTM aims to solve the problem of gradient vanishing and exploding.

Efficient Market Hypothesis – An hypothesis that states share prices reflect all information and consistent alpha generation or 'outperforming the market' is impossible.

NLTK – A leading platform created to work with human language data.

Textblob – A library created to handle processing of textual data.

Flair – Developed by Humboldt University, Flair is an NLP framework that allows for NLP techniques including the proposed Flair Embeddings proposed in their original paper.

MplFinance – matplotlib specialised for financial data plotting.

yFinance – A market data downloader used to download financial market data from Yahoo Finance.

Pandas – Open Source library that serves as data analysis tool.

Chapter 1

Introduction

The area of stock market price prediction is a field that has attracted industry professionals and researchers alike. Most recent works explore machine learning techniques and models better to predict the price movement of markets or relevant stocks. There are well-known barriers to the progression of the field, most notably the *efficient Market Hypothesis (EMH)*, which highlights the issue of models not having enough information to make accurate predictions. Despite this, researchers have still been able to produce models that have a) a high degree of accuracy, an example being Lui Huicheng 2018 paper that had an accuracy score of 65%. b) producing models that do not increase investor risk. In an ever-increasing number of works, the type of data has been text data. Whether in the form of headlines or, increasingly, social media comments. They are being used more within machine learning models. This suggests a relationship between certain types of data and market price movement aside from the traditional company performance data.

Another challenge is the trustworthiness of the news in question. Few people are unaware of the concept of *fake news* – false stories that appear to be news but are used to influence thought and opinion (Cambridge Dictionary, n.d). The potential of this malicious form of information manipulation alone is enough to cast doubt on the reliability of our sources of financial text data. However, for brevity, the validity/reliability of our headlines data is not within the scope of this project.

This paper aims to establish a graphical correlation between financial news headline data. We are deriving positive/negative sentiment or *sentiment polarity* and historical market price movement of the Standard & Poor 500 (S&P 500). The paper will utilise four pre-existing models and one custom model; NLTK's *SentimentIntensityAnalyser*, which uses the Valence Aware Dictionary for Sentiment Reasoning (VADER) model a lexicon and rules based model, Textblob another lexicon and rules based model, Flair, a *Character-level* LSTM that uses both word embeddings and its flair embeddings, finally, a custom RNN model implemented using TensorFlow. These four models will be used to establish whether such a relationship exists and determine their accuracy in calculating sentiment.

1.1 Motivations

There is precedent for investigating this relationship. Several related works have investigated the relationship between public sentiment and market fluctuations for price prediction. A frequently highlighted point of motivation is to improve the accuracy of predicting stock price movements (Xin Du, 2020). The primary benefit to a solution for this is, of course, higher accuracy for predicting stock price movements.

Another motivator for this project is establishing a standard for the kind of data we should be using. As mentioned before, several papers have used textual data to predict price movement based on their models. One example of such data is press releases and statements about governmental and economic policies released by banks and the government. Central banks can release statements that can be interpreted as *dovish* or *hawkish*. By using NLP techniques, companies can analyse the sentiment of the newly released statement. Based on related historical data correlating to either a historical price drop or price rise, the companies and firms can make informed decisions about certain investments in economic areas or other companies.

Another point of motivation is the adoption of NLP in the broader industry. Companies are now more than ever using automated trading through complex algorithms geared towards trading. Improved accuracy and a higher proven correlation will improve how these algorithms respond to a quick turnaround of highly impactful news.

Our motivation for this paper builds on these last points; we want to investigate the relationship between sentiment data and market price movement.

1.2 Project Aims

The primary objectives for this project were to implement either an existing proposed model or create a variant architecture of a custom RNN model. The model needed to be capable of discerning whether the headline data was positive negative in sentiment. This objective was achieved, thus completing the initial objectives stated in our initial project proposal:

- Use NLP to capture and analyse sentiment from historical financial news headline data.
- Become familiar with the TensorFlow library to implement and optimise the model.

After producing a model, we would then aim to correlate this across financial market graphs to clearly express the market sentiment at the time of that headline allowing market price data movement after that to be interpreted. For the custom model produced, this goal was also met completing other primary objectives expressed in our initial project proposal:

- Correlate the results against historical market data. We will visualise it in a graph to make it clear for all readers to interpret and understand.

1.3 Paper Overview

This paper is outlined as follows. Chapter 2 discusses the background, specifically concepts surrounding what is included in the pre-built models and custom model. Chapter 3 reviews related work in this field and the results that came from it as well as project methodology. Section 4 discusses experiments and model comparisons. Chapter 5 discusses the results of each of the models, future work, and the conclusion.

Chapter 2

Background

Chapter 2 introduces the concepts surrounding this paper, specifically the concepts and components present within the three models that are being compared. Following this, related works are introduced.

Sentiment analysis is a field of study within natural language processing that seeks to analyse and derive sentiment polarity (positive or negative) or sentiment intensity, how positive or negative a piece of text or word is. There are many approaches to deriving these values, with active research in both the lexicon-based and machine learning approaches.

2.1 Natural Language Processing: An Introduction

Natural Language Processing (NLP) is the study of programs being able to take in and process human language. More specifically, to understand what the text is trying to say. NLP applies algorithms that work to extract natural language rules such that the data can be converted to a form that the computer understands (Garbade, 2018). The computer will then work to extract meanings from all the sentences given to it as input. This is an essential field. With today's computers and their processing power, the NLP algorithms can analyse more data than we humans could ever achieve, without rest, and at a consistently high level (SAS, n.d). The problem with the data humans produce is that it is immense, complex, diverse, and, most importantly, unstructured. A lot of human languages are ambiguous and are constantly changing and evolving (Goldberg, 2017). As such, we must find ways to structure the data to allow the computers to understand.

At present, the best-known methods for dealing with the challenge of human language is supervised machine-learning algorithms or lexicon based approaches. They attempt to find patterns and regularities that define rules to the pre-annotated data (Goldberg, 2017). By feeding the algorithm pre-annotated data, we can train it to develop patterns that allow it to categorise the language.

Another problem for NLP is that the data, in this case, the human text is discrete (Goldberg, 2017). Whilst we as humans can understand the relation of terms and specific words, besides its external meaning. Furthermore, as words come together to form sentences, the overall meaning of the sentence changes; these properties mean that the formed meanings of the words can be infinite (Goldberg, 2017).

Neural Networks (NNs), which we will go into more detail about later, provide a way to combat this challenge. A significant component of NNs is that they map discrete symbols to continuous vectors in low dimensional space (Goldberg, 2017). This is known as an Embedding layer. By doing this, we allow them to become mathematical objects for which we can then calculate the distance between these vectors and, by extension, the words of the sentence themselves. As the supervised learning progresses, the network learns to combine the words to make them useful. Our project will find positive and negative meaning within the text to then further map to price movements on the stock market.

Two main techniques used to complete NLP tasks are *syntactic analysis* and *semantic analysis* (Garbade, 2018). Syntactic analysis is used to check how well a given piece of text aligns with a given set of grammatical rules. A few notable examples of techniques used to achieve this are lemmatisation, word segmentation and parsing. Lemmatisation is the process of reducing inflected forms of a word into a single form for easy analysis. As the term suggests, word segmentation is the breaking up of large texts into smaller distinct units. Lastly, parsing involves a high degree of processing due to it using grammatical analysis to understand the given text.

Semantic analysis works to understand the meaning behind the text. This is the more complex part of NLP as we must apply algorithms to understand the interpretation of what the text is trying to convey. Techniques within semantic analysis include Natural language generation, word-sense disambiguation, and Named Entity Recognition (NER) (Garbade, 2018). Natural language generation uses databases to try to find the semantics and translate this to human language. Word sense works to understand the context of the text, to provide meaning to the words within the body of the text. Finally, named entity recognition categorises text into pre-set groups (Garbade, 2018).

2.2 Lexicons

Existing sentiment analysis approaches depend heavily on underlying sentiment lexicons. Sentiment lexicons are simply a list of lexical features labelled according to their semantic meaning (C.J.Hutto, 2015). The very best of these lexicons are manually created and validated. These prove to be the most reliable. However, this is a highly time-consuming process; most applied research leverages pre-existing solutions for sentiment lexicons.

By incorporating an emphasis on context awareness, one can improve sentiment analysis due to a deeper understanding of lexical properties (C.J.Hutto, 2015). For example, word-sense disambiguation is a particular resulting application. Despite the base definition and polarity (and intensity), surrounding words in the sentence can affect the intensity/polarity. This is because we are now considering its contextual meaning rather than the word's primary definition. This lexical language-based approach is the basis of the VADER and Textblob models, which will be referred to later in this paper.

2.3 Neural Networks: An Introduction

The other approach leverages the field of machine learning, specifically the use of neural networks.

Inspired by the neurons and their behaviour found in the human brain, Neural Networks consist of thousands of simple processing nodes that are densely interconnected (Hardesty, 2017). These interconnected networks focus on recognising different and complex relationships within vast amounts of data. The most basic types are 'vanilla' Neural Networks or feed-forward networks. Neural networks are used to classify large amounts of data into a small number of classes and predict new behaviours or data points, amongst other uses. As a result, they have seen applications within several industries, from social media and big tech to education to financial services. Neural networks provide the distinct advantage of mimicking the behaviours of the human brain at a capacity that human brains cannot achieve. Figure 1 below details the structure of a simple feed-forward network:

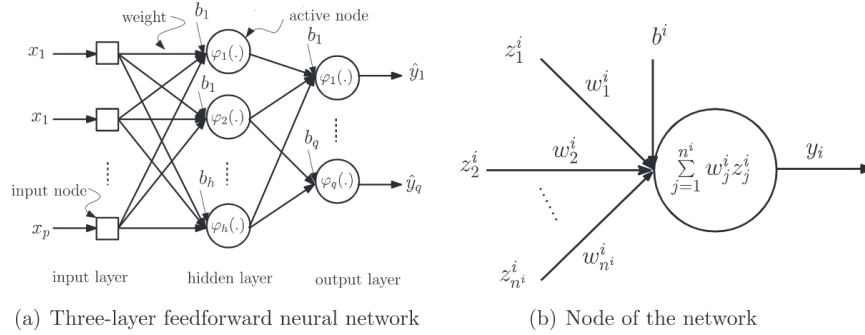


Figure 1: Three Layer Feed-Forward Network (a), where the input layer has p input nodes, the hidden layer has h activation functions and q output nodes.

As can be seen in Figure 1, a feed-forward network (figure 1a) consists of the following:

- An input layer denoted by input x .
- A hidden layer in the centre of the network consisting of active nodes with weights.
- An output layer also consisting of active nodes with a weight.

As shown in figure 1b, the structure of an active node is the summation of multiple multiplications between a weight and the output value of the node before it. The summation output is then applied to an activation function, traditionally, a sigmoid function, detailed in figure 2 below.

2.4 Recurrent Neural Networks: An Overview

Recurrent Neural Networks (RNNs) are another type of neural networks that are particularly well-suited for modelling ordinal or temporal data, such as language in the text because they allow information to be stored. This is because they contain loops in

the network that take the output and use it again within the input. Unlike neural networks, where inputs and outputs are considered independent of each other, recurrent neural networks present outputs depend on the prior outputs from which they can draw an inference. This key feature allows information to persist within the network, allowing them to build upon experiences to learn, thus creating a kind of memory. A singular example of this is shown in the figure below:

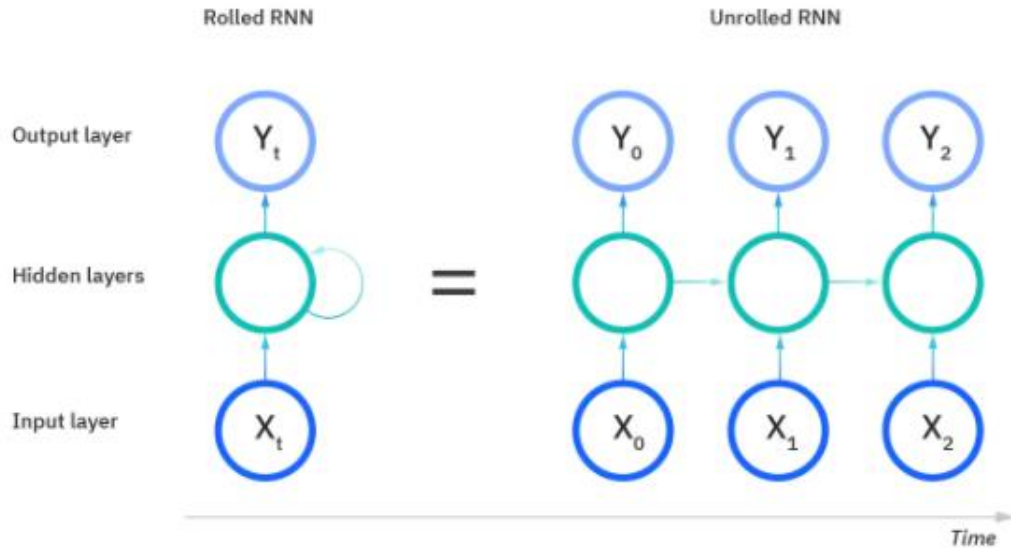


Figure 2: RNN cell unrolled (IBM, 2020).

The figure above shows a singular RNN component or a 'rolled' RNN representing the entire network. After the equals sign, we conceptually 'unroll' the RNN to understand the inner workings of the network. We will use the example phrase "today is a nice day" to understand an RNN. The unrolled model in the figure above represents each layer at a particular t timestep. Each layer maps to a single word of the phrase. Prior input/outputs are stored in the hidden layers of each layer (IBM, 2020). (Deep AI, n.d) (Haghian, 2019). As a result, RNNs are particularly useful for NLP as they are well-suited for sequential or temporal data. Input is a sequence of items that produce an output vector of fixed size that 'summarises the 'sentence' (Goldberg, 2017). A key benefit of RNNs is that they are trainable components that can be fed into other networks. An example of this could be feeding RNN output data into a Feed-Forward Network. A diagram of the network is shown below:

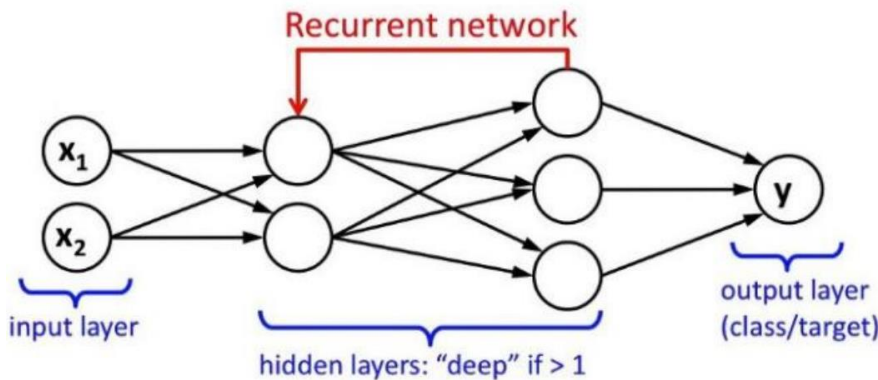


Figure 3: Diagram of a recurrent network.

Because there are feedback loops, there is a recursive nature to RNNs that we can describe as a *recurrence relation*. The recurrence relation of a basic RNN is defined as (Valkov, 2017):

$$S_t = f(S_{(t-1)} * W_{rec} + X_t * W_x)$$

Where:

S_t – the state at time step t
 X_t – External input at time t
 $S_{(t-1)}$ – the previous state
 W_{rec} – weights parameter
 W_x – weights parameter

Because of these loops, RNNs can compute the current state from the previous state and predict the next state from the current state.

Another characteristic of RNNs is the ability to parameter share (IBM, 2020) . In each layer of an RNN, the weight parameters are shared across that layer of the network, unlike NNs where each node has a different weight. These weights are still adjusted through backpropagation through time (BPTT), and gradient descent aid the model's learning. The sharing of weights introduces an input length constraint, making differing input lengths problematic to handle, but this is made up in the 'hidden' state where the relationships are captured in a constantly changing but serial fashion (Venkatachalam, 2019).

The custom model outlined further in chapter 5 uses an architectural variant of the RNN, known as a Long Short-Term Memory network, outlined in the next sub-chapter.

2.5 Long Short-Term Memory Networks

Introduced in 1997 by Hochreiter, Long Short-Term Memory (LSTM) networks are building units for layers in RNNs (Liu, 2018). LSTMs were designed to address specific issues within RNNs, specifically, the exploding or vanishing gradient problems. As a result of the backpropagation algorithm used (BPTT). As the errors are summed at each time step and gradient descent is applied, gradually, the gradient size, which represents the loss function along the error curve, gets smaller (IBM, 2020). This results in the updating of the weights till they become negligible. At this point, the model has effectively stopped learning. Occurring when the gradient is too big, exploding gradients create unstable models that result in weights growing to a state of NaN (IBM, 2020). A third issue is context retention. As the length of the text or other

input grows, we increasingly need more information or *context* which RNNs struggle to connect. LSTMs address these long-term dependency issues by retaining information for long periods of time. The architecture and Computation of LSTM cells are shown in the figure and equations below:

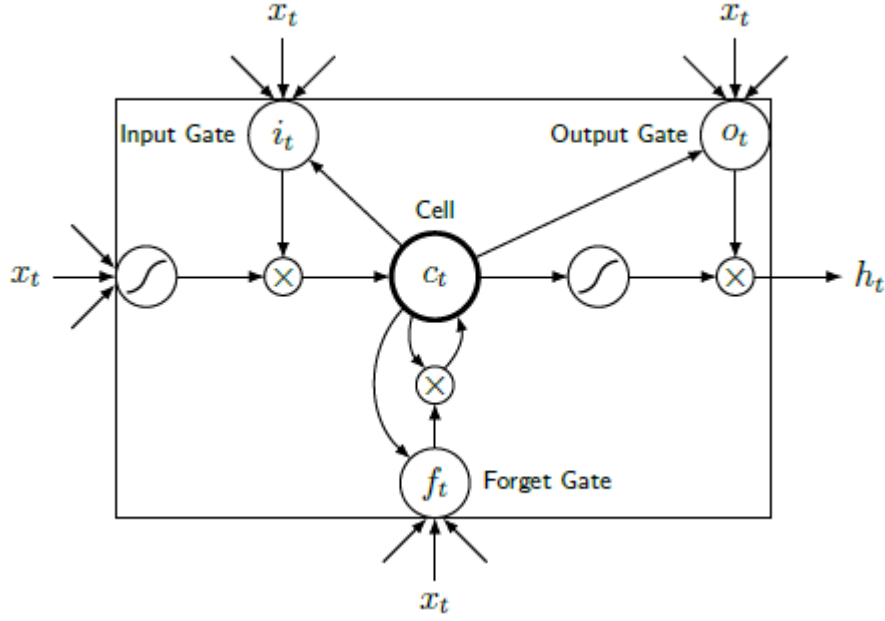


Figure 4: The original Long short term memory neural network architecture, as introduced by Hochreiter et al. f_t represents the forget gate's activation, i_t represents the input gate's activation, o_t represents the output gate's activation. x_t represents the input vector to the LSTM unit, h_t is the output vector of the LSTM unit. Finally, C_t represents the cell state.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad (1)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (2)$$

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \quad (3)$$

$$C_t = f_t \otimes C_{t-1} + i_t \otimes \tilde{C}_t \quad (4)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (5)$$

$$h_t = o_t \otimes \tanh(C_t) \quad (6)$$

As illustrated above in figure 4, an LSTM cell comprises an input gate, an output gate, and a forget gate and a central cell. Described by equation (1), the forget gate decides what to keep and what to throw away (Liu, 2018). It looks at the previous output vector

(h_{t-1}) and the current input vector x_t . Based on this a value between 0 and 1 is output. 0 'throw away', and 1 denoting 'keep'. Equation (2) represents the input gate i_t . This vector determines the values that will be updated at each time step t . It is at this point \tilde{C}_t is calculated using the tanh function as shown in equation (3). We update C_t from C_{t-1} and it is here we decide using f_t and i_t whether to keep or discard the information currently being stored, as shown in equation (4). Equation (5) represents the output gate filtering out redundant values. It is at equation (6) that we finally output. W and b denote weight and bias respectively (Liu, 2018). The custom model highlighted later in this paper use a variant of the LSTM, the Bi-directional LSTM, and the Flair model a Character-level LSTM, both of which will be discussed later in the paper.

2.6 Word Embedding - Bag of Words

Also known as BoW, the Bag of words model represents text data with machine learning algorithms (Brownlee, 2017). It allows us to extract features from the text for use in modelling. The text representation allows us to describe the occurrence of words within a body of text. It consists of:

- A vocabulary of known words
- A measure of the presence of known words

It is known as a 'bag' of words as we disregard grammar and word order but keep multiplicity (Liu, 2018). Put simply; we do not care *where* the word occurred but that it occurred at all. BoW comes under the umbrella of Word Embedding, a set of language modelling and feature extracting techniques found in natural language processing (NLP). Words or phrases are mapped to vectors of real numbers. A visualisation of this can be found in Chapter 5.

2.7 A brief overview of Financial Markets

Financial markets are marketplaces that provide an avenue for the purchasing and selling of stocks, bonds, foreign exchange (FX) and derivatives (Corporate Financial Institute, 2015). It allows businesses and investors a way to make more money or raise capital to grow their business.

As mentioned above, there are different markets, and there are markets for stocks, bonds, FX, and derivatives. This paper will briefly outline each one, but this paper will focus on stock price movement.

Stock markets deal in the trading of shares of a company. All shares are priced, and this price will rise and fall depending on how said company performs. Events like mergers and acquisitions can cause prices to rise and fall, and significantly negative news, such as a scandal or bankruptcy, can cause a stock price to nosedive. Usually, investors can either invest in a company directly, such as with Tesla and will have a stock price symbol accordingly. Tesla's is TSLA. Alternatively, one can also invest in indices; these are collections of leading companies for a given country and represent country market health

because the leading companies span all industries. A notable example is the FTSE 100 in the United Kingdom (UK).

The bond market's focus is for governments and companies to raise money to fund projects or investments (Corporate Financial Institute, 2015). Investors can buy and sell debt securities from a company or government for a fixed period before then paying it back along with interest.

The foreign exchange (FX) market is where the trading of currencies takes place. It is merely traders exchanging money at specific rates and then converting back when those rates are better. Unlike the other marketplaces, there is no centralised place for FX. Instead, it is done over the counter electronically (James Chen, 2020).

Finally, we have the derivatives market. As the name suggests, this is the buying and selling of derivatives, contracts whose value is based on the value of an asset that is being traded (Corporate Financial Institute, 2015). Commodity Futures is an example of this. For the paper, the focus will be on stock market predictions as prices are purely down to company performance or news surrounding the company and the market.

Chapter 3

Related Work and Methodology

Chapter 3 discusses papers relating to the work found in this paper. Research within this field has primarily been for stock trend prediction, whether the stock price will rise or fall. The principle behind this paper is similar; we aim to find the polarity within headlines to determine price movement at that particular point in time.

3.1 Leveraging Financial News for Stock Trend Prediction with Attention-Based Recurrent Neural Network

In this paper, the author proposed an attention-based LSTM model to make predictions on the S&P 500 market to surpass previous machine learning models. The proposed model outperformed many existing models, namely, the SVM, Bag-At-LSTM and E-NN models (Liu, 2018). The approach by Liu characterises the model into four stages:

- Input and Embedding Layer
- News-level Bi-LSTM and self-attention layer
- Day-level Bi-LSTM and self-attention layer
- Output and prediction layer

Each layer attempted to address a known issue regarding its scope. Using the first layer as an example, the paper addressed the handling of input; to address the sparsity problem, an LSTM based encoder was created to handle distributional representation. For word and character embedding, Liu proposed creating a custom embedding layer due to the limitation of existing pre-trained word embeddings Glove (Jeffrey Pennington, 2014) and Word2Vec (Tomas Mikolov, 2013). These existing word-embeddings were trained generally and were not suited to the specialised text of financial headlines (Liu, 2018). Furthermore, the resulting word embedding was then concatenated with a character level convolutional network (CNN) for even richer representation.

The model achieved an accuracy average accuracy score of 63.06% and a max accuracy score of 65.53%. Interestingly, the experimental results suggested that the proposed model could compete with existing models, incorporating other elements such as knowledge graphs into the learning process (Liu, 2018).

3.2 Stock Prediction using Twitter Sentiment Analysis

More similar to the aim of this paper, Mittal et al. proposed deriving sentiment polarity on Twitter data and used market data in (Dow Jones Industrial Average)DIJA to predict stock market movements. This paper operated on the premise of behavioural economics; emotions and moods of individuals affect their decision-making process, thus, leading to a direct correlation between "public sentiment" and "market sentiment" (Goel, 2011). Standard sentiment analysers proved inadequate. Thus, a custom model was created using three principles:

- Word List Generation – Develop a custom word list based on Profile of Mood States (POMS).
- Tweet Filtering – Filter out tweets that are less likely to express a feeling
- Daily Computation – use a word counting algorithm to score each POMS word.

The paper goes describe four machine learning models they used to predict price movement. SVMs and Logistic Regression prove to perform poorly with the proposed Self Organising Fuzzy Neural Networks (SOFNNs) model proving to perform best. What was notable about this paper was the introduction of a cross-validation technique to measure accuracy, *k-fold sequential cross-validation* (Goel, 2011). The principle behind this was to train all models for every day for n days. After this, they would then be tested for k days.

3.3 Related Work Summary

Our related work shows extensive research into stock trend prediction, specifically using several machine learning models on text data. The approaches are different, some utilise sentiment data (Goel, 2011) and others derive meaning using neural networks (Liu, 2018), the results prove to be similar in accuracy. The idea behind sentiment analysis found in Mittal et al.'s paper is the basis for the paper; however, coincidentally, we are taking a heavy machine learning approach using networks to predict sentiment to predict price movement very similar to Liu's paper. Whilst a GPU was available and used, the headline data was stripped down due to the memory constraints of the laptop on which the model was trained.

3.4 Methodology

In response to the research outlined above, chapter 4 outlines the implementation behind each of the four models that we implemented. Of the four, three are machine learning-based except for Vader, which is lexicon and rule-based.

3.4.1 Implementation

The Flair, Vader and Textblob models, are three pre-built and pre-trained models. As such, in this chapter, we will discuss the original author implementation behind them. They serve as controls to compare against our custom model built using TensorFlow.

3.4.1.1 Dataset

There are two datasets used; the first is an open dataset provided on Kaggle. See here for more details: <https://www.kaggle.com/aaron7sun/stocknews>. The second dataset is yahoo finance but via a library known as yFinance, a financial market data downloader. Using yFinance, we pull market data for the S&P 500 market spanning from July 2008 to July 2017 on a 1month scale. The yFinance data contains several data sources, but Open, Close, Adjusted Close and Volume data were extracted for our project. Volume was cleared of its original values and instead held the sentiment values representing the sentiment polarity predicted by each of the models. The dataset which for the rest of this paper shall be referred to as 'headline dataset' is structured in a particular way which table 1 below illustrates:

Date	Label	Top1	...	Top 22
2008-08-08	0	"b" Georgia downs two Russian warplanes' as countries move to brink of war""	...	"b"" No Help for Mexico's Kidnapping Surge " ""
...

Table 1: Dataset Structure.

Each line within the dataset holds 22 headlines for one day, all with the same sentiment label. 0 denotes a negative sentiment, and 1 denotes a positive sentiment. For the pre-build models, headline columns of the same day were dropped to ease parsing and better handling of memory. Headline text loaded as dataframe using the pandas library. For the custom model, the data was pre-processed slightly differently. All top 25 headlines were split into separate dataframes with their respective label. The columns were then renamed and to 'label' and 'text' and then concatenated along the columns building a larger dataset than we originally had. This larger headline dataset was then split into training, testing and dev with a split of 80/10/10, respectively. Using the training dataset split as a reference, figure 5 illustrates the resulting dataset, a tab spaced CSV file with only the headline data and the label.

```
0 "b"Georgia 'downs two Russian warplanes' as countries move to brink of war""
```

Figure 5: Snippet from the training dataset post-pre-processing.

3.4.1.2 Flair Model

We implemented an instance of the Flair framework. Developed by Humboldt University, the Flair framework is a state-of-the-art NLP Library. There are many features of the framework, such as implementing their proposed *flair embeddings*). We

made use of the pre-built text classifier of the TextClassifier class. We loaded the en-sentiment model. This model is a transformer model that makes use of the DistillBERT architecture. It has been trained on the IMDB movie reviews dataset and can detect positive or negative comments within movie reviews. We adapted this for use on our financial headlines dataset to find sentiment polarity within our text. The classifier was loaded onto the disk before being introduced to the dataset to predict sentiment; then, the headlines were converted to a list of sentences before being parsed into the model for prediction. Each sentence was then given a label and confidence score regarding how confident the model thought the sentiment was. Sentiment results were split from labels to then be added into added as a pandas dataframe.

Development wise an OOP approach was taken, placing the building of the model as its own class. We then create an instance of the model and treat the sentiment as a source of data. This data is then concatenated with the data pulled from yFinance. This is because the plotting tool used, mplFinance, only takes traditional values from market data. The sentiment is not included. To account for this, as mentioned above, we clear volume data already present and replace it with our sentiment polarity data. The data is then plotted on the graph along with financial data. A figure of the resulting render can be found below in chapter 5.

3.4.1.3 Vader Model

Using the NLTK library, we created an instance of the VADER model. The Vader model is a lexicon and rule-based model and utilises the principles introduced in sub-chapter 2.1. The Vader engine is constructed with a sensitivity towards social media text data (C.J.Hutto, 2015). As the model is constructed from a generalisable valence-based, human-curated sentiment lexicon, there is no need for training.

Figure 6 below highlights the overall method and approach used to construct VADER:

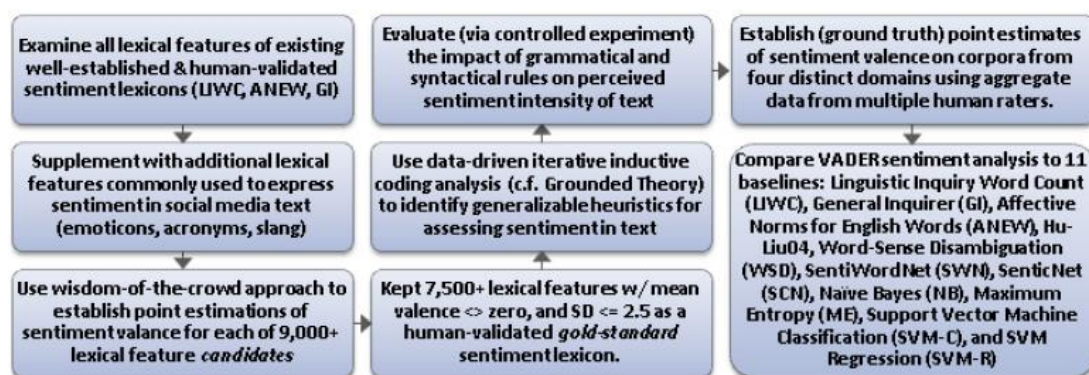


Figure 6: Method and approach overview (C.J.Hutto, 2015)

Like Flair, a class was constructed for handling the model and deriving sentiment from the headline data. The resulting data is four metrics: compound, negative, neutral and positive. The compound score is the resulting summed normalisation of all lexicon ratings between a value of +1 (representing positive) and -1 (representing negative) (Placeholder1). As this was done for each sentence, we ended up with a list of values

converted into a dataframe, and like the flair model above was loaded in place of the volume data pulled from yFinance. The data was then rendered as well and is shown in the figure below in chapter 5.

3.4.1.4 Textblob Model

The Textblob model is the last of the pre-build models we used. As mentioned in chapter 2. Textblob does use a lexical approach, but there are other elements also. Textblob calculates the average polarity and subjectivity over each word in text utilising a dictionary of words, from the WordNet3 lexical database (Kohli, n.d), and their tagged scores. As was the case with the previous pre-build models, we created a separate class to hold the Textblob model. Pulling headlines out of a headline dataframe we created, we constructed a corpus that was passed into a Textblob instance. We then pull sentences constructed by the Textblob instance and apply sentiment analysis to each sentence. This returns a tuple of subjectivity and polarity, but we specify and use only the polarity. The values are appended to the global list. An instance of the Textblob model we created was loaded into a rendering class to load the sentiment polarity values into the volume column and rendered using the mplFinance library. The results can be seen in the figure below in chapter 5.

3.4.1.5 Custom Model

The custom model we constructed used the TensorFlow library based on the text classification model example. Figure 7a and 7b below shows the architecture used following the model described in the library example:

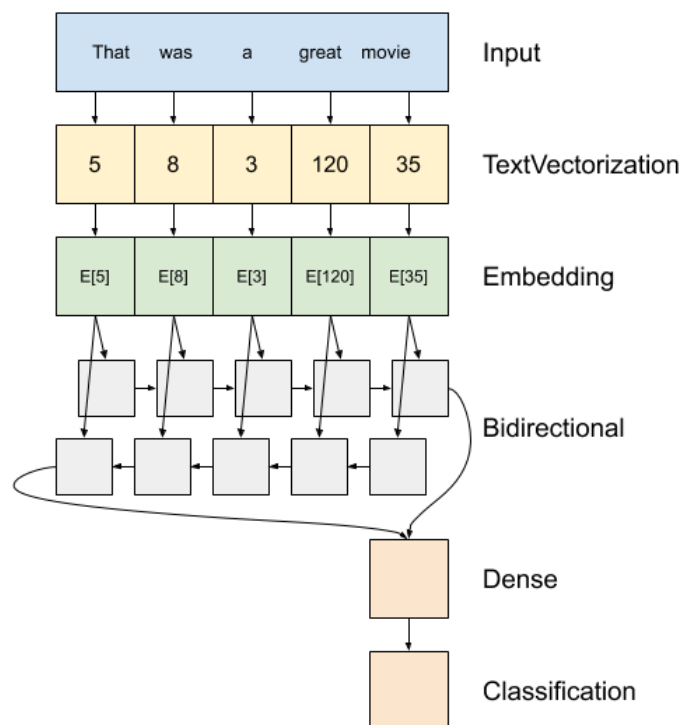


Figure 7: Original Model described by TensorFlow (Google TensorFlow, 2021).

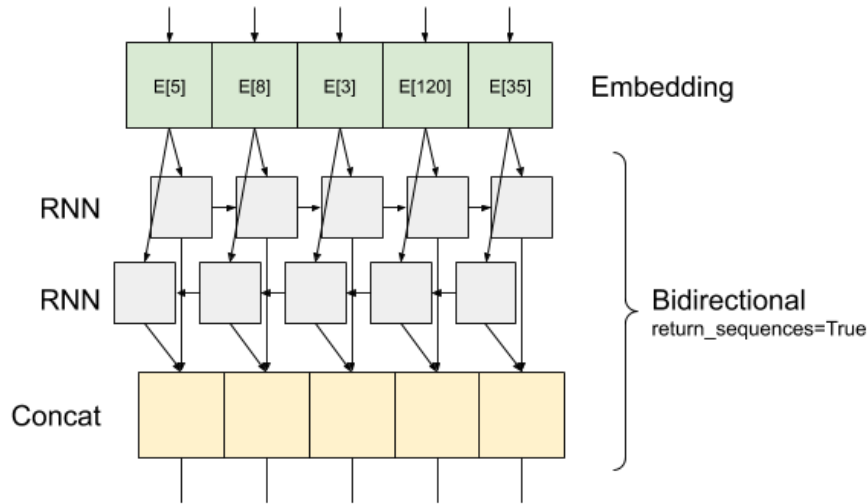


Figure 8: Adaptation using the double RNN modification (Google TensorFlow, 2021).

It features a vectorisation layer with a modified vocab size of 10000. This is used to vectorise the text. The encoder then adapted to the training dataset setting the layer's vocabulary (Google TensorFlow, 2021). We then encode the layer into token indices for use in the next layer. The indices are zero-padded to the longest sequence in the batch. The dimensions produced at this layer are then fed into the embedding layer, which embeds a vector per word. After this, there three Bi-directional LSTM layers of reducing cell numbers. These first two layers propagate the input forward and backwards through the RNN layer and concatenate the final output (Google TensorFlow, 2021). The first and second Bi-LSTM layers return the complete sequences at each timestep which is then passed to the next RNN layer. There is a dropout layer to prevent overfitting. Finally, the Dense layer processes the final output reducing to one logit for use in classification. The entire architecture is shown below in figure 9:

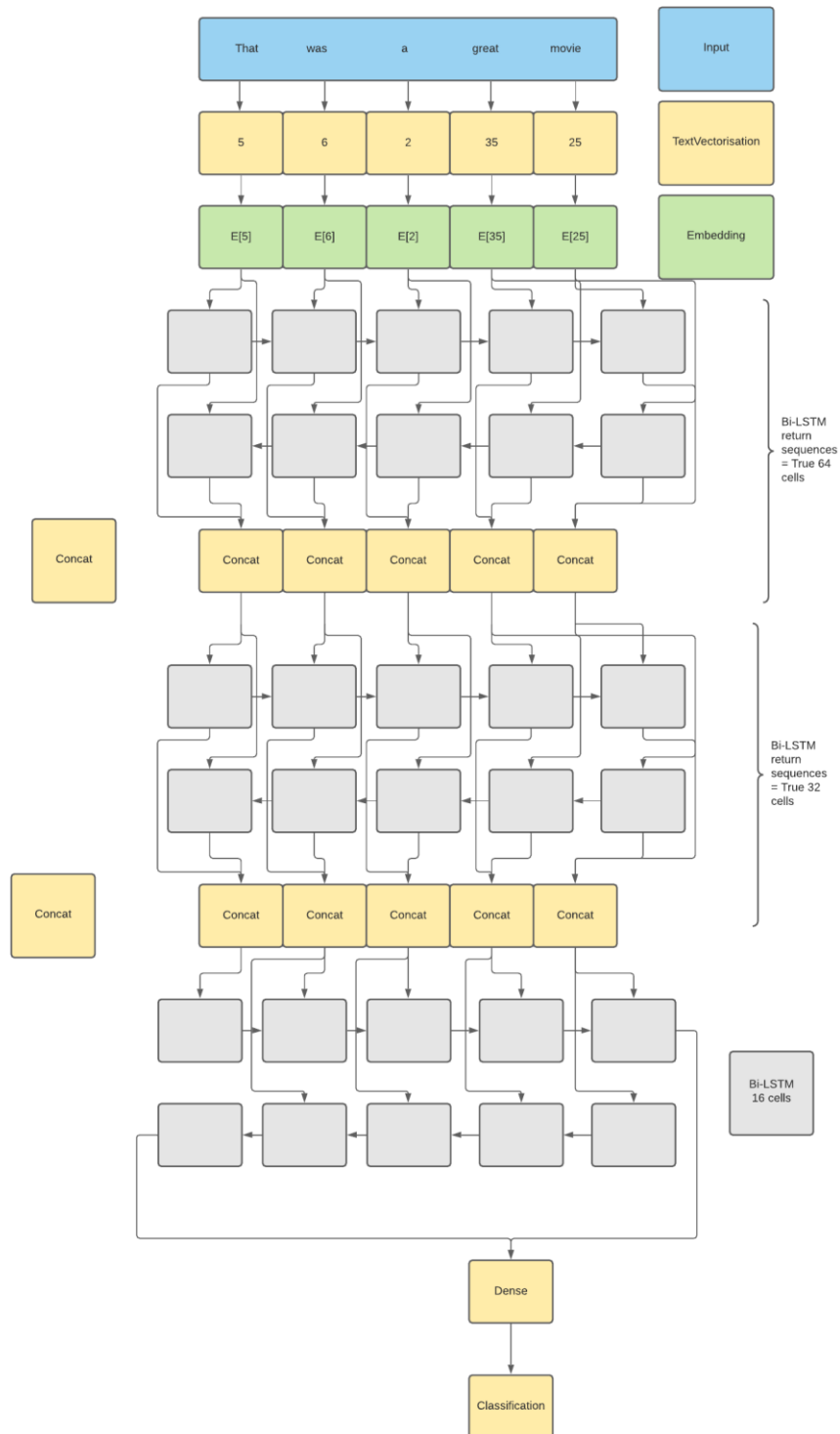


Figure 9: Custom Model Architecture based on the original RNN example model by TensorFlow (Google TensorFlow, 2021).

3.4.2 Challenges

4.2.1 Computational Limitations

Before starting this project, we identified that the device on which this model would be trained used a dedicated NVIDIA GPU. And, for training, this was used. The issue, however, was the memory. We quickly found that the memory available was far too small, and as a result, we downsized the dataset to a sufficient size so that the device could handle the training process better.

Chapter 4

Experiments and Results

In chapter 4 we outline the proposed experiments as discussing the result of each experiment performed. We then finally compare models in terms of accuracy discussing results that were found.

4.1 Experiments Outline

Experiments covered the custom model. The pre-build models served as controls to compare our custom model against. For experiments we broke down the approach into 4 experiment classes:

1. Dataset Size – Initially, we use only the top headline for the day, after further pre-processing we used the top 22 headlines for the day and their respective labels, vastly increasing dataset size and variance.
2. Architecture change – This happened once and from this adjustment onwards, all models used this architecture. This was due to time constraints.
3. Batch size – We varied batch size to speed up training and see if this affected accuracy. As the focus was only batch size for this experiment, the epoch was set to 1. Moreover, the results are shown below in subsection 5.4. The batch size was set to the following:
 - a. Batch size of 10
 - b. Batch size of 25
 - c. Batch size of 50
 - d. Batch size of 100

4.2 Experiment 1: Dataset Size

This was the simplest of the experiments and involved changing the dataset size. Initially, we started with a small dataset; only the top headline of the day was used. This resulted in the model overfitting massively and scoring a very low testing accuracy of 48%. This is likely because there was little variance in data, and so it is unable to adapt well to new information from the testing dataset.

Figure 10 below highlight accuracy and loss metrics:

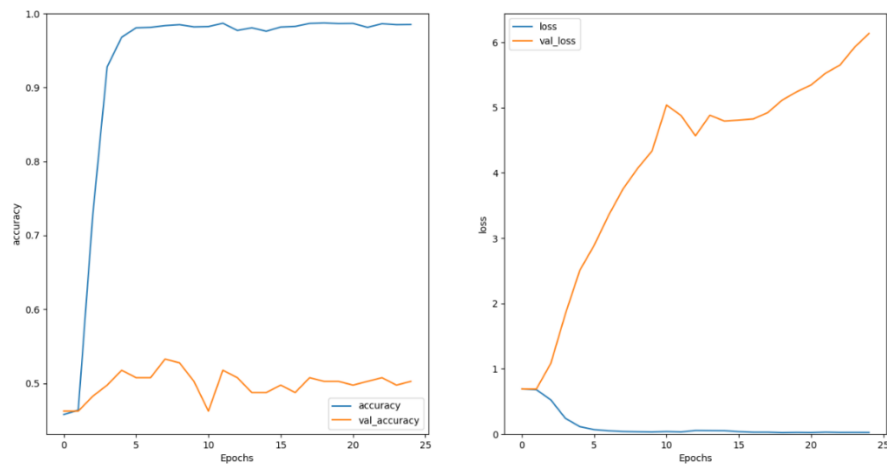


Figure 10: Accuracy and loss metrics from our low dataset size model.

This model was then used to predict sentiment for the top headlines of the day with the values then mapped on to the S&P 500 market graph at the bottom as sentiment bar charts as shown below in figure 11:

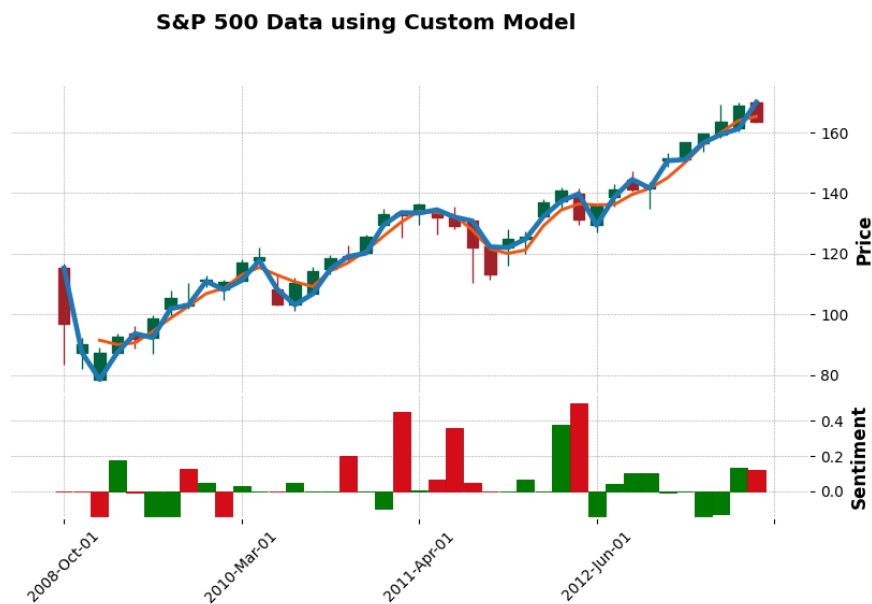


Figure 11: S&P 500 market data with predicted sentiment mapped across.

The model did a decent job at mapping sentiment onto the market graph with clear sections of the graph showing a period of negative or low positive sentiment just before the fall in market price and other areas showing slight bumps in market prices such as early 2008 where the price rises after a series of small positive headlines in a generally negative period as shown in figure 11. For broader context, this was in the year following the financial market crash of 2008. Market sentiment would have been predominantly negative.

Interestingly, we can visualise the word embeddings and their vectors to understand better what is being passed through the model. For this, we use a tensor board that visualises a 3D render of the embeddings using principal component analysis (PCA). This is shown in figure 12 below and concludes the analysis for this experiment.

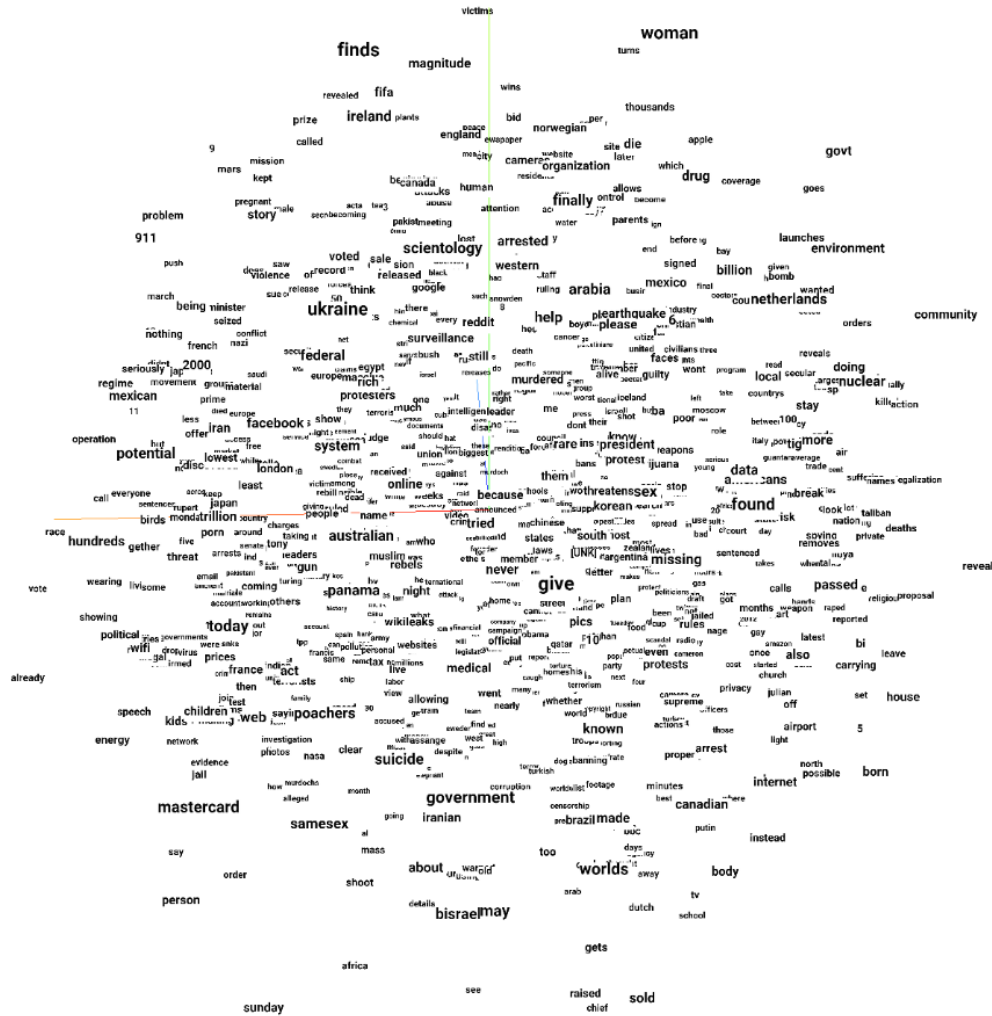


Figure 12: PCA of word Embedding in a 3D space.

4.3 Experiment 2: Architecture Change

For the second experiment, we changed the architecture. We now had five LSTM layers set at 64 or 32 units alternating. The first four layers use a call called return-sequences, which concatenate the output and pass it to the subsequent layer. All LSTM layers had dropout for aggressive regularisation to avoid overfitting where possible we also added another dense layer and dropout. Vocab size is set to 10,000 whereas, in the smaller model, vocab size was set to 1000. We introduced shuffling to create the testing and validation datasets and repeatability, creating the dataset again periodically.

Finally, we then add shuffling at the fitting stage of the model, which further shuffles the training data at the end of each epoch. The initial accuracy and loss metrics are shown below in figure 13:

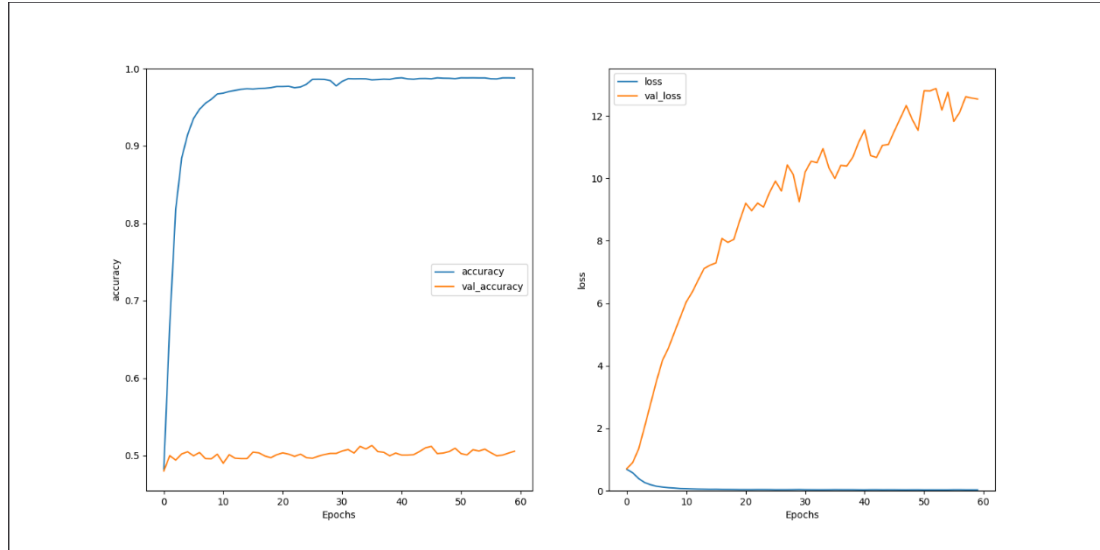


Figure 13: Accuracy and Loss metrics for the adjusted model.

The training accuracy initially went up before beginning to plateau. Furthermore, the validation accuracy crept up slightly. This can be explained when looking at the metrics on the left, there is a high degree of loss in the validation dataset, whilst the loss in the training data is decreasing. This likely means less and less data is being used to validate the training data as the epochs go on. We had already attempted to avoid this with aggressive regularisation via dropout layers. We concluded that the model's simplicity was to blame for not using the full potential for the provided data.

4.4 Experiment 3: Batch Size Variance

Accuracy wise, there was very little difference. All models of varying batch size had a testing accuracy of around 46%. However, training timewise, the batch size of 100 proved to be the best with a bit of overfitting from the first epoch. Batch size affects how quickly gradient descent occurs as training times over one epoch grew very quickly, and with testing accuracy, scores showing very little difference between each model. We felt it would be a fruitless endeavour to increase epoch number for all models to record similar accuracy scores. The results to these can be seen in figures 21 to 22 in the appendix for context.

4.5 Final Custom Model and Comparisons

Figure 14 shows the accuracy scores of the pre-build models and the custom model. The Textblob model performed best, which was surprising given that the custom model was trained on headline data before predicting sentiment. The other machine learning model, Flair was not trained on this domain specific data, so its poor

performance was to be expected. Although different in the case of the rule-based models, However, we had expected a much more significant gap in performance from the custom model. This highlights the difficulties of NLP in the financial space. Financial markets are highly stochastic, and some of the best machine learning based models have an accuracy of around ~65% (Liu, 2018).

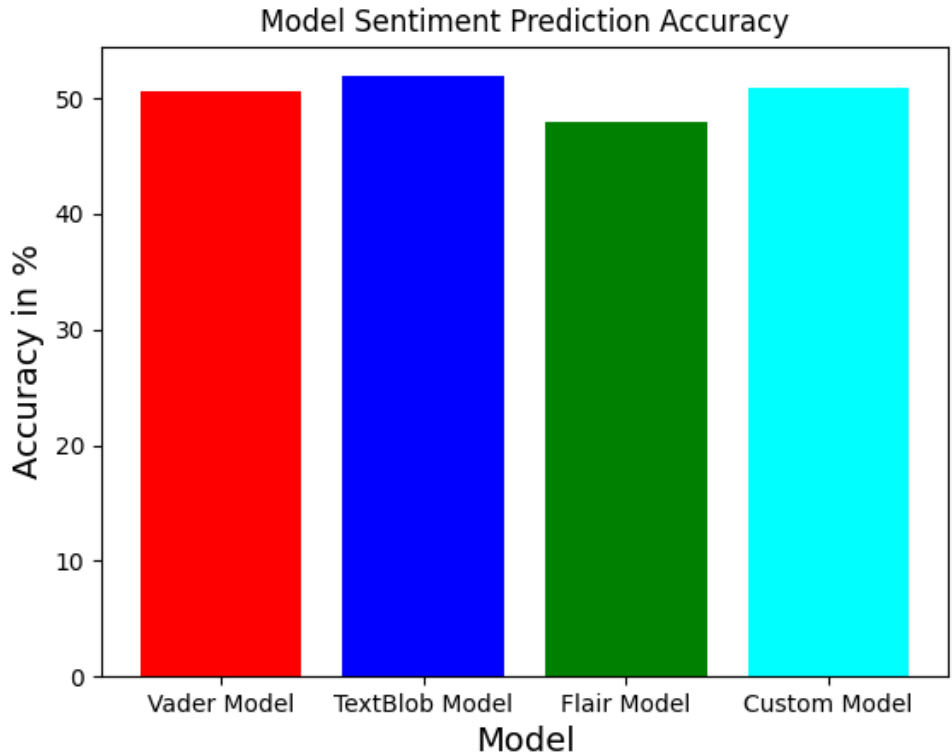


Figure 14: Model Accuracy Scores.

As mentioned above, the Flair model performed poorly, scoring 46%, categorising most of the headlines it read as negative as shown in figure 15 below:

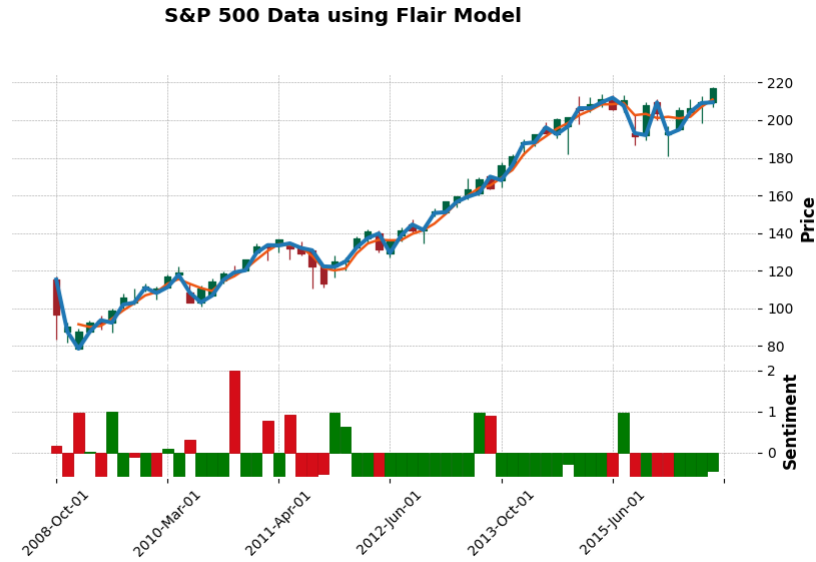


Figure 15: Flair Model sentiment Results on S&P 500 market graph.

The explanation for this is the lack of exposure to the headlines data as mentioned above. The headline data is domain specific. While particular words may be negative in movie reviews, much of the language used may not necessarily be so in the financial news headline. This, despite the means for deriving sentiment classifier being machine learning based. It is in this aspect that Vader and Textblob shine due to their rule-based nature; they can adapt.

Vader performed better, scoring 50.6%, which is likely due to its construction concerning social media text. Social media text varies wildly in language, domain area and subject matter. It is this breadth I believe allowed it to perform better compared to Flair, as can be seen in figure 14 and figure 16 in its mapping:

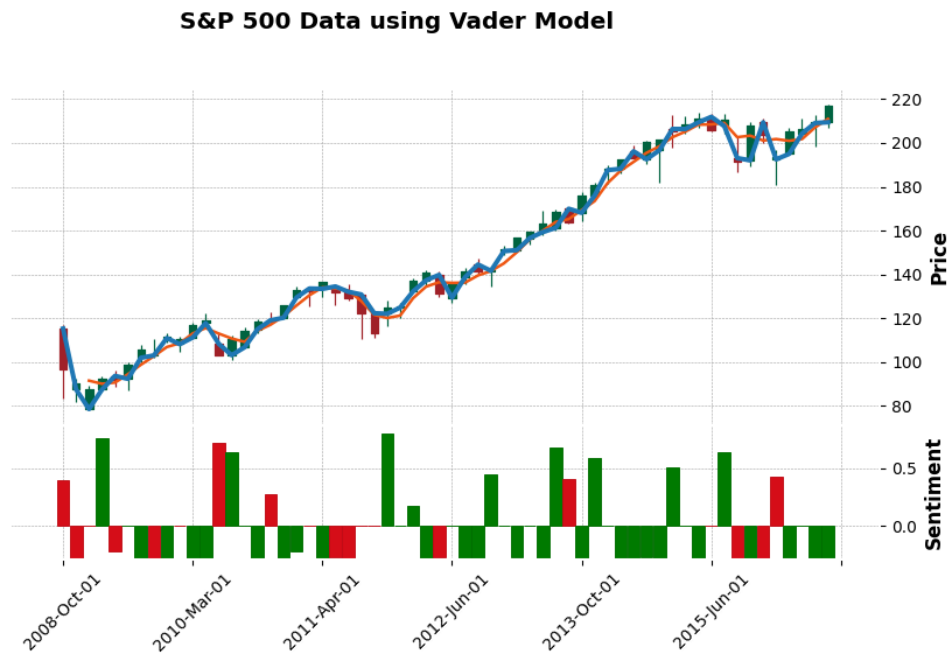


Figure 16: Vader Model sentiment Results on S&P 500 market graph.

Of the three pre-built models, Textblob proved to be the best, scoring an accuracy score of around 51%. We believe the maths based averaging approach is what carried it through to perform better despite being unfamiliar with the headline data and having a lexical database like the Vader model. Its performance can be seen below in figure 17:

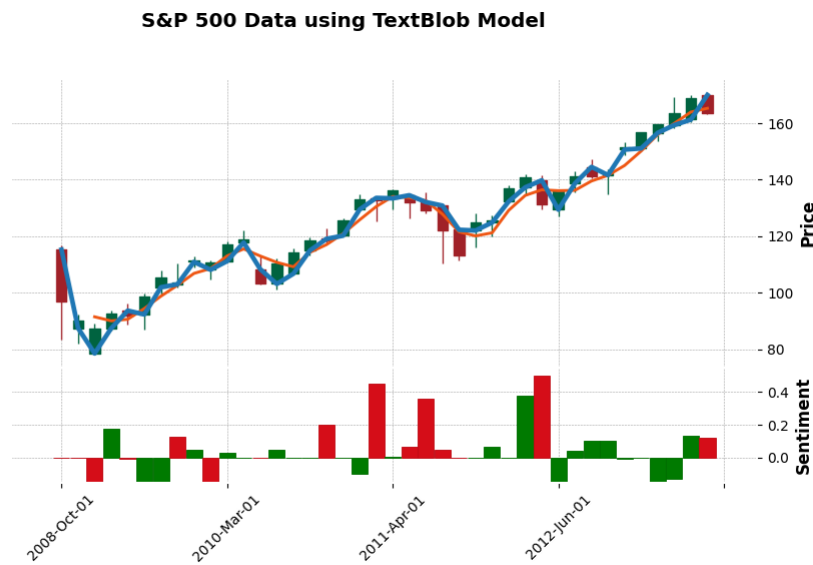


Figure 17: Textblob Model sentiment Results on S&P 500 market graph.

For the custom model, a batch size of 500 epoch number of 80. The model achieved a testing accuracy of 50.7%. Their mapped performance is shown in figure 18 below:

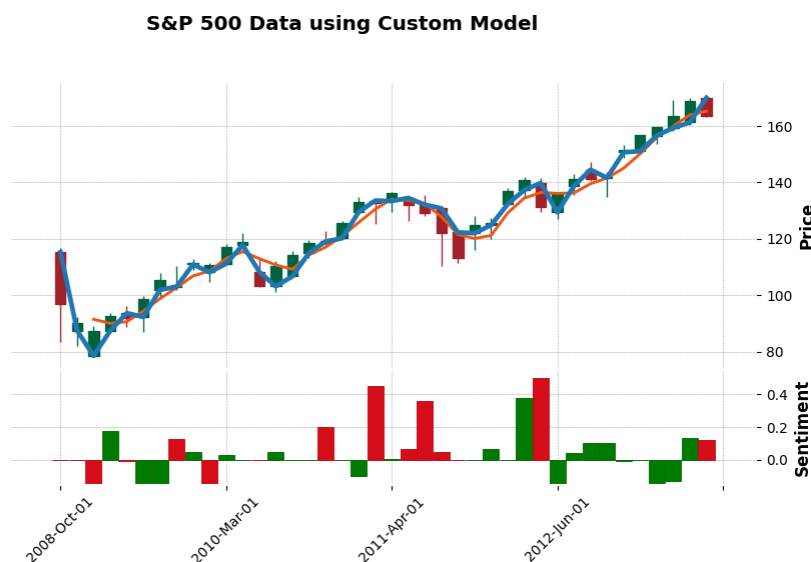


Figure 18: Custom Model sentiment Results on S&P 500 market graph.

The model derived sentiment that matched up with market movement better than the other machine learning model Flair, and did only slightly outperform Vader, but failed to beat Textblob. Whilst disappointing, we were able to identify two things that are needed to improve the model's accuracy:

- Far more data – Lui's paper we highlighted in chapter 3 utilised headline data from Reuters and Bloomberg (Liu, 2018). Upon further research, we found that over 15 million headlines were available for use in that paper's model; in contrast, we had roughly 48,000 in total.
- Model Complexity – the model we have created here, is too simple. A lot of data is being lost or wasted that could be fed into a more complex network. To illustrate, Liu's model in their paper made a custom word embedding module and concatenated the results with a CNN responsible for the character level embedding, vastly increasing representation within the subsequent layers of the model.

Chapter 5

Conclusion and Future Work

5.1 Comments on results

When reviewing our findings, we can see that our custom model outperformed the two out of the three pre-built models available but only slightly, also it failed to beat Textblob. Whilst disappointing, the overall experiment was successful in mapping and visualising the relationship between sentiment data and market data. It is evident that in areas where the sentiment polarity and intensity was high, the price movement followed accordingly. The interesting point was that the intensity did not need to be high in either negativity or positivity but concerning the sentiments directly surrounding that time slice.

The custom model could predict market movement, and this showed in the mapping in figure 18 above. Comparing the dataset size and simplicity of the model with more recent works, these results are promising. Nevertheless, we were also able to identify the weakness of the current model, which can be addressed in future works. Also, looking at the findings, there is clear overfitting occurring, but this is due to a lack of data. Responding to these successful results, we find that several goals have been achieved:

- Becoming familiar with Python, the TensorFlow library and the Keras API.
- Building a custom model that is capable of deriving sentiment from financial news headline data, with a degree of accuracy.
- Mapping the newly found sentiment data across on to financial market data to explore and discuss the relationship of headline data and market price movement.

A few extra goals were completed also:

- Visualise the word embedding structure that is fed into our custom network.
- Create three pre-build models to serve as controls and comparison.

5.2 Future Work

As mentioned previously, it is clear there is more work to do. In the future, we hope to achieve the following:

- Provide more data and train for far longer. As mentioned in chapter 5, Liu Huicheng's model used the Reuters dataset, which alone is about 8.5 million headlines. In contrast to our dataset, which was far too small. Furthermore, due to time constraints, we could not train for as long as we would have liked.
- A more complex model. For the vectorisation of the words, we could have either used Word2vec, a known word embedding tool or made a custom vectorising tool to represent the text better. Also, recent works have shown attention-based models to perform well in this NLP classification task. Finally, another approach we could have used is the inclusion of knowledge tree graphs shown to boost performance.
- Better Visualisations – we used mplfinance which was essentially matplotlib. In the future we could add another element to the model but using better visualisation tools such as seaborn or plotly dash which would give us a cleaner visualisation but also allow us to experiment with features such as real-time rendering.

5.3 Conclusion

In this project, we successfully implemented a custom model capable of deriving sentiment from headline data. Whilst results were not, what we had hoped, after reviewing experimental results and comparisons concerning the pre-build models, we are happy with what has been achieved so far. As a result of undertaking this project, we have come to understand the field of natural language processing better, understanding the complexities and obstacles to practical sentiment analysis and the difficulty of predicting stock price movement. A key benefit has been that our programming ability in Python has improved tremendously, helping us be more competent at programming in Python. Whilst we would have like to have solved key issues observed within the custom model, such as validation loss, given the time available for the project, we are ultimately happy with where the project has concluded.

Bibliography

- 3Blue1Brown. (2017, October 5). *Youtube: But What is a Neural Network? [Video]*. Retrieved from Youtube: <https://youtu.be/aircAruvnKk>
- BBC News. (2020, March 25). *US markets close higher after emergency virus deal*. Retrieved from BBC News: <https://www.bbc.co.uk/news/business-52030034>
- Brownlee, J. (2017, October 9). *A Gentle Introduction to the Bag-of-Words*. Retrieved from Machine Learning Mastery: <https://machinelearningmastery.com/gentle-introduction-bag-words-model/>
- C.J.Hutto, E. G. (2015). VADER: A Parsimonious Rule-based Model for. *Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media*. Ann Arbor, MI.
- Cambridge Dictionary. (n.d). *Fake News Definition*. Retrieved April 7, 2021, from <https://dictionary.cambridge.org/dictionary/english/fake-news>
- Corporate Financial Institute . (2015). *Financial Markets* . Retrieved from Corporate Financial Institute: <https://corporatefinanceinstitute.com/resources/knowledge/trading-investing/financial-markets/#:~:text=Financial%20markets%2C%20from%20the%20name,of%20assets%20such%20as%20bonds&text=Simply%20put%2C%20businesses%20and%20investors,to%20make%20more%20money%20>
- Deep AI . (n.d, n.d n.d). *Deep AI: What is a Recurrent Neural Network* . Retrieved from Deep AI : <https://deepai.org/machine-learning-glossary-and-terms/recurrent-neural-network>
- Deep AI. (n.d, n.d n.d). *Deep AI: What is a Feed Forward Network*. Retrieved from Deep AI: <https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network>
- Garbade, D. M. (2018, October 15). *A simple introduction to Natural Language Processing*. Retrieved from Becoming AI : <https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32>
- Geeks For Geeks . (2019, August 26). *CNN Introduction to Pooling Layer*. Retrieved December 10, 2020, from <https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/>
- Goel, A. M. (2011). *Stock Prediction Using Twitter Sentiment Analysis*. n.a, n.a.
- Goldberg, Y. (2017). *Neural Networks for Natural Language Processing* . California: Morgan & Claypool Publishers.
- Google. (2020, 12 08). *TensorFlow*. Retrieved 12 10, 2020, from https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D
- Google. (n.d.). *Google Developers: Clustering In Machine Learning*. (Google) Retrieved November 01, 2020, from <https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages>

- Google TensorFlow. (2021, April 2). *Text classification with an RNN*. Retrieved from TensorFlow:
https://www.tensorflow.org/tutorials/text/text_classification_rnn
- Haghian, P. Z. (2019). Deep Representation Learning and. *n/a, n/a(n/a), n/a*.
- Haghian, P. Z. (2019). Deep Representation Learning and Prediction for Forest Wildfires. *n/a, n/a*.
- Hardesty, L. (2017, April 14). *Explained: Neural networks: MIT News*. Retrieved from MIT News: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>
- Heller, M. (2019, January 28). *InfoWorld: What is Keras? The deep neural network API explained*. Retrieved from InfoWorld:
<https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html>
- IBM. (2020). *Recurrent Neural Networks*. Retrieved April 19, 2021, from
<https://www.ibm.com/cloud/learn/recurrent-neural-networks>
- Imran, M. (2012). An Overview of Particle Swarm Optimization Variants. *Malaysian Technical Universities Conference on Engineering & Technology 2012, MUCET 2012*. Kuala lumpur.
- James Chen, G. S. (2020, September 12). *Forex Trading: A beginner's guide Investopedia* . Retrieved from Investopedia :
<https://www.investopedia.com/articles/forex/11/why-trade-forex.asp#what-is-the-forex-market>
- Jeffrey Pennington, R. S. (2014). GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543). Doha: Association for Computational Linguistics.
- Kohli, P. P. (n.d, n.d n.d). *Sentiment Analysis - Methods and Pre-Trained Models Review*. Retrieved from Pahul Preet Singh Kohli:
<https://pahulpreet86.github.io/sentiment-analysis-methods-and-pre-trained-models-review/>
- Liu, H. (2018, November 15). *Leveraging Financial News for Stock Trend Prediction with Attention-Based Recurrent Neural Network*. Retrieved April 23, 2021, from <https://arxiv.org/abs/1811.06173v1>
- Machine Learning Mastery . (2018, December 3). *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks* . Retrieved December 10, 2020, from <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>
- Mohammad-Hossein Golbon-Haghighi, H. S.-M. (2018). Pattern Synthesis for the Cylindrical Polarimetric Phased Array Radar . *Progress In Electromagnetics Research M*, 66(N/A), 87-98.
- Olena. (2018, February 8). *Medium: GPU vs CPU Computing: What to choose?* Retrieved December 10, 2020, from <https://medium.com/altumea/gpu-vs-cpu-computing-what-to-choose-a9788a2370c4>

- SAS. (n.d, n.d n.d). *SAS Insights: Analytics and Data Science Insights: Natural Language Processing (NLP) what it is and why it matters*. Retrieved from SAS: https://www.sas.com/en_gb/insights/analytics/what-is-natural-language-processing-nlp.html#howitworks
- Sharma, S. (2017, September 9). *Towards Data Science: What the Hell is a Perceptron?* Retrieved from Towards Data Science : <https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53>
- Tomas Mikolov, K. C. (2013). Efficient Estimation of Word Representations in vector space. *n.a*, n.a.
- Valkov, V. (2017, May 25). *Medium: Making a Predictive Keyboard using Recurrent Neural Networks — TensorFlow for Hackers (Part V)*. Retrieved from Medium: <https://medium.com/@curiously/making-a-predictive-keyboard-using-recurrent-neural-networks-tensorflow-for-hackers-part-v-3f238d824218>
- Venkatachalam, M. (2019, March 1). *Recurrent Neural Networks* . (Towards Data Science) Retrieved April 23, 2021, from <https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce>
- Wikipedia . (n.d, n.d n.d). *wikipedia: Boids*. Retrieved November 02, 2020, from <https://en.wikipedia.org/wiki/Boids>
- Wikipedia . (n.d, n.d n.d). *Wikipedia: Long Short Term Memory*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Long_short-term_memory
- Wikipedia. (n.d, n.d n.d). *Wikipedia: Artificial Neural Networks* . Retrieved from Wikipedia : https://en.wikipedia.org/wiki/Artificial_neural_network#/media/File:Colored_neural_network.svg
- Wood, T. (n.d, n.d n.d). *Deep AI: Sigmoid Function* . Retrieved from Deep AI: <https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function>
- Xin Du, K. T.-I. (2020). Stock Embeddings Acquired from News Articles and Price History, and. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 3353–3363.

Appendix A

Batch Size Model Results

```
Model: "sequential"
-----
Layer (type)                Output Shape              Param #
-----
text_vectorization (TextVect (None, None)          0
-----
embedding (Embedding)       (None, None, 64)         640000
-----
bidirectional (Bidirectional (None, None, 128)     66048
-----
bidirectional_1 (Bidirection (None, None, 64)      41216
-----
bidirectional_2 (Bidirection (None, 32)           10368
-----
dense (Dense)               (None, 64)               2112
-----
dropout (Dropout)          (None, 64)               0
-----
dense_1 (Dense)             (None, 1)                65
-----
Total params: 759,809
Trainable params: 759,809
Non-trainable params: 0
-----
438/438 [=====] - 43s 99ms/step - loss: 0.6907 - accuracy: 0.4630
Test Loss: 0.6907419562339783
Test Accuracy: 0.4629798829555115
[[0.10918449]]
```

Figure 19: Model with a batch size of 10

```
Model: "sequential"
-----
Layer (type)                Output Shape              Param #
-----
text_vectorization (TextVect (None, None)          0
-----
embedding (Embedding)       (None, None, 64)         640000
-----
bidirectional (Bidirectional (None, None, 128)     66048
-----
bidirectional_1 (Bidirection (None, None, 64)      41216
-----
bidirectional_2 (Bidirection (None, 32)           10368
-----
dense (Dense)               (None, 64)               2112
-----
dropout (Dropout)          (None, 64)               0
-----
dense_1 (Dense)             (None, 1)                65
-----
Total params: 759,809
Trainable params: 759,809
Non-trainable params: 0
-----
176/176 [=====] - 20s 116ms/step - loss: 0.6904 - accuracy: 0.4630
Test Loss: 0.6903899908065796
Test Accuracy: 0.4629798829555115
[[0.14783809]]
```

Figure 20: Model with a batch size of 25


```

Model: "sequential"
-----
Layer (type)                Output Shape                Param #
-----
text_vectorization (TextVect (None, None)                0
-----
embedding (Embedding)        (None, None, 64)           640000
-----
bidirectional (Bidirectional (None, None, 128)           66048
-----
bidirectional_1 (Bidirection (None, None, 64)            41216
-----
bidirectional_2 (Bidirection (None, 32)                  10368
-----
dense (Dense)                (None, 64)                  2112
-----
dropout (Dropout)            (None, 64)                   0
-----
dense_1 (Dense)              (None, 1)                    65
-----
Total params: 759,809
Trainable params: 759,809
Non-trainable params: 0
-----
88/88 [=====] - 11s 124ms/step - loss: 0.6906 - accuracy: 0.4630
Test Loss: 0.6905981302261353
Test Accuracy: 0.4629798829555115
[[0.12223405]]

```

Figure 21:: Model with a batch size of 50

```

Model: "sequential"
-----
Layer (type)                Output Shape                Param #
-----
text_vectorization (TextVect (None, None)                0
-----
embedding (Embedding)        (None, None, 64)           640000
-----
bidirectional (Bidirectional (None, None, 128)           66048
-----
bidirectional_1 (Bidirection (None, None, 64)            41216
-----
bidirectional_2 (Bidirection (None, 32)                  10368
-----
dense (Dense)                (None, 64)                  2112
-----
dropout (Dropout)            (None, 64)                   0
-----
dense_1 (Dense)              (None, 1)                    65
-----
Total params: 759,809
Trainable params: 759,809
Non-trainable params: 0
-----
44/44 [=====] - 6s 131ms/step - loss: 0.6906 - accuracy: 0.4630
Test Loss: 0.6905657052993774
Test Accuracy: 0.4629798829555115
[[0.13622774]]

```