Predicting Stock Price Movement Using Neural Networks and Sentiment Analysis of Social Media posts

Robert Turff - 1822011
MSci Computer Science Project
University of Birmingham
Under Supervision of
Dr Harish Tayyar Madabushi

Word Count: 11454

May 17, 2021

(This page had been intentionally left blank)

1 Abstract

Past attempts to solve the problem of predicting stock prices are typically based around the analysis of historical stock price data, many of these also attempt to combine this with social media sentiment from a single source, however, very few if any go as far to include sentiment data from multiple sources. This project evaluates the use of two popular sentiment analysis algorithms, namely VADER and RoBERTa, in order to derive the sentiment of posts from the social media sites Stocktwits and Twitter. It then explores the use of this information as inputs to various neural network models so as to make intraday stock price predictions. Initially this was framed as a regression problem to directly predict one day ahead close prices of a particular stock such as Tesla. However later this changed to predicting the simple moving average across a two day window of the closing price, which was found to allow for much more accurate predictions. The results of the project have shown that the use of sentiment derived features reduces the RMSE loss of stock price predictions by roughly 25% with RoBERTa being the best method of providing that sentiment data. We also present a neural network architecture based on uni-directional LSTMs and CNNs which was best able to use this information. These findings present the potential for future study into the combination of further sources of sentiment related to stock prices along with more expansive exploration of sentiment analysis models in order to achieve better classification performance.

Contents

2 Introduction

The prediction of stock prices is a popular area of research since achieving even slightly better than random chance prediction's can help investors make more informed decisions resulting in higher returns. However, the stock market is notoriously difficult to predict due to the high volatility and the sheer number of factors that can affect stock prices. Recent examples of this are the rising bond yields in the US which caused a significant drop in value of US stocks or conversely the announcement of stimulus checks, again in the US, which had the opposite effect. There are also much more unpredictable elements at play such as a CEO posting on social media that the share price of their own company is too high promptly causing it to drop 10% in a matter of minutes.

There have been many different approaches to tackling this challenging problem, more recently success has been had with the application of neural networks based models (Mehtab and Sen 2020) (Althelaya, El-Alfy, and Mohammed 2018) due to their ability to pick out and model complex patterns in data that aren't obvious to the human eye. Couple this with the fact that many retail investors i.e. non professionals, potentially don't have the time and or experience to pick out the small details that can significantly effect a stocks performance, meaning that these models could provide valuable information to such investors so as to make better investment decisions.

Sentiment analysis is another technique that has been employed in tandem with neural networks or machine learning algorithms, such as support vector machines (SVMs). This has been used in papers, such as (K.Nirmaladevi and Krishnamoorthy 2019), to classify the sentiment of social media posts, which is then inputted in some format as features to the SVM, in this case in order to classify up/down price movements. The potential impact social media can have of company's share prices was highlighted recently when Reddit users collaborated through posts on the subreddit r/wallstreetbets to perform a short squeeze, which resulted in the share price of Gamestop (GME) rising several hundred percent over the course of a few weeks. Hence, capturing this sort of information has the potential to provide powerful indicators of future stock price changes.

This project aims to combine social media sentiment from several sources along with historical price data to use as features for a neural network model, the goal of this model then being to make predictions that can be used by investors to make better investment decisions. As part of this, the project will look into the most accurate method for classifying social media sentiment as well as evaluating the best neural network architecture for making the final predictions. The project will focus on the use of intraday (e.g. hourly or bi-hourly) price data to make short term one day ahead predictions so as give higher resolution results and make better use of the sentiment data available.

3 Background

This section of the report will give some background information on how the various models and techniques that make up the project function.

3.1 Artificial Neural Networks

Neural networks have been shown, as discussed later in section 4, to perform well for the task of stock prediction. To begin, we introduce these at the base level, specifically the Artificial Neural Network (ANN) which loosely attempt to mimic the structure of the human brain by modeling layers of connected neurons with weighted inputs/outputs which can be adjusted in order to train/learn a task. The simplest form of this would just be a layer of output neurons which output a weighted combination of the inputs, this is also an example of a feed forward neural network in that the outputs of layers only feed into subsequent layers and do not feedback into previous ones forming loops. Layers of neurons known as hidden layers can be added in between the input and outputs layers increasing the complexity of the network. The weights within the network are updated by a processes known as backpropagation wherein the outputs of the network are passed into a loss function such as root mean square error (RMSE). The gradient of this loss with respect to the weights can be calculated for each layer, weights are then subsequently adjusted in order to minimise this loss value.

3.1.1 Recurrent Neural Networks

Recurrent neural networks (RNNs) are neural networks which, unlike feed forward neural networks, contain neurons that feedback to neurons in the same or previous layers as inputs. The idea behind this being to allow some sort of retention of information between sequential inputs. This is a very desirable trait for time series tasks, such as stock price prediction, as it can be assumed that the input data at previous time steps is still linked to the desired result at later time steps.

$$h_t^l = tanh\left(W^l\left(\frac{h_t^{l-1}}{h_{t-1}^l}\right)\right) \tag{1}$$

Equation 1 shows how the output state h can be calculated for layer l and time step t where for each layer a unique weight/parameter matrix W^l is applied to the output state from the last layer h_t^{l-1} and the last time step h_{t-1}^l (Karpathy, Johnson, and Fei-Fei 2015). The weights in RNNs are adjusted slightly differently compared to feed-forward neural networks, this process is known as backpropogation through time. This process differs from the original

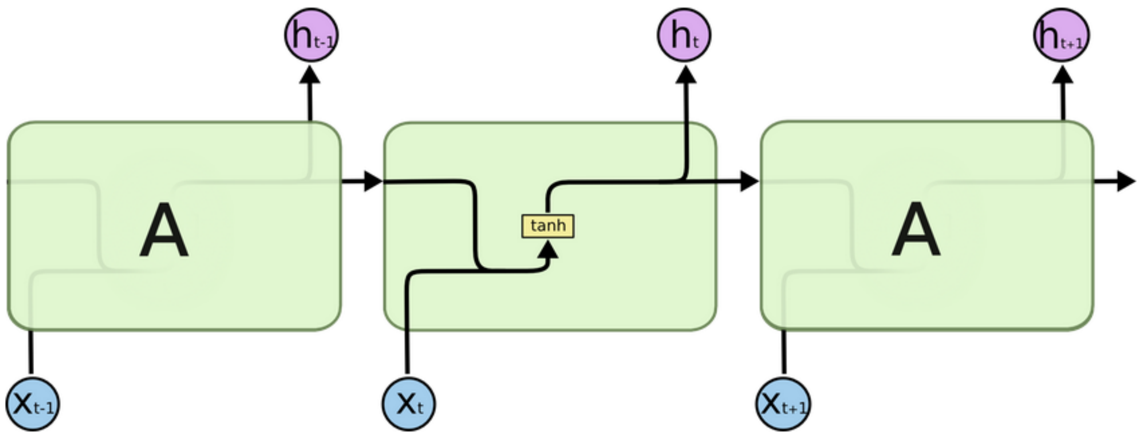


Figure 2: Repeating module of an RNN where the output of the network for time step $t-1$ is fed back into the network and combined within to produce the output for the next time step (*Understanding LSTM Networks – Colah's Blog* 2021)

in that since the output at one time step relies upon the previous step, one must run the backpropogation sequentially through the time steps. This exacerbates the problems of exploding and vanishing gradients as now the loss has to be propagated backwards through each layer and each time step during which the loss gradient has a tendency to exponentially increase (exploding gradient) or decrease (vanishing gradient).

3.1.2 Long-Short Term Memory

Long-short term memory neural networks were designed as a variant of RNNs to address the issue of the vanishing gradient problem. RNNs in particular suffer from this as the tanh activation function shown in equation 1 produces gradients between 0 and 1 which, when multiplied together during backpropogation, produce smaller and smaller values. This is mitigated though not entirely solved through the addition of 3 gates and an extra cell state passed between recurring modules. The goal of the extra cell state is to retain long term information about the task while the hidden state keeps the short term information, the gates then control which information from these steps is kept/used for later predictions. The purpose of each of these gates is detailed below, although ultimately they result in the gradients that vanish slower.

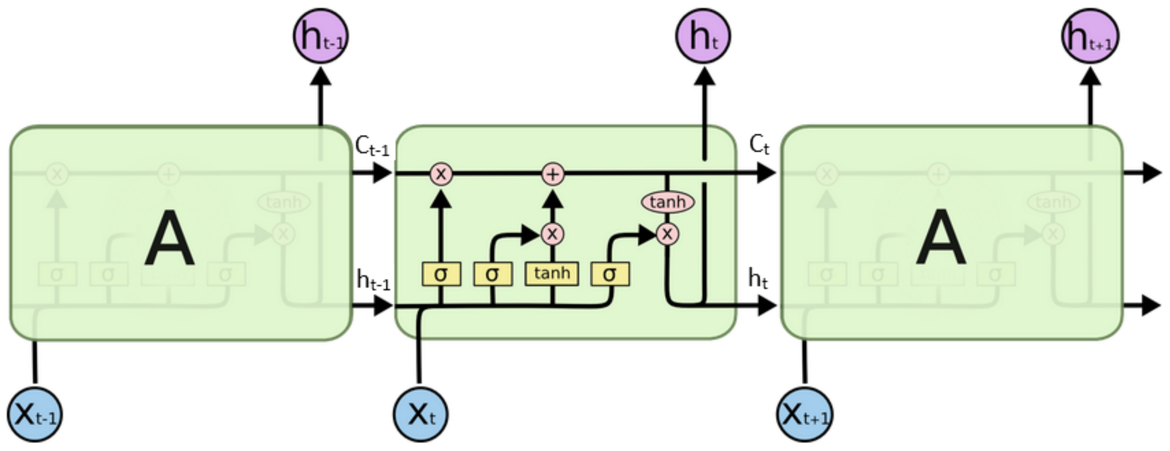


Figure 3: Repeating module of an LSTM where h_{t-1} is the hidden layer state at time step $t-1$, C_{t-1} is the cell state and X_t the input. (*Understanding LSTM Networks – Colah's Blog* 2021)

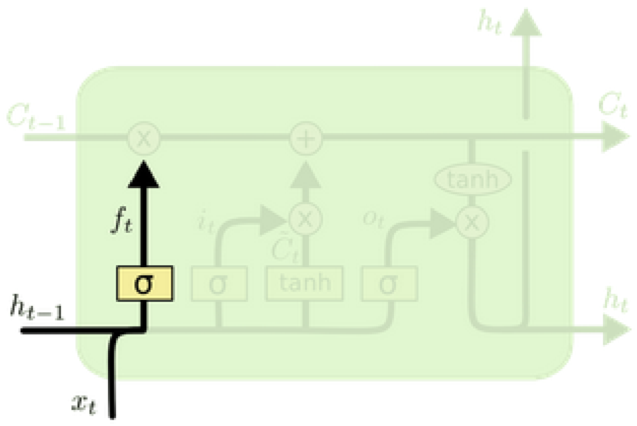


Figure 4: Forget gate of an LSTM where f_t is the forget gate output at time step t produced by passing the previous hidden layer state h_{t-1} combined with the current input x_t through a sigmoid layer σ which maps the inputs to numbers between 0 and 1. (*Understanding LSTM Networks – Colah's Blog* 2021)

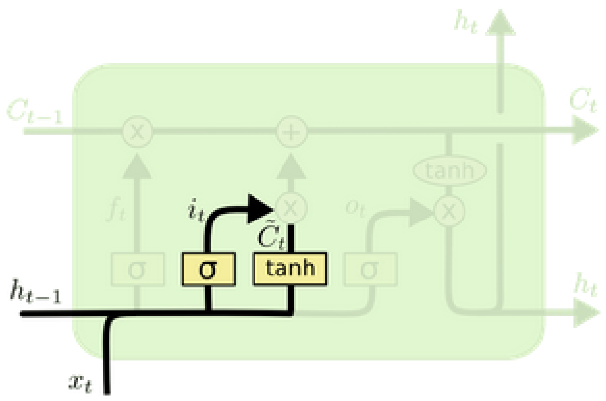


Figure 5: Input gate section of an LSTM where i_t is the input gate output formed by passing h_{t-1} and x_t through a sigmoid function. $\tilde{C}_t$ are the potential updates to the cell state produced by passing h_{t-1} and x_t through a tanh function which maps inputs to values between -1 and 1. (*Understanding LSTM Networks – Colah's Blog* 2021)

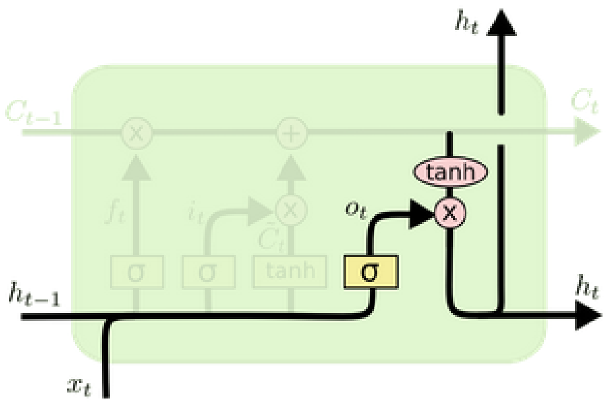


Figure 6: Output section of an LSTM where o_t is the output gate and h_t is the output/new hidden state formed by the element wise multiplication of the output gate with tanh of the cell state C_t. (*Understanding LSTM Networks – Colah's Blog* 2021)

Figure 4 shows the first of these gates, the forget gate, the output of which is multiplied element wise with the cell state from the previous time step. This in effect chooses how much of the long term memory of the network should be kept for the current time step where an output of zero for an element of f_t would result in the corresponding element in the C_t being completely forgotten and vice versa for an output of one. Next, the input gate in figure 5 decides what information from the short term memory h_{t-1} and input x_t should be added to form the new cell state C_t. Finally the output gate in figure 6 filters stored cell state to produce the output of the LSTM cell for that time step.

3.1.3 Bidirectional Long-Short Term Memory

Bi-directional LSTMs are a form of LSTM in which an additional backwards layer is added which is identical to that of the single uni-directional layer except that the inputs are passed in reverse, the output states of the two layers are then combined for example by concatenating the two in order to produce the final output. The purpose of this is to allow such a network to learn from the input data in both directions, potentially giving it a greater understanding of the context of each data point.

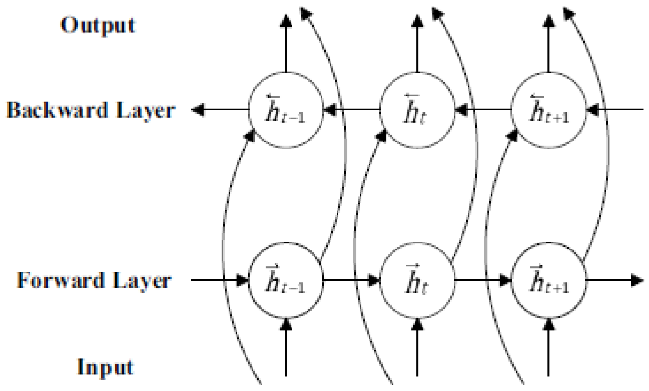

Output
Backward Layer
Forward Layer
Input
$\overleftarrow{h}_{t-1}$
$\overleftarrow{h}_{t}$
$\overleftarrow{h}_{t+1}$
$\overrightarrow{h}_{t-1}$
$\overrightarrow{h}_{t}$
$\overrightarrow{h}_{t+1}$

Figure 7: Bi-directional LSTM (*INTERSPEECH 2014 Abstract: Fan et Al.* 2021)

3.1.4 Convolutional Neural Network

Convolution Neural Networks or CNNs are typically used for image processing however they have also been successfully applied to make time series predictions. Their function is split into two main components (Mehtab and Sen 2020): the first of these is the convolutional layer wherein a set of learn-able image filters, i.e. a matrix (or array if applying a 1 dimensional convolution) of values that weight values surrounding a particular data point, are applied to a set of 2d input "images" formed from the input features and time steps. This is used to generate a set of feature maps which represent different interpretations of the input data. The second main component is the pooling stage which is used to extract the most important information from the generated feature maps, the max pooling function is an example of this which extracts the maximum value from within it's filter/kernel. These two steps can be repeated several times before the final output can be generate by passing the outputs of the final layer through a dense/linear layer.

3.2 Transformers

The Transformer (Vaswani et al. 2017) is a neural network architecture built for sequence-to-sequence problems such as language translation, which, similarly to recurrent neural network (RNN) based approaches, employs an encoder-decoder architecture. The key difference between the two is that the Transformer model uses an attention mechanism rather than recurrent inputs to handle the sequence data. This means that it can be run in parallel thus massively improving computational efficiency allowing to be trained on much larger data-sets.

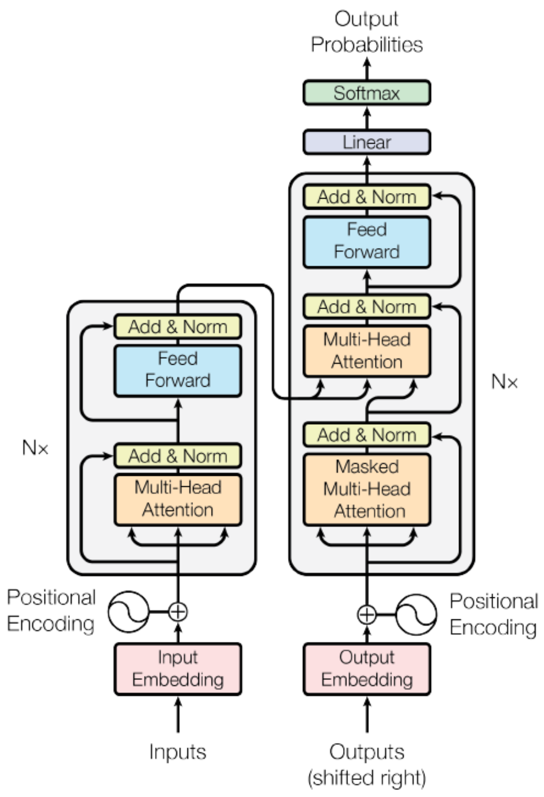


Figure 8: Transformer model architecture (Vaswani et al. 2017) Encoder (left), Decoder (right)

The full architecture is shown in figure 8, however in this section we will focus specifically on the workings of the encoder stage as this is the part employed by later language models for state of the art performance on various machine learning tasks. The encoder stage is made up of several parts outlined in the following sections:

3.2.1 Input and Positional Embeddings

The input embeddings are simply vectorised numerical forms of the tokens making up the input text, these are then positionally encoded to preserve the locations of each token within the sentence since each word isn't inputted sequentially.

3.2.2 Attention

For each token in the original embedding, an attention vector is computed which encodes how important/relevant each other token is in the input to it, this effectively captures the context between tokens within the input. Equation 2 is used to calculate these attention vectors:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2}$$

Where:
Q = matrix of query vectors
K and V = matrices of key value pairs
d_k = Number of dimensions of vectors within matrix K Q, K and V are derived by applying different weights to the original input.

3.2.3 Multi-Head Attention

Multi-head attention produces multiple attention vectors for each token based on a different set of weight vectors being applied to the inputs (V, K and Q) in the attention calculation in section 3.2.2, this lets the architecture capture different levels of context between tokens which are then weighted to form a single vector again for each token which is passed to the feed forward neural network layer. This process is illustrated in figure 9.

3.2.4 Feed Forward Layer

This step is just a simple feed forward neural network which maps the attention layers from the previous layer to an output used by either the decoder stage or fed back to the next encoder in the stack.

3.3 BERT

There are two strategies for applying pre-trained language representations to down stream tasks, feature-based and fine-turning approaches. The first of these uses a pre-trained model

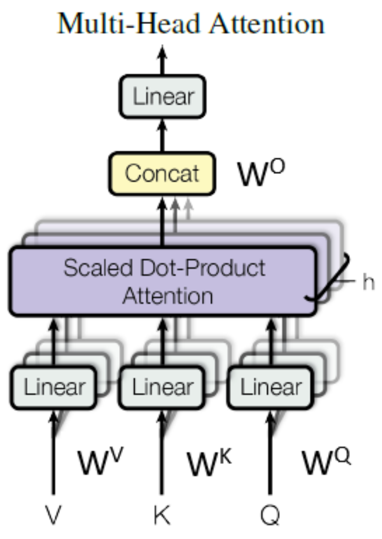


Figure 9: Multi-Head Attention Visualised(Vaswani et al. 2017)

to produce the embeddings which are then used as features for a task specific architecture. The second involves fine-tuning the parameters of the pre-trained model alongside a minimal number of task specific parameters for that downstream task, such as sentiment analysis. BERT or Bidirectional Encoder Representations from Transformers (Devlin et al. 2019) is an example of the fine-tuning approach, which, as the name suggests, makes use of the Transformer model described in section 3.2 by stacking multiple encoder layers on top of each other.

There are two main steps to the BERT framework, pre-training and fine-tuning. The first stage, pre-training, aims to teach the model a general understanding of language i.e. the context of words in relation to each other. In order to achieve this, BERT uses an unsupervised learning task with the objective of a Masked Language Modeling (MLM) i.e. randomly mask a number of tokens in the text and then attempt to predict the original word. The purpose of this is to avoid the issue other similar models have of only understanding the unidirectional context of a text by including both the left and right context around the masked words. There is also a secondary objective of next sentence prediction (NSP), the model is trained to minimise the combined loss between these two tasks.

The fine-tuning stage, in the case of a classification downstream task such as sentiment analysis, simply involves the addition of a classification layer to the output class label (CLS) which is a special token added at the start of the text being classified. The model is then trained on a task specific data-set so that the output embedding of the CLS token now encapsulates information on the predicted sentiment of the input sequence, this process is shown in figure 10.

Both of these steps are used to train the weights used for multi headed attention and within the feed forward neural network of the encoder stage of the transformer architecture.

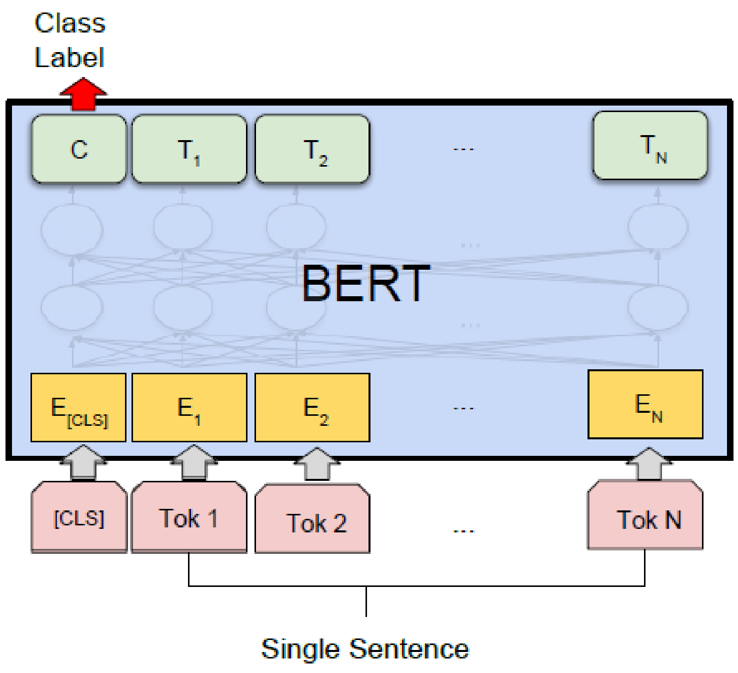


Figure 10: BERT Fine-tuning for Classification Tasks (Devlin et al. 2019)

3.3.1 RoBERTa

RoBERTa or Robustly optimized BERT approach (Liu et al. 2019), is a optimised version of the BERT model which effectively extends the training procedure by using a much larger data-set and larger batch sizes. They also remove the next sentence prediction task from the model and adjust the masking strategy for the MLM task so that different combinations of masked words in a sentence are used for training (Dynamic masking).

3.4 VADER

Valence Aware Dictionary and sEntiment Reasoner (VADER) (Hutto and Gilbert 2014) is a lexicon based tool for sentiment analysis which unlike previously mentioned methods, holds no knowledge of the context of words in the text. Instead lexicon based methods have a positive/negative weight assigned to known words in it's lexicon and simply sum these weights for all words that occur in the text. It also considers some hand crafted rules relating to negation, punctuation and capitalisation. Despite the relatively simple approach to the problem, VADER does show relatively good performance particularly on social media posts as discussed in section 4 of this report.

3.5 Technical Indicators

Technical indicators are a group of commonly used tools for market analysis which refer to various signals that can be derived from patterns in past/present data. This project only considers a small subset of these which were outlined below.

3.5.1 Simple Moving Average

One of the most common and simplest of these is the simple moving average (SMA) which is just the average closing price of the past n number of days.

3.5.2 Bollinger Bands

Bollinger bands are a pair of bands defined as being two standard deviations either side of the simple moving average of the closing prices and are typically used to identify whether a stock is overbought or oversold, an illustration of these bands is shown in figure 11.

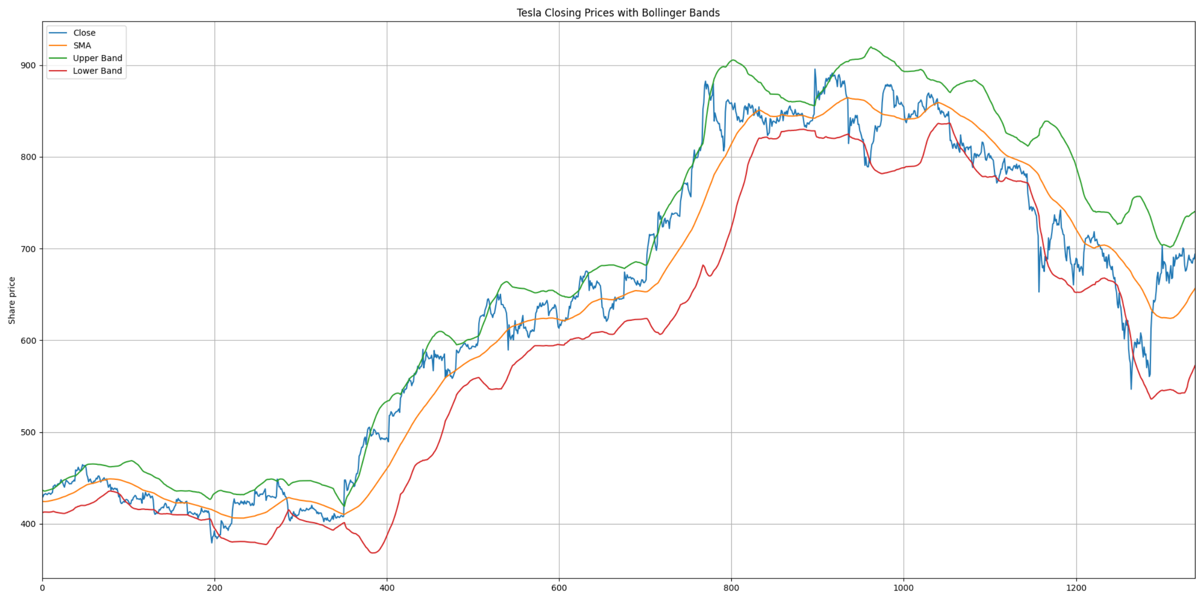


Figure 11: Bollinger bands & SMA chart for Tesla over 30 minute time intervals (x-axis)

4 Related Work

Stock prediction has consistently been a hot topic of interest for research despite the long standing theory of the Efficient Market Hypothesis (EMH) (Fama 1965) which suggests that share prices always reflect all available information thus implying that it's impossible to predict future stock prices. Many studies have however disputed this theory, such as (Bollen, Mao, and Zeng 2011), which found that public mood data on social media significantly improved the prediction accuracy for closing prices of a US index well past pure chance. (Mittal and Goel 2012) builds upon this by testing different sentiment analysis techniques and running the results through a neural network, the results of this confirmed the success of the previous paper albeit with slightly lower accuracy. These early papers give fairly concrete evidence against EMH while also showing how useful a tool both the sentiment of social media and neural networks can be for making these predictions.

Later papers such as (Persio and Honchar 2016) explored the effectiveness of applying different neural network models such as the Convolutional Neural Networks (CNNs) and Long-Short Term Memory (LSTM) recurrent neural networks to stock prediction, both of which showed promising results. This work also went further by testing an ensemble of these two models which resulted in significantly improved predictive performance as compared to the original standalone models. Another approach tested in (Mehtab and Sen 2020) attempted to combine LSTMs and CNNs sequentially as opposed to in parallel and actually found that the performance was worse than just applying these models individually. Further work into the use of LSTM's for stock prediction (Althelaya, El-Alfy, and Mohammed 2018) compared the base LSTM model to Bi-directional LSTMs (BLSTMs) as well as stacked unidirectional LSTM models and found that the BLSTM model outperformed both other models by a considerable margin.

There has also been considerable work on the sentiment analysis side, (Sohangir, Petty, and Wang 2018) in particular evaluates the performance of various machine learning such as SVM's along with lexicon based approaches to tagging Stocktwits posts. It concludes that VADER, a lexicon based approach, achieves the best performance. It is important to note however that in the paper the performance metrics for the VADER didn't include results it classified as "neutral" sentiment which were about a third of the total posts whereas the machine learning algorithms only predicted two classes "positive" and "negative". This has the unfortunate effect of obscuring the true results making it difficult to make a proper comparison between the two types of method. (Renault 2020) similarly investigates the accuracy of machine learning approaches to sentiment analysis, however it also goes on to look at the correlation between user sentiment on one day and share prices on the following day and despite finding that the two are correlated didn't find it useful for actually making stock predictions with daily data. This paper does however achieve comparatively low performance for sentiment analysis at roughly 75% accuracy, as compared to over 80% for the machine learning approaches tested in (Sohangir, Petty, and Wang 2018) which could have a significant impact on the validity of that final point. On the other hand, a previous paper by the same author (Renault 2017) did find evidence that investor sentiment on social media sites such as Stocktwits does in fact aid in intraday stock prediction and hence this report will focus on a similar resolution of time steps as opposed to using daily changes in prices/sentiment etc.

More recently language models based on the Transformer architecture such as RoBERTa (Liu et al. 2019) have shown state of the art results for NLP tasks, however since these are transfer learning models they are typically evaluated on general domain data sets such as GLUE. This makes it very difficult to compare this sort of model's performance on sentiment analysis to VADER's for example, especially considering that the domain of the data being investigated i.e. financial social media posts, will heavily effect their respective performance due to changes in language use as studied in (Qudar and Mago 2020) and (Oliveira, Cortez, and Areal 2016). Therefore this is something I will investigate in the report.

5 Methodology

This section will first cover the what and the why of the main steps this project took starting with the data collected before moving onto the testing of sentiment analysis algorithms and finally the neural network stock prediction model.

5.1 Data Collection and Processing

The data used in this project can be separated into two main groups, financial data which includes information on stock prices, volumes etc and social media data that consists of data relating to posts from the sites Stocktwits and Twitter. The specifics of the collection, contents, processing and selection of this data is presented in the following subsections.

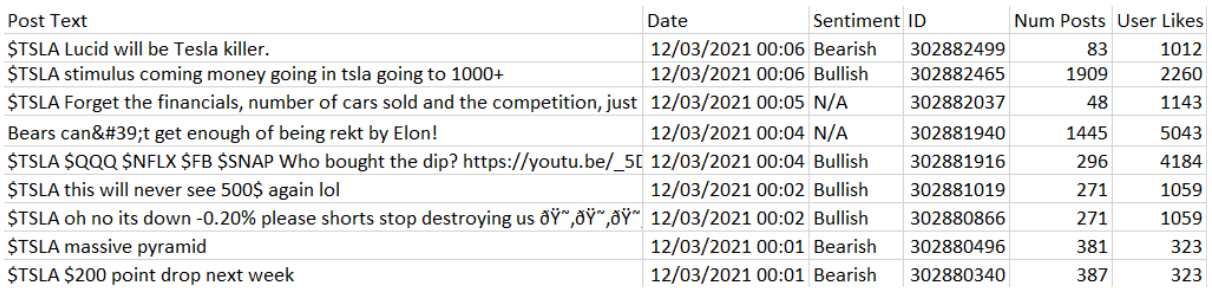
5.1.1 Financial Data

The financial data used in this project was collected using the yfinance python library which mimics the tools offered by the discontinued Yahoo! Finance API by scraping data off the Yahoo Finance website. The data collected consisted of open, high, low, close and volume values for a number of stocks at time intervals of 30 minutes so 13 data points for each full trading day (9:30 - 4:00pm). This time interval was selected as to balance the improved prediction resolution and increase in data points for training from using shorter time intervals against the amount of Stocktwits and Twitter posts available for each time step. It is also consistent with the findings of previous work (Renault 2017) as discussed in section 4. The data returned from the yfinance library comes in the form of a pandas dataframe, which is then split up and saved to separate csv files for each day and stock.

Unfortunately, the cost of using intraday data, irrespective of the interval used, is that data going back more than 60 days is unavailable without paying a substantial amount of money. As such the intraday stock data I have collected currently covers a date range from 2020/09/25 (the earliest date available from when data collection was setup) to 2020/04/05. From this data some technical indicators were also calculated and added to the dataset on the finance side, these were the Simple Moving Average and Bollinger Bands across a 2 day interval therefore later models that used these indicators made predictions on data starting from 2020/10/02.

5.1.2 Social Media Data

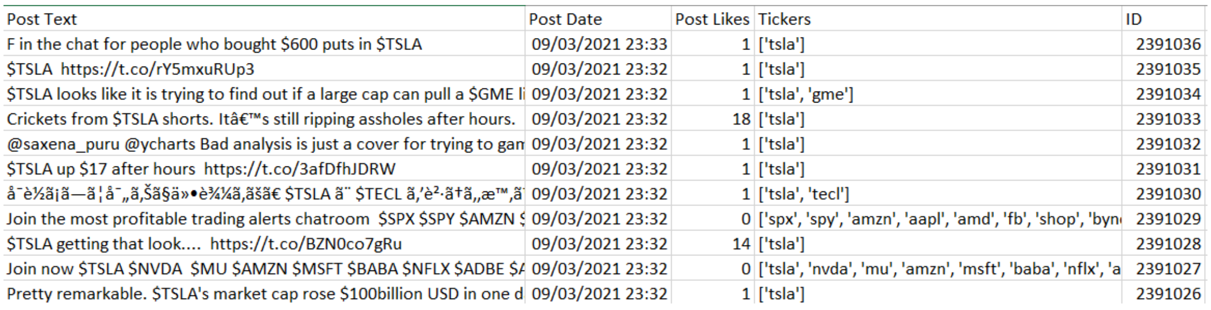
The social media data used was from two sources, the first of which is Stocktwits posts collected via their rate limited API. As a result, similarly to the intraday stock data, the collection of this was an ongoing processes throughout the project in order to both collect the most recent data and all the past data required. Initially this was setup to run 24/7 on a raspberry-pi alongside my own computer to maximise the rate of data collection. Later, once the bulk of posts had been downloaded just running the required code on my computer was sufficient to keep up with collection on new posts. Other than the text and date of each post some posts are also tagged by the user posting as either having a bullish (positive) or bearish (negative) sentiment, examples of which are show in figure 12. Unfortunately the number of likes each post has isn't accessible via the API, however some measure of post popularity can be extracted from the total number of posts and total number of likes the user making the post has, therefore this is also stored for each post. Posts are stored grouped by the stock they discuss and the day they were posted on for ease of use later.



Post Text	Date	Sentiment	ID	Num Posts	User Likes
$TSLA Lucid will be Tesla killer.	12/03/2021 00:06	Bearish	302882499	83	1012
$TSLA stimulus coming money going in tsla going to 1000+	12/03/2021 00:06	Bullish	302882465	1909	2260
$TSLA Forget the financials, number of cars sold and the competition, just	12/03/2021 00:05	N/A	302882037	48	1143
Bears can&#39;t get enough of being rekt by Elon!	12/03/2021 00:04	N/A	302881940	1445	5043
$TSLA $QQQ $NFLX $FB $SNAP Who bought the dip? https://youtu.be/_5D	12/03/2021 00:04	Bullish	302881916	296	4184
$TSLA this will never see 500$ again lol	12/03/2021 00:02	Bullish	302881019	271	1059
$TSLA oh no its down -0.20% please shorts stop destroying us ðŸ˜‚ðŸ˜‚ðŸ˜	12/03/2021 00:02	Bullish	302880866	271	1059
$TSLA massive pyramid	12/03/2021 00:01	Bearish	302880496	381	323
$TSLA $200 point drop next week	12/03/2021 00:01	Bearish	302880340	387	323

Figure 12: Example of TSLA Stocktwit posts showing what information is stored

The other source of social media data for this project is from Twitter and was collected by using the python library Twint to search for posts containing the ticker names i.e. $TSLA or $AMD of a select group of stocks. This process specifically collects posts that are discussing the stock rather than general discussion related to the company. While this data might also be useful it is outside the scope of this project as it would require significantly more work to filter out irrelevant information. Similarly to the Stocktwits data, the post text and data is stored alongside a measure of each posts popularity which in Twitter's case is the actual number of likes on the post. Finally the tickers mentioned in the tweet are also stored and are later used to filter out tweets where more than two tickers are mentioned, as past that point it was decided that any information gleaned from such posts would be impossible to attribute to a single stock easily. Also from looking at the data it was clear that past that point a lot of posts were just mentioning tickers in order advertise something irrelevant to this project, examples of Tweets and what information is stored are shown in figure 13. Initially we did attempt to collect this information directly through the Twitter API, however they restrict the searching of cashtags e.g. $TSLA behind a paywall hence the use of Twint for this.



Post Text	Post Date	Post Likes	Tickers	ID
F in the chat for people who bought $600 puts in $TSLA	09/03/2021 23:33	1	['tsla']	2391036
$TSLA https://t.co/rY5mxuRUp3	09/03/2021 23:32	1	['tsla']	2391035
$TSLA looks like it is trying to find out if a large cap can pull a $GME li	09/03/2021 23:32	1	['tsla', 'gme']	2391034
Crickets from $TSLA shorts. Itâ€™s still ripping assholes after hours.	09/03/2021 23:32	18	['tsla']	2391033
@saxena_puru @ycharts Bad analysis is just a cover for trying to gan	09/03/2021 23:32	1	['tsla']	2391032
$TSLA up $17 after hours https://t.co/3afDfhJDRW	09/03/2021 23:32	1	['tsla']	2391031
å¯è½ã¡ã—ã¦å¯„ã,Šã§ä»•è¾¼ã,ãšã€ $TSLA ã¨ $TECL ã,'è²·ã†ã„æ™,ã	09/03/2021 23:32	1	['tsla', 'tecl']	2391030
Join the most profitable trading alerts chatroom $SPX $SPY $AMZN $	09/03/2021 23:32	0	['spx', 'spy', 'amzn', 'aapl', 'amd', 'fb', 'shop', 'byn	2391029
$TSLA getting that look.... https://t.co/BZN0co7gRu	09/03/2021 23:32	14	['tsla']	2391028
Join now $TSLA $NVDA $MU $AMZN $MSFT $BABA $NFLX $ADBE $A	09/03/2021 23:32	0	['tsla', 'nvda', 'mu', 'amzn', 'msft', 'baba', 'nflx', 'a	2391027
Pretty remarkable. $TSLA's market cap rose $100billion USD in one d	09/03/2021 23:32	1	['tsla']	2391026

Figure 13: Example of TSLA Twitter posts showing what information is stored

5.1.3 Social Media Data Further Processing

For both sources of data, links to websites were replaced with the token [LINK]. This was necessary since the actual text of the links carry very little information and cause issues when tokenised since they won't be in the vocabulary of the model. However the presence of the link is still useful to the context of the sentence hence the replacement with [LINK] token. Similarly tickers e.g. $TSLA where replaced with [TICKER] to allow the sentiment analysis model to generalize better between different stocks. Specifically for Twitter posts, tokens that took the form @username were removed since again they didn't contain any useful information and nearly always occurred at the start of a retweet so weren't useful to the structure of the post either. #'s were also removed but not the text following them. Finally posts that, after this pre-processing, posts which contained 5 characters or less (without the [LINK] and [TICKER] tokens) were discarded since they didn't contain enough information to be able to make an accurate prediction of it's sentiment. Punctuation and stops words etc were left in the text since RoBERTa still uses this information to understand the context of the sentence.

5.1.4 Data Normalisation

All data used by the stock prediction neural network model need to be normalised such that the scales of all inputs were similar. This is necessary as input features of different scales will result in the model having to learn weights with vastly different scales which is possible but just makes it more difficult to resolve the local minima for error. Also activation functions such as tanh and sigmoid used in LSTMs expect relatively small values with magnitudes ranging between 0-10 so the scale for normalisation is kept relatively small. In this project a MinMaxScaler is used which normalises input features to values between 0 and 1, the equation used to calculate the normalised value is show in equation 3.

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3}$$

Where x_{norm} is the normalised value of input x and x_{min} and x_{max} are the minimum/maximum known values of the input feature.

5.1.5 Stock Selection

This report looks at the following stocks: AAPL (Apple), AMD, AMZN (Amazon), FB (Facebook), MSFT (Microsoft), NVDA (Nvidia) and TSLA (Tesla) with Tesla as the primary focus. These were selected based on their relative popularity in terms of the quantity of posts discussing them on social media with Tesla being foremost among these hence their specific focus. Other stocks such as Intel and Uber were initially considered but later disregarded as their weren't enough posts for each time interval for the data to be of use. Since this project only uses posts made just before and during market hours, this limits the number of useful posts available for day. Figure 14 illustrates why we chose to focus on TSLA since there were on average around 100 posts made in each 30 minute time interval, as compared to around 10 for AMD. The benefit being that the greater the number of posts available the less it is effected by either noise in the underlying data or by posts being incorrectly classified by the sentiment analysis algorithm. AMD was on the tail end of what was really useful since stocks with fewer posts quite commonly had time intervals with no posts made however due to time constraints on the project only TSLA was really used for testing so this didn't become an issue.

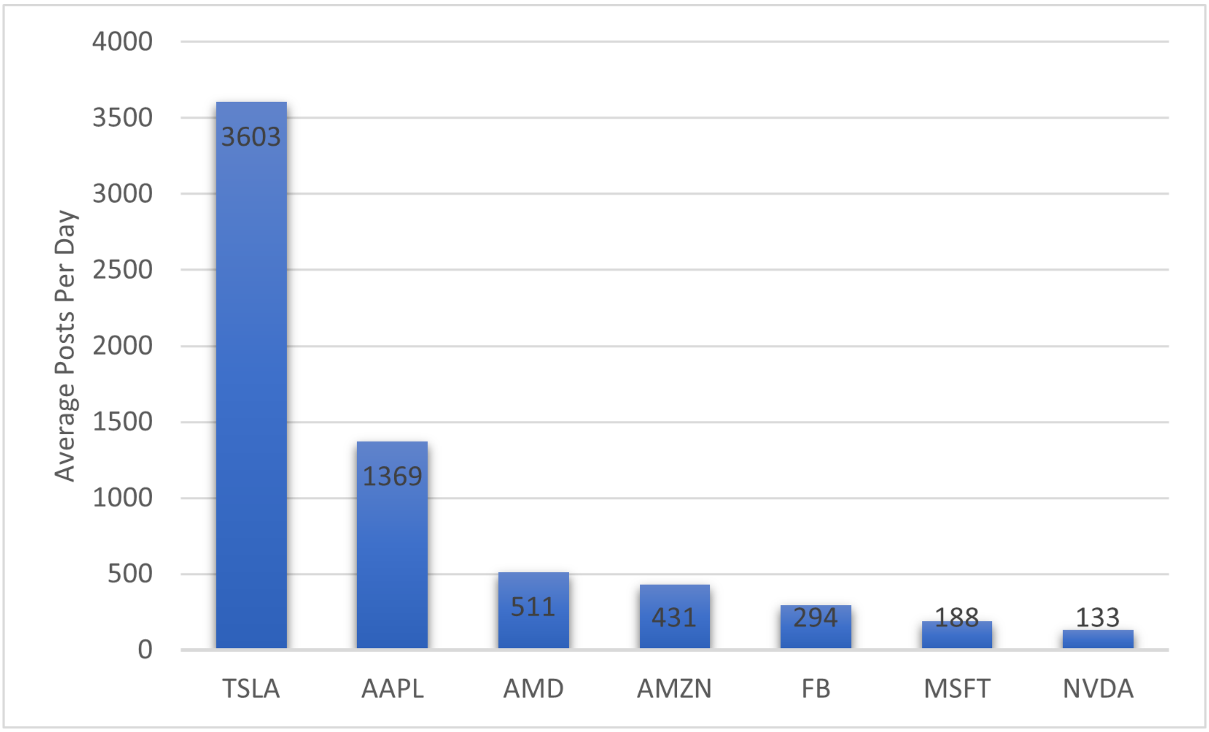


Figure 14: Chart showing average number of Stocktwit posts per day for each stock over a period of 30 days from 2021/03/07 to 2021/04/05

5.2 Sentiment Analysis

Initial experimentation with sentiment analysis was done using VADER on a tagged twitter dataset based on the success that such methods had shown in previous work. Initial results on the twitter dataset performed significantly worse than expected so some experimentation was undertaken using different lexicons such as NTUSD-Fin (C.-C. Chen, Huang, and H.-H. Chen 2018) which is based on 330k financial social media posts, however, even with this the VADER model still didn't perform as well as hoped.

5.2.1 RoBERTa - Model Generation

RoBERTa was the other model tested for sentiment analysis with the base pre-trained model, RoBERTaForSequenceClassification, taken from the huggingfaces transformer python library and built using Pytorch. The purpose of using a pre-trained model being to avoid this costly step of training on the over 150GB of data that RoBERTa uses (Liu et al. 2019) which we simply don't have the compute power and or time to do.

In order to fine tune the model a training dataset had to be generated from the tagged Stocktwits posts, initial experiments with this used 250,000 TSLA posts to train a model specifically to tag TSLA posts. The reason for this being that by training a model to tag posts for a single stock on just posts from that stock it's more likely to be able to correctly capture domain specific context e.g. discussion of batteries for Tesla. Experiments were also conducted with larger dataset of 750,000 mixed posts from the 7 stocks mentioned in section 5.1.5 to investigate if simply using more data outweighed the performance benefit of the previous test. Such a model would also have the benefit of not needing to train a separate model for each stock which is very time consuming and relies upon having significant data available for training/validation.

Once the dataset is generated and saved to a file, it is then split into training, validation and testing sets that are then saved to Pytorch dataset classes, which when a specific post is requested, will tokenise and then return a list of token id's that correspond to words in the model dictionary along with a attention mask. The attention mask dictates which of the tokens the model should consider where the tokens that represent the actual text of the post are marked with a 1 and 0. This is used to indicate the location of padding tokens that have been added to ensure all tokenised posts are of the same length. A max length of 140 tokens was imposed as deemed sufficient enough to encode nearly every post whilst keeping computation time as low as possible.

Finally the model was trained over 5 epochs on the training set with the best epoch selected as the one which performed best on the validation set, this approach was also used to test some different hyper-parameters, however testing of these was not very extensive since running the model for 750k posts over 5 epochs took roughly 8 hours per run on a GTX 1080. The testing was limited to a few variations/combinations of the following hyper-parameters:

- Number of epochs
- Batch size
- Learning Rate

Gradient clipping was also added so that gradients during backpropogation could not exceed a value of 1.0, this was implemented to prevent the issue of exploding gradients. By default these parameters were set as suggest in the RoBERTa paper (Liu et al. 2019).

The best performing model on the validation dataset was then evaluated on the testing set to give the final results and then saved to be used later to classify the untagged Twitter and Stocktwits posts.

5.2.2 RoBERTa - Data Tagging

To classify the untagged Stocktwits and Twitter posts, the best performing model produced by the previous step is loaded and used to tag the data similarly to the model generation phase. However in this case the predictions are now saved to two csv files (one for Stocktwits and another for Twitter posts) where the classifications (-1 = negative, 1 = positive) are summed for every 30 minute time interval of each day. Stored alongside this is the number of posts for each time interval and a weighted sum based on the sentiment and post popularity e.g. likes, these values are then used later to generate features to input into the stock prediction neural network model. The original tagged Stocktwits posts used for training are also stored in a similar way, although in a separate file, as these are still equally if not more useful than the data tagged by this model since the sentiment of this data is 100% accurate. This process is summarised by the diagram in figure 15.

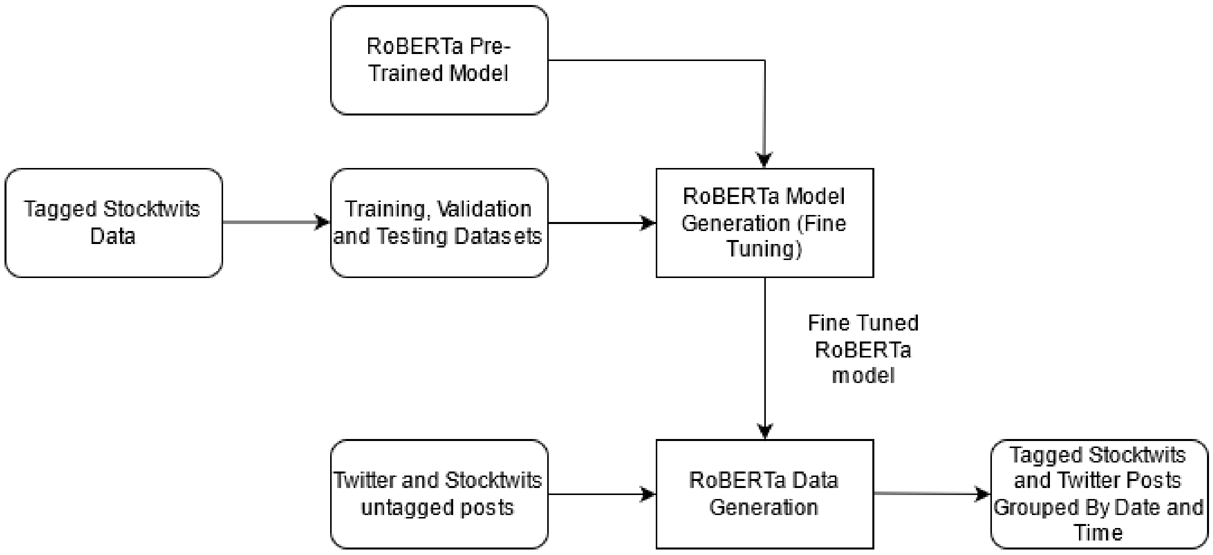

RoBERTa Pre-Trained Model
Tagged Stocktwits Data
Training, Validation and Testing Datasets
RoBERTa Model Generation (Fine Tuning)
Fine Tuned RoBERTa model
Twitter and Stocktwits untagged posts
RoBERTa Data Generation
Tagged Stocktwits and Twitter Posts Grouped By Date and Time

Figure 15: Diagram illustrating sentiment analysis model

5.3 Stock Prediction - Neural Network Models

As with the sentiment analysis model, the various neural network models tested for stock prediction were constructed using Pytorch. The generalised view of the model is as follows: first a matrix of tensors is constructed from the past day (13 time intervals) of input features which are discussed in section 5.4, for example an input of 21 features would result in a 13 x 21 matrix. The matrix is then passed into the neural network model which consists of one or more layers of LSTMs and or CNNs, the outputs of which are then run through one or more dense/linear layers in order to give the final prediction. This was framed as a regression task where the prediction represents some form of future stock price e.g. closing price one day in the future, further predictions are made by sliding the window of data used as input features shown in figure 16 one time step along. The choice of making this a regression problem was due to the fact that doing so provides a much better visualisation of the results of the stock prediction task which in turn makes analysis of the model much better. It also has the potential to give much more useful information as compared to a classification problem where, for example, buy/sell signals are generated since the results are more detailed.

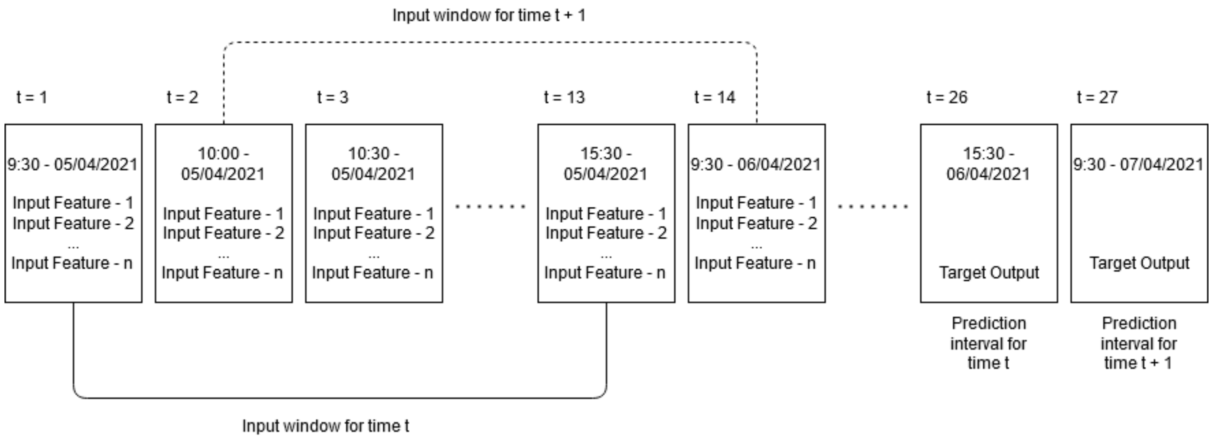


Figure 16: Input Windows visualised for an input window size of 13 (1 day) and a prediction offset of 13 (1 day)

While the input/output features and structure of the neural network varies across the different experiments and as such introduce their own model specific hyperparamters, there are some that are common to all the models and are listed below:

- Batch size
- Number of epochs
- Learning rate
- Input window size

Different prediction offsets were also tested and although this would be an interesting area of further research/testing, an offset of 1 day was chosen so as to balance the usefulness of the actual prediction vs how difficult it is for a model to predict. Similarly different optimiser functions such as RMSprop, SGD and Adam were also tested, however they all performed pretty much the same in terms of the RMSE loss achieved. Hence, we choose to use the Adam optimiser going forward since it was used in previous works such as (Mehtab and Sen 2020), also the default parameters for the Adam optimiser require little tuning which is ideal for this project due to the number of other hyperparameters that need to be tested.

Hyper-parameter optimisation is performed for each experiment so as to ensure that accurate comparisons of performance can be made, however the rigorousness to which this was done varied depending on the initial observations made on the performance after changes were made. This was the case as this process can be quite time consuming especially as the models got more complex, hence, if a change had clearly made no significant difference to performance, fewer parameter combinations would be tested so as to avoid wasting too much time. The following subsections detail the different experiments performed.

5.3.1 LSTM - Closing price

Initial experimentation focused on the use of LSTMs and BLSTMs with a limited set of input features i.e. simple features for a stock such as open and close prices along with weighted sums of sentiment from both Twitter and Stocktwits and the respective numbers of posts per time interval for each. Time of day and day of the week were also included as input features since trading behaviour differs over a trading week e.g. higher trading volume just after the market opens particularly at the beginning of the week. The target output for this experiment was the closing price 1 day ahead.

After some early testing, dropout i.e. adding a random probability of setting one output of an LSTM layer to zero was implemented so as to add some random noise to the training process alongside using weight decay. These help prevent over fitting the model to the training data meaning it can generalise better for unseen data. Gradient clipping was also added to help prevent the issue of exploding gradients common to recurrent neural networks.

The overall aim of this first experiment was to investigate the effectiveness of LSTMs and compare it to the bi-directional form, also it was a good way to get an idea of what sort of hyper-parameter values are suitable before making the model too complex which would in turn increase the difficulty in finding the optimal values. The hyper-parameters specifically tested for this experiment were:

- Hidden layer size of LSTM
- Number of layers within the LSTM
- Whether to use bi-directional LSTMs
- Dropout and weight decay of LSTM

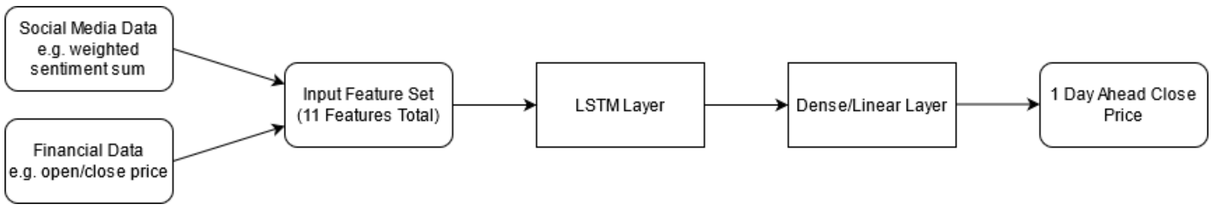


Figure 17: Diagram illustrating LSTM - Closing Price model

5.3.2 LSTM - Split Sentiment

This next experiment attempted to improve the LSTM performance of the previous iteration by separating the features derived from social media posts and using them as inputs to a separate LSTM layer before combining it with another LSTM layer using features based on financial data. The idea behind this came from (Prakash and Tayyar Madabushi 2020) which achieves state of the art performance for a sentence classification task by combining the hidden layers of two models which each have their own feature set, while the particular use case differs as compared to this project the general idea behind it may still be of use. Several features based on the existing social media sentiment data were also added such as an average weighted sentiment sum and raw sentiment sum i.e. just sum of sentiment classification (1 or -1) ignoring post popularity. This was done to make better use of the existing data that was available and because prior to this, the sentiment side of the model only made use of 4 features.

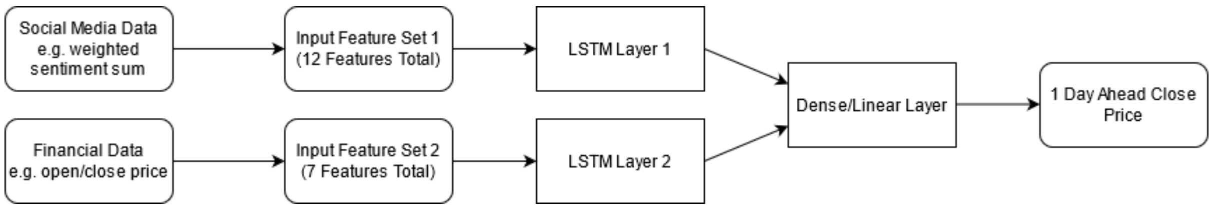


Figure 18: Diagram illustrating LSTM - Split Sentiment model

The aim of this experiment was to simply compare the split sentiment model to the original LSTM model in section 5.3.1, however the results weren't conclusive due to the relatively high variation in results from repeated runs, thus it was decided to include this as a hyper-parameter to optimise for later experiments, therefore hyperparameters added at this stage were:

- Whether to use the split sentiment model
- Input and output dimensions of linear/dense layers

5.3.3 LSTM - Closing Price Percentage Change

Based on results from previous experiments it seemed that the models struggled with the change in scale of stocks' closing prices as many of the stocks this project focuses on have seen sharp increases in value over the course of the pandemic. Tesla's share price in particular has risen upwards of 400% since April 2020 which coupled with the fact that most of these sharp rises occurred within the latter part of the dataset at the time i.e. the testing and validation dataset, compounded the difficultly of this already challenging task. As such it was decided to change the model output to the percentage change of the closing price to simplify what the model needs to do since it would no longer need to also learn the scale changes in the data on top of everything else. Closing price percentage change was also added as an input feature.

5.3.4 LSTM - Simple Moving Average Percentage Change

In this experiment technical indicators e.g. simple moving average were added as mentioned in section 5.1.1. This decision was based on previous research such as (Sezer, Ozbayoglu, and Dogdu 2017) which found that adding indicators like Simple Moving Average (SMA) can improve the performance of their neural network. Again, based on this and the results of the previous experiment, the model output was again changed this time to predicting the percentage change of the simple moving average. The reason behind this decision being that switching to predicting percentage change amplified the already present volatility in the data, however by predicting the SMA some of the volatility is smoothed out. This does have the detrimental impact of the fact that the model output is no longer a direct representation of the future closing price. To counteract this, the window size of the data used to calculate the SMA was kept to a minimum. A window size of 2 days was used so as to provide a balance between the prediction accuracy and the usefulness of the prediction, although other window sizes were also tested.

5.3.5 LSTM + CNN - Ensemble Model

With the results from the previous experiment being a lot more promising it was decided to try and incorporate CNNs into the model. With the choice of either adding a CNN sequentially, so feeding the CNN output into an LSTM, or adding a CNN layer in parallel, the latter of these was chosen since (Mehtab and Sen 2020) had found that an approach of the former of these options actually performed worse than a CNN layer on it's own. Thus the second approach was implemented and was done so in a similar fashion to the split sentiment model i.e. apply the models to the data separately then combine the results using a dense/linear layer.

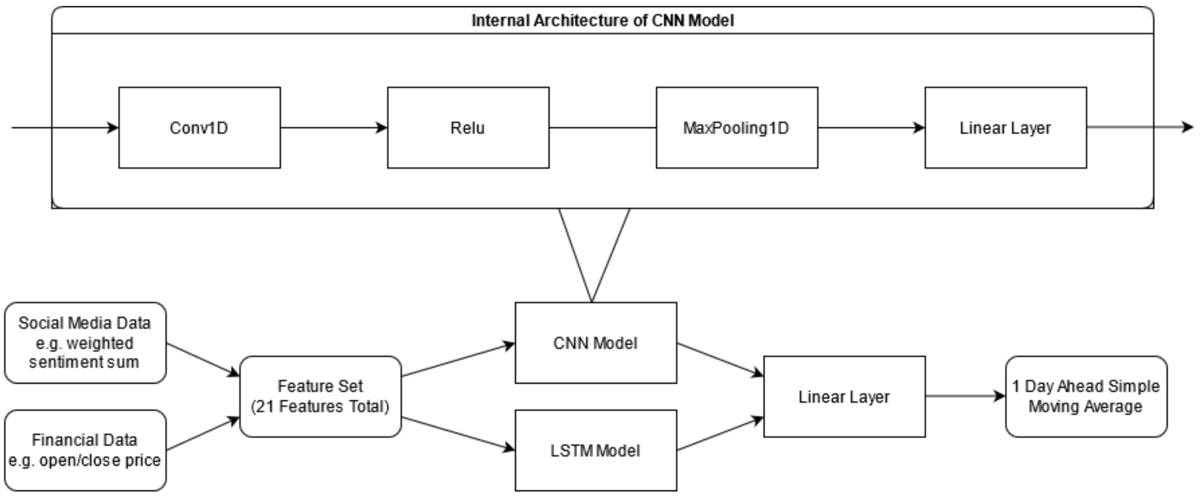

Internal Architecture of CNN Model
Conv1D
Relu
MaxPooling1D
Linear Layer
Social Media Data e.g. weighted sentiment sum
Financial Data e.g. open/close price
Feature Set (21 Features Total)
CNN Model
LSTM Model
Linear Layer
1 Day Ahead Simple Moving Average

Figure 19: Diagram illustrating LSTM + CNN - Ensemble model where the cutout section is the architecture of the CNN model used in the ensemble and the LSTM model is equivalent to that used in the previous experiment.

This model architecture added a number of hyperparameter relating to the CNN which were:

- Whether to include the CNN model
- Kernel size i.e. size of the filter used for the convolution and max pooling steps, for example a 1x3 filter.
- The stride for applying these filter i.e. after performing the convolution for one data point, how many data points to skip before performing the next convolution.
- Number of output channels for the convolution i.e. how many different filters to use.

Due to the larger number of hyperparameters in this model particularly with the inclusion of all the CNN related parameters a decision was made to test these separately as there wasn't enough time to perform extensive hyperparameter optimisation across every parameter at once. As such, another model which just featured the CNN layer detailed in figure 19 was developed and used to test specifically the CNN related hyperparameters. The results of this were then used to guide a much smaller subset of features to be tested on the full model.

5.4 Model Feature Sets

The following lists the set of features used by the stock prediction model and from which point in the experiments they were included:

- Percentage change of whatever metric the target is e.g. 2 day simple moving average close price - added in LSTM percentage change model
- Open price
- Close Price
- High price for interval
- Low price for interval
- Trading volumes
- Day of the week
- Time of day
- 2 day window simple moving average price - added in LSTM simple moving average % change model
- 2 day window upper bollinger band - added in LSTM simple moving average % change model
- 2 day window lower bollinger band - added in LSTM simple moving average % change model
- Total Stocktwits sentiment score weighted by post popularity
- Number of Stocktwit posts
- Total unweighted Stocktwits sentiment score - Added in split sentiment model
- Average Stocktwits sentiment score weighted by post popularity - Added in split sentiment model
- Average unweighted Stocktwits sentiment score - Added in split sentiment model
- Total Twitter sentiment score weighted by post popularity
- Number of Twitter posts
- Total unweighted Twitter sentiment score - Added in split sentiment model
- Average Twitter sentiment score weighted by post popularity - Added in split sentiment model
- Average unweighted Twitter sentiment score - Added in split sentiment model

6 Results

6.1 Evaluation and Testing Procedure

6.1.1 Sentiment analysis

On the sentiment analysis side of the project, model performance was evaluated based on the accuracy and F1 measure of the classification results. The testing process involved splitting the data into training, validation and testing datasets with a ratio of 80:10:10, hyperparameter tuning was done by comparing each model using the validation dataset before with the best performing model being evaluated on the testing dataset. This ensures that the final results give a realistic measure of performance on unseen data. Later the split ratio was changed to a 80:20 split between training and validation data followed by a fixed sized testing set of 50,000 posts. This change was motivated by an increase in the data available allowing more to be used for the validation and testing datasets. This meant that a more accurate comparison and final evaluation of the models tested could be achieved. The fixed size testing dataset was created so that the models trained on the larger mixed set of posts could be compared to the smaller stock specific models, for the same reason the test set contained the same posts for each run unlike the training and validation sets which were randomised.

6.1.2 Stock Prediction Model

The results of the stock prediction model were evaluated using root mean square error (RMSE) the calculation of which is shown in equation 4, the use of this measure is primarily to compare models against each other rather than give some real world indicator of performance.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}} \tag{4}$$

It’s important to note however, that direct comparisons between the performance of models that produce different outputs e.g. closing price vs percentage change of SMA cannot be made since the change in output has a significant impact on the values of error produced. That being said the model architectures used in the earlier experiments were re-examined for the final output tested i.e. percentage change of the simple moving average to allow some comparison between experiments. A simple trading strategy was also developed to simulate potential profits/losses made based on trades using the predictions produced by the models. The purpose of this was to present a more practical evaluation of performance as compared to the RMSE error. With the change to predicting the SMA it was no longer as applicable and so was excluded from the results, however it is still calculated and displayed within the program for curiosities sake. It's also important to reiterate here that the aim of this project is not to make automated trades but rather help the user make more informed decisions about their trades, thus the RMSE error makes a far better measure of performance and profit/loss would have been more of an interesting side note.

For evaluation/testing, 4 fold cross validation was employed to avoid over-fitting the models to a single pattern of data while also ensuring there is enough data to train the model effectively which might not be the case if using a larger validation set. By validating the results across different sections of the training data we are able to produce a model that is better able to generalise to new data as the validation loss produced is the average loss from testing across different sections of the input data. A measure of the models performance on unseen data is given by the results on the testing dataset.

For final testing of which all the stock prediction results are based on, the date range of the data used was from 2020-09-25 to 2021-04-19, this was fixed for all the tests so that fair comparisons between the different experiments could be made.

6.2 Sentiment Analysis

Two sources of information were used to evaluate the various sentiment analysis models, the first of this as mentioned previously is the tagged Stocktwits posts for specific stocks. The second is a dataset of roughly 6000 hand tagged Twitter posts discussing stocks available here: https://www.kaggle.com/yash612/stockmarket-sentiment-dataset. The Stocktwits dataset is by far the most useful of the two as there is far more data available (In excess of 1 million posts downloaded so far) and since the posts are tagged by the user posting them, the tagged sentiments are likely to be more accurate than those of the twitter posts which are hand tagged by the creator of the dataset. That being said the formatting and contents of Twitter posts are likely to differ from their Stocktwits counterparts so the second dataset does at least still give some indication of how performance of the models will vary when applied to the different data sources used in the project.

Figure 20: Table comparing the performance of models on the Twitter dataset

Model	Class 0 Precision	Class 0 Recall	Class 1 Precision	Class 1 Recall	Macro Avg F1-Score	Accuracy
VADER Base Lexicon	0.57	0.35	0.7	0.85	0.6	0.67
VADER Custom Lexicon	0.67	0.51	0.75	0.86	0.69	0.73
RoBERTa Mixed Model	0.8	0.66	0.82	0.91	0.86	0.82

When evaluating on the Stocktwits posts TSLA and AMD were selected to be tested on since they both had ample data available, the test set size was 50,000 posts for both stocks. The class imbalance present in the data was left as it was so as to best represent unseen data.

Figure 21: Tables showing class balance for Stocktwits testing and training datasets

(a) Testing

Stock	Positive (Class 1)	Negative (Class 0)
TSLA	35631	14369
AMD	46629	3371

(b) Training

Stock	Positive (Class 1)	Negative (Class 0)
TSLA	169440	80560
AMD	212452	37548

Figure 22: Table comparing the performance of models on TSLA Stocktwits dataset

Model	Class 0 Precision	Class 0 Recall	Class 1 Precision	Class 1 Recall	Macro Avg F1-Score	Accuracy
VADER Custom Lexicon	0.57	0.39	0.78	0.88	0.65	0.74
RoBERTa TSLA Model	0.77	0.70	0.88	0.91	0.82	0.85
RoBERTa Mixed Model	0.79	0.72	0.89	0.92	0.83	0.87

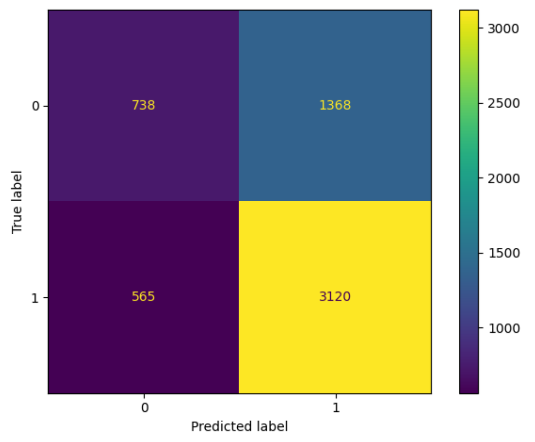
Figure 23: Table comparing the performance of models on AMD Stocktwits dataset

Model	Class 0 Precision	Class 0 Recall	Class 0 Precision	Class 0 Recall	Macro Avg F1-Score	Accuracy
VADER Custom Lexicon	0.27	0.37	0.95	0.93	0.63	0.89
RoBERTa AMD Model	0.51	0.55	0.97	0.96	0.75	0.93
RoBERTa Mixed Model	0.42	0.65	0.97	0.93	0.73	0.92

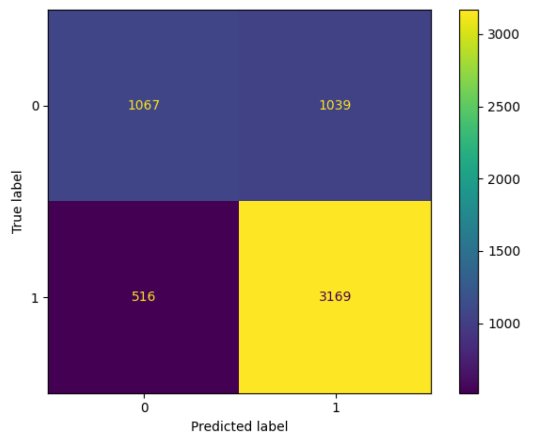
6.2.1 VADER

As is evident from figure 20 and figure 24a vs 24b, the use of a lexicon customised to the use case significantly improves the algorithms performance. While the base lexicon was built to predict social media sentiment it doesn't hold any information about the financial context specific to this project and so it makes sense that the addition of such information to the algorithm would positively impact performance. However the impact of this was not as significant as originally hoped with the F1 score below 0.7 for each of tests run, much of the error being from it's inability to correctly predict the class 0 or negative posts. While some more performance could have perhaps been gleaned from experimenting with different thresholds or fine tuning the lexicon, the impact of this would most likely be fairly negligible hence the focus on the next model, RoBERTa.

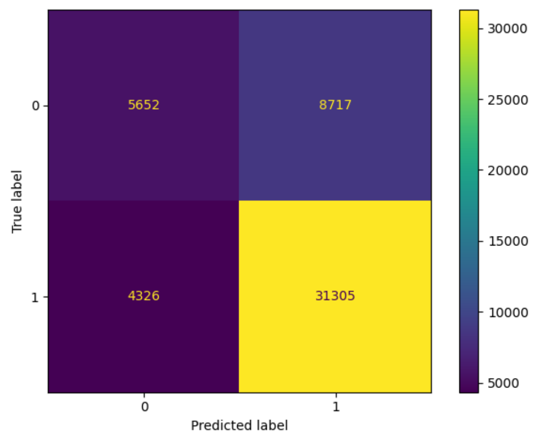
(a) Twitter - VADER - Base Lexicon



(b) Twitter - VADER - Custom Lexicon



(c) Stocktwits - TSLA - VADER



(d) Stocktwits - AMD - VADER

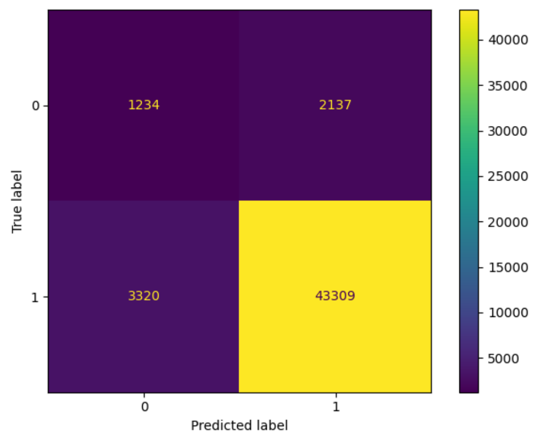
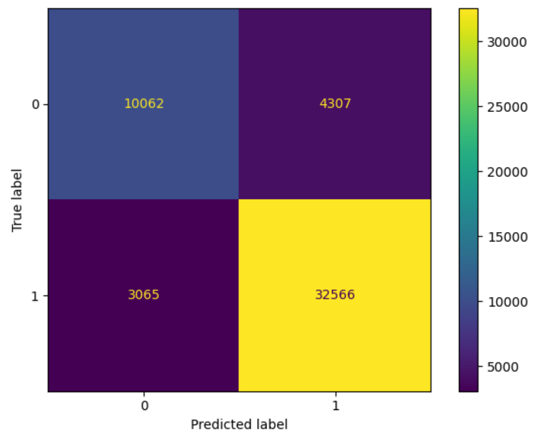


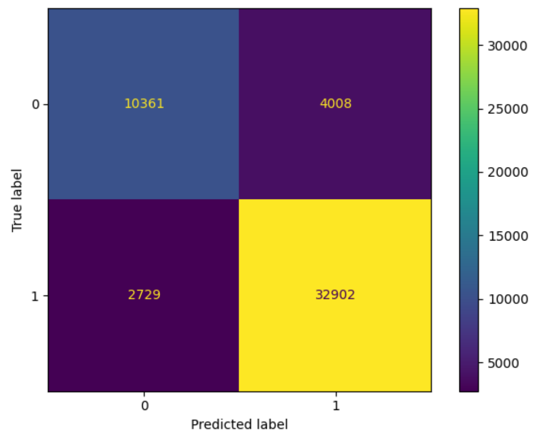
Figure 24: Confusion matrices produced by testing VADER on the Twitter and Stocktwits datasets (0 = negative post, 1 = positive post)

6.2.2 RoBERTa

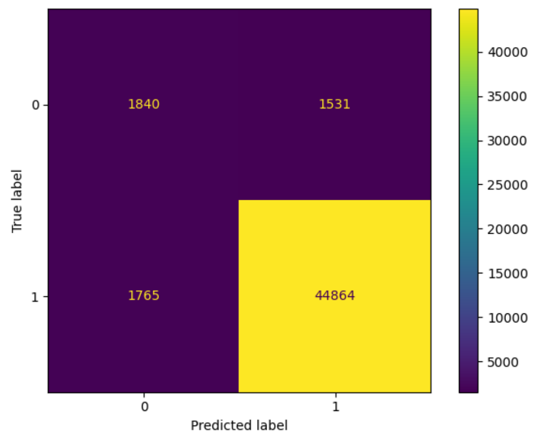
(a) Stocktwits - TSLA - RoBERTa TSLA Model



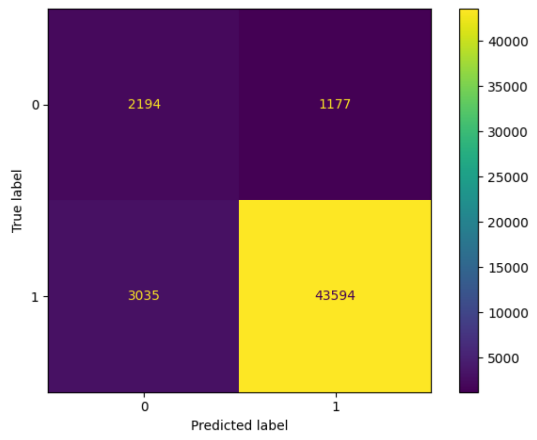
(b) Stocktwits - TSLA - RoBERTa Mixed Model



(c) Stocktwits - AMD - RoBERTa AMD Model



(d) Stocktwits - AMD - RoBERTa Mixed Model



(e) Twitter - RoBERTa Mixed Model

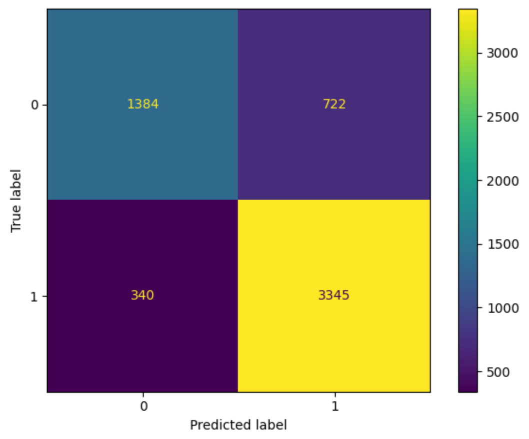


Figure 25: Confusion matrices produced by testing RoBERTa models trained on posts from either a specific stock or a mixed set, on the Twitter and Stocktwits datasets

The RoBERTa models significantly outperformed the approaches using VADER by achieving accuracies and F1 scores in the mid-low 80s with the exception of the Stocktwits AMD dataset where the high F1 score achieved is closer to mid seventies. While the accuracy for this model is actually quite high due to the level of class imbalance present in the AMD data, highlighted in figure 21a, it is not a good performance metric in this particular case for the same reason as a high accuracy could be achieved by just predicting everything as class 1. On the other hand, the macro average F1 score is actually very useful as it weights the F1 scores of both classes equally regardless of the support for each class. This is exactly what we want as the fewer negative posts there are for a stock the more important it is to get the predictions for those correct since each individual post could provide a much stronger indication of stock price movement.

Similarly to the VADER models, negative posts were the most difficult to correctly classify as shown by the low precision and recall values for that particular class which is reflected by the numbers of false positives/negatives shown in the confusion matrices. The AMD experiments were a particularly bad case of this, this could be due to comparatively few negative examples in the training dataset indicated in figure 21b. A potential solution to this could be to artificially balance the training dataset so that more negative posts can be trained on. However, this doesn't explain why VADER also performed so poorly for the same data as the class imbalance in the training dataset wouldn't have effected it since the model is not trained on the data. The implication of this being that these negative posts are simply harder to classify, this could be because some of these posts are instead being positive about competitors which neither algorithm would be able to distinguish between. It would be interesting to see in future if actually leaving the base tickers in the text e.g. $TSLA actually improves the model performance since it might be possible for the model to learn that positive text surrounding $NIO in a post about Tesla for example, indicates that the post is instead bearish about TSLA. This trend of poor performance on the negative class continued for the Twitter dataset tested, which resulted in a considerable number of false positives being predicted. Despite this the performance on the Twitter dataset was remarkably good considering the model wasn't trained on it. This suggests that the model should work quite well on the unseen Twitter data, however, since we don't have a testing dataset that directly represents actual Twitter data, we don't have solid proof of this.

Figure 26: Table displaying the performance of RoBERTa Mixed model on AAPL and AMZN

Model	Class 0 Precision	Class 0 Recall	Class 0 Precision	Class 0 Recall	Macro Avg F1-Score	Accuracy
AAPL	0.68	0.71	0.93	0.92	0.81	0.88
AMZN	0.65	0.64	0.92	0.93	0.79	0.88

The RoBERTa model trained on the mixed dataset actually performed very similarly to the models trained on specific stocks with the results in figure 26 (AAPL + AMZN) showing consistent performance across two more stocks. This implies that the model is able to generalise quite well and therefore shows some promise for applying to stocks that perhaps don't have as much data to train on. Despite this the stock specific models were applied to classify the posts used to generate the final feature sets for the stock prediction model of the AMD and TSLA stocks. This decision was based on the fact that AMD model in particular saw a greater relative increase in performance and would be faster to retrain once more data becomes available.

This testing has shown that the RoBERTa models are capable of making sufficiently accurate predictions such that these predictions could be used as features for the stock prediction model. Whether or not his data actually aids the stock prediction model remains to be seen and will be investigate among other parameters in the following subsections, it may also be the case that this works for the TSLA stock but not AMD for example due to the comparatively poor performance of sentiment analysis on posts from that particular stock.

6.3 Stock Prediction - Outputs

This section compares the results of the stock prediction model when predicting the different outputs discussed in the methodology, for all the outputs a 1 day ahead prediction of the value is made. The results used to compare the different outputs were generated using the best model produced by hyperparamter optimisation for each of the outputs at the time. The different model architectures e.g. Split Sentiment and LSTM + CNN models were included in these hyperparameters to ensure that the issues that lead to the decisions to change the model outputs earlier on in the project, were still present in the results and not simply solved by using a different model architecture that was developed later on.

6.3.1 Close Price Predictions

Figures 27 and 28 shows the results of training and testing a model that directly predicts the close price 13 time intervals in the future. These graphs show that the model architectures tested in this project perform quite poorly when predicting the close price, in particular there are two mains sources of error shown in the graphs. Firstly, as shown most clearly between time intervals 700-1000 in figure 27 and 1600-1700 in figure 28, there are significant portions of the results where the predictions generally match the trend of the true outputs but are offset vertically i.e. in price by upwards of 100$. The second source of error is present in the same sections but better highlighted between 1100-1400 in figure 27 where the predicted output is horizontally offset from the true outputs. This offset is the same as the prediction offset i.e. how far into the future the predictions are being made, meaning that the model is simply predicting the last known data from the input features. Considering this is occurring within the training predictions it shows that there is a degree of underfitting present, however adjustments to hyperparameters such as the number of training epochs just resulted in overfitting to the training data hence the inclusion of gradient clipping and weight decay.

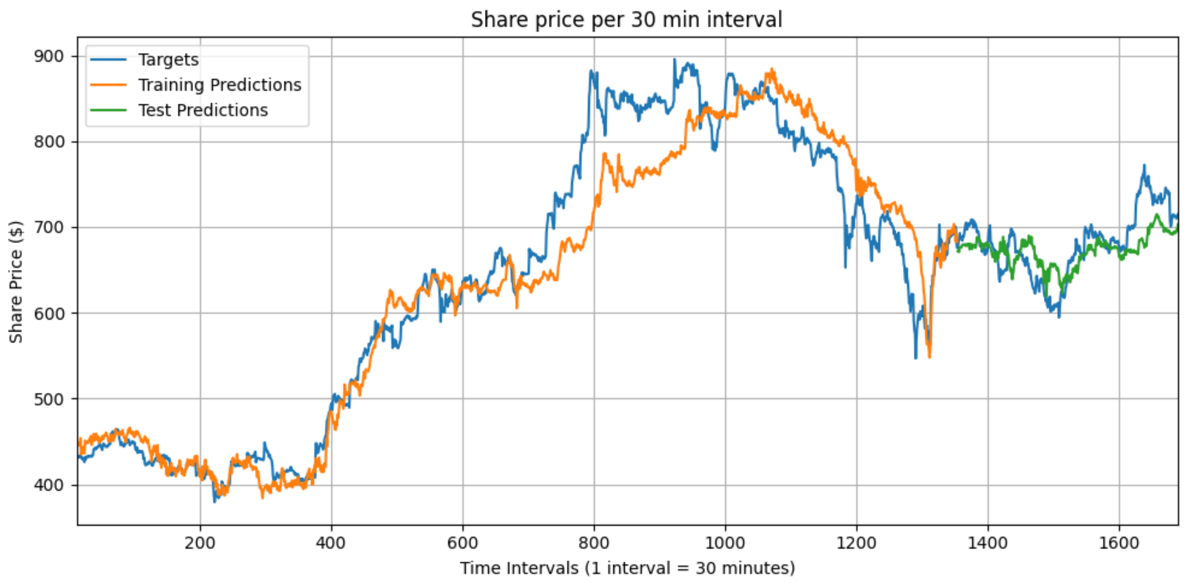


Figure 27: TSLA close price - training and testing predictions

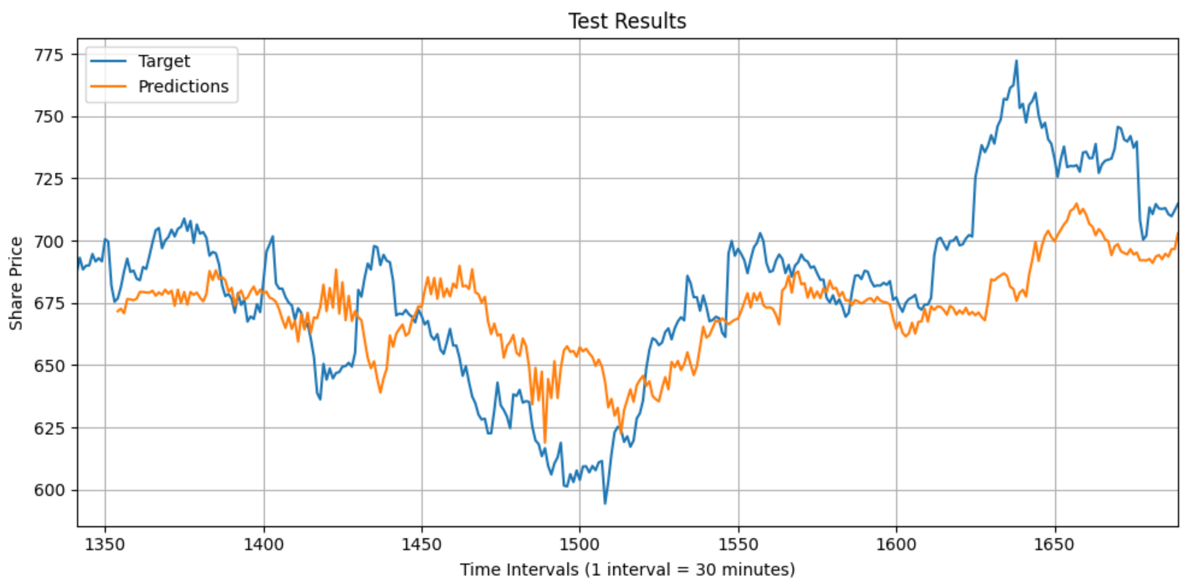


Figure 28: TSLA close price - testing predictions

While it's possible that the predictions could be improved with further hyperparamter optimisation it's unlikely that doing so would produce a model that can consistently generate accurate predictions that can be used to help make trading decisions.

6.3.2 Percentage Change Predictions

Percentage change of the close price was the second prediction output tested, at first glance figures 30 and 31 seem to show significantly improved performance due to the "vertical offset" error being eliminated by the change in output. This is due to the fact that predictions are now based on the change from the most recent data and so getting the prediction significantly wrong is considerably more unlikely. However the "horizontal offset" error where the model predicts close to the latest input data is now much more evident. Figure 31 indicates the cause of this which is that the actual percentages predicted don't fit the data well and vary closely around 0, therefore only predicting the general upwards/downwards trend of the data. As mentioned previously, this is likely due to the high volatility of the underlying stock price which is exacerbated both by using data with a relatively short interval of 30 minutes and by using the percentage change we are effectively taking the gradient of the data which amplifies the already present volatility much like it would noise. This makes it more difficult for the model to fit the data, so much so that none of the models tested in this project were capable of making predictions of this output to a reasonable degree of accuracy that would be useful to an investor.

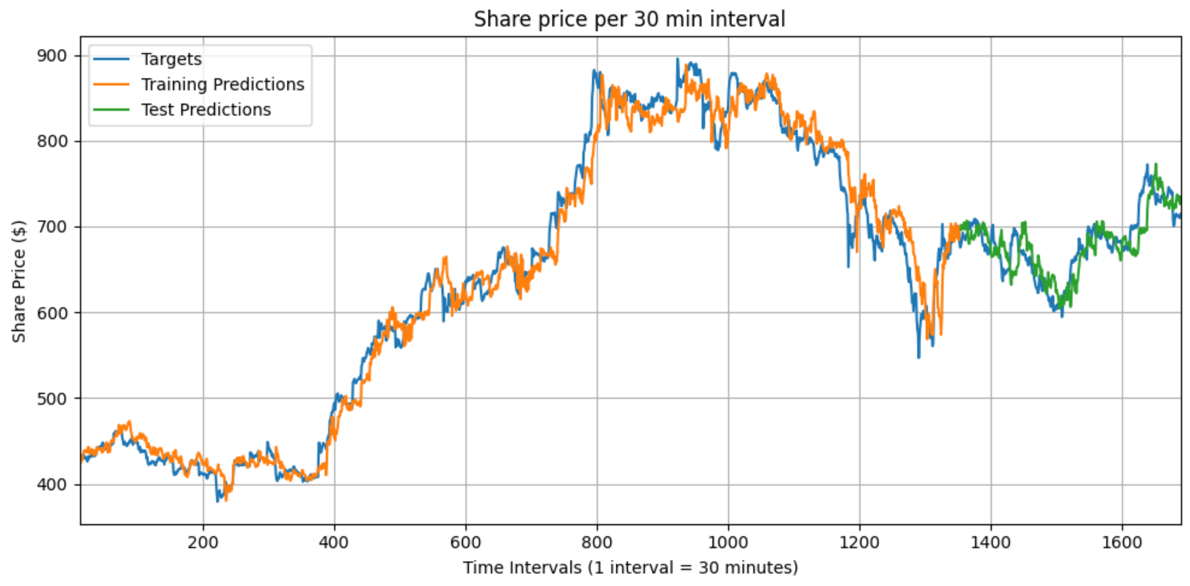


Figure 29: TSLA percentage change converted to close price for visualisation - training and testing predictions

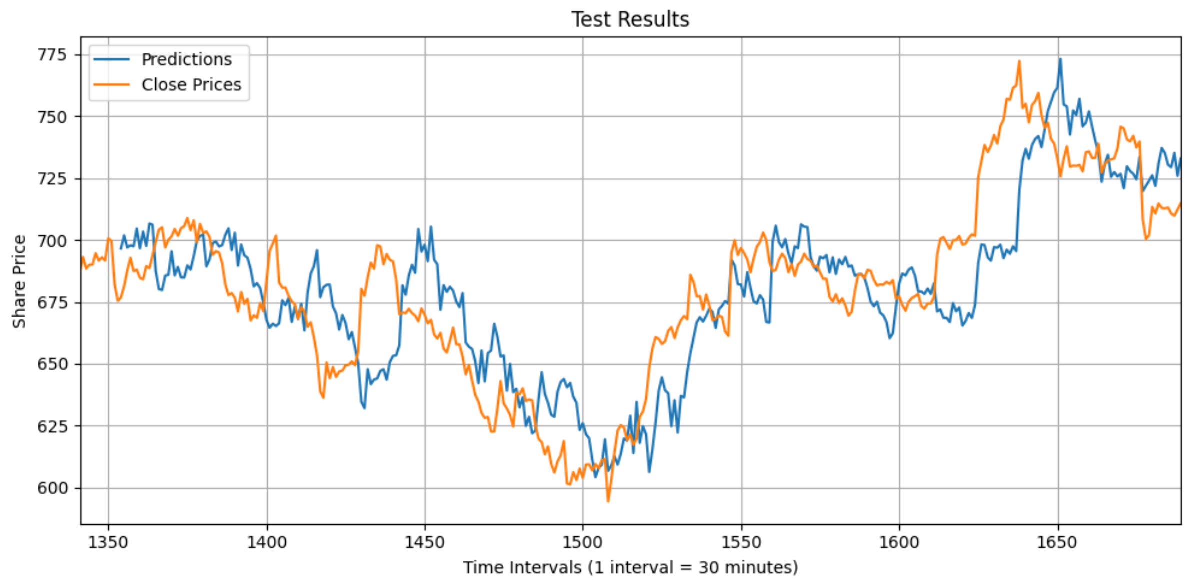


Figure 30: TSLA percentage change converted to close price for visualisation - testing predictions

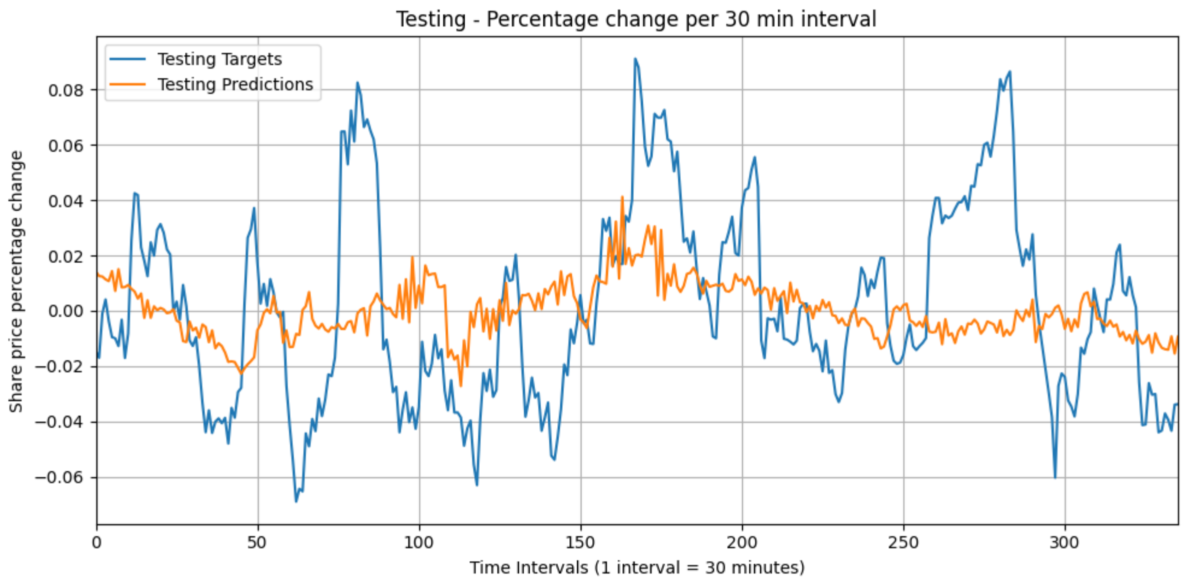


Figure 31: TSLA percentage change - testing predictions

6.3.3 Simple Moving Average Predictions

The percentage change of the simple moving average (2 day window) was the final model output tested, figures 32 and 33 clearly show that the model is much better able to fit the data compared to models predicting the previous outputs. The fit still isn't perfect though as, for example around time interval 1450 in figure 33, a lag in the predictions is visible. Inspection of the corresponding close prices plotted in figure 35 shows a sharp reversal in the close prices just before this point, the SMA prediction lag here indicates that the model was unable to predict this drop. This lag is more obvious when looking at the raw percentages predicted, shown in figure 34, however despite this, the percentage change of SMA was the best output of those tested by far.

A window size of 1 day was also tested for the SMA, however attempting to predict suffered from similar issues to the percentage change of closing price model in that the volatility of the data was quite high and as a result the model was unable to learn the training data well without overfitting.

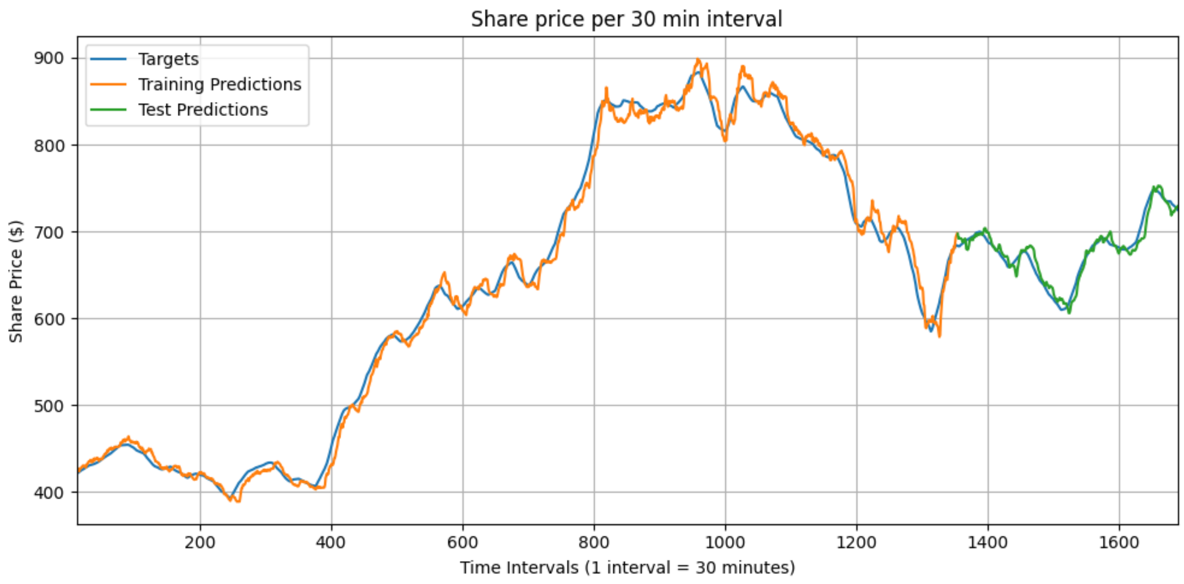


Figure 32: TSLA SMA percentage change converted to SMA for visualisation - training and testing predictions

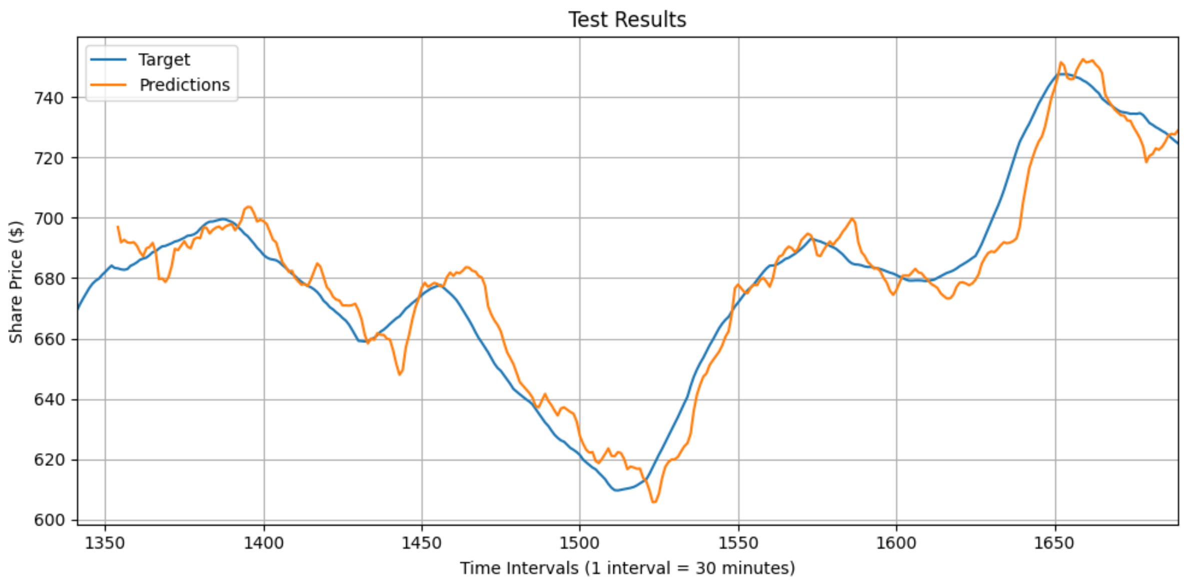


Figure 33: TSLA SMA percentage change converted to SMA for visualisation - testing predictions

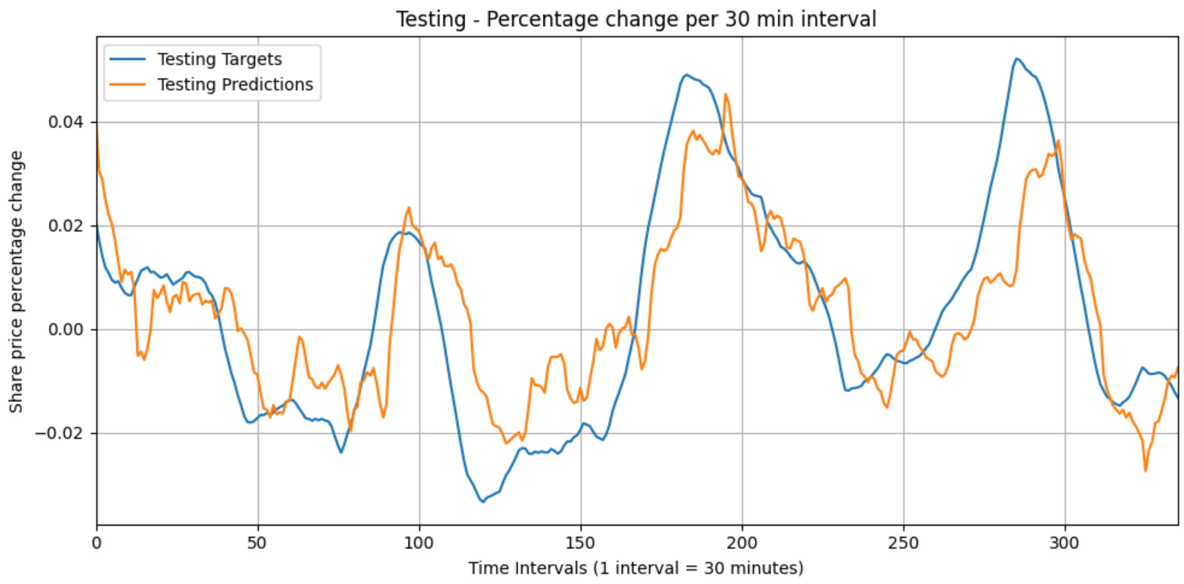


Figure 34: TSLA SMA percentage change - testing predictions

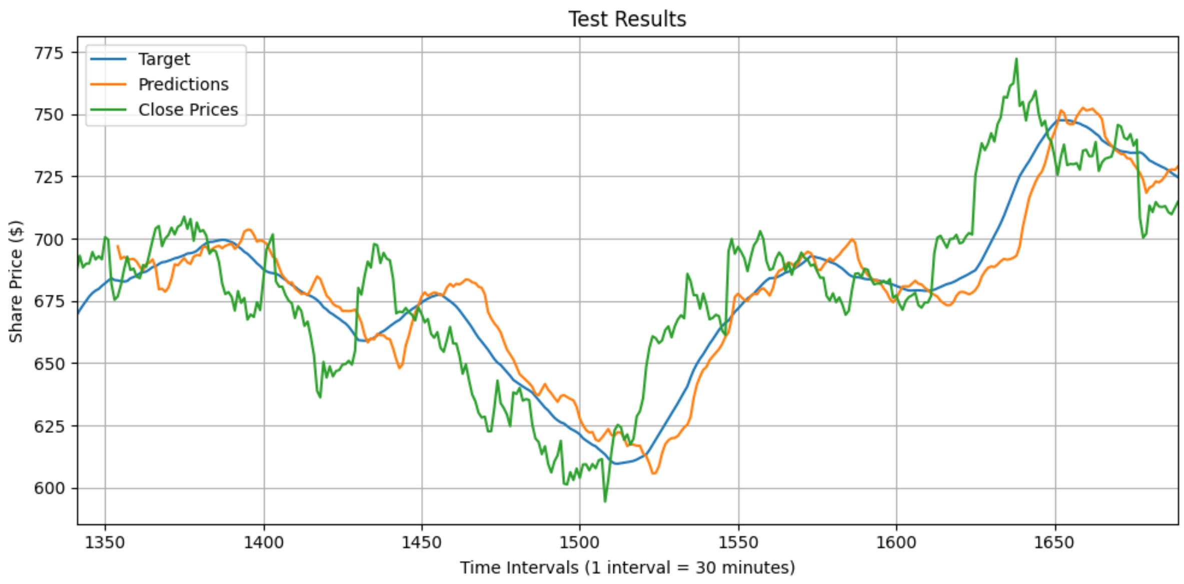


Figure 35: TSLA SMA displayed with with close prices - testing predictions

6.4 Stock Prediction - Architecture

This section compares the results of testing the different model architectures set out in section 5.3, extensive hyperparamter optimisation was conducted for each of these models. All models were used to predict the 1 day ahead percentage change of the simple moving average (2 day window size), the consistent output allowing the use of RMSE to compare them.

Figure 36: Table indicating the best performance achieved for each of the architectures tested where the validation loss is the average across the 4 validation folds and testing loss is calculated based on a testing dataset 20% the size of the base dataset.

Model Architecture	Average Validation Loss (RMSE)	Testing Loss (RMSE)
Base LSTM Model Uni-directional LSTM	0.1410	0.0856
Base LSTM Model Bi-directional LSTM	0.1363	0.0840
Split Sentiment Model Uni-directional LSTM	0.1410	0.0833
Split Sentiment Model Bi-directional LSTM	0.1318	0.0819
LSTM + CNN Model Uni-directional LSTM	0.1095	0.841
LSTM + CNN Model Bi-directional LSTM	0.1134	0.0884
LSTM + CNN Model Uni-directional LSTM Split Sentiment	0.1172	0.933
LSTM + CNN Model Bi-directional LSTM Split Sentiment	0.1151	0.0771
Only CNN	0.1292	0.1004

The validation losses presented in figure 36 provide the best evaluation metric between the different models since it's an indication of the models ability to generalise to different patterns in the data while the testing dataset could be potentially limited to one upward trend for example. Also since the models used aren't deterministic i.e. repeated runs don't produce the same model, there is some variation in loss when training/testing the same model hence the validation loss, which is the average calculated across 4 folds, provides a more accurate representation of model performance as compared to the testing loss. The testing loss was still left in however so as to provided a rough indication of what each model is capable of on unseen data.

The results from the table show that bi-directional LSTMs outperformed their uni-directional counter parts for the base and split sentiment models by having both a lower validation and lower testing loss. This was to be expected and is consistent with the related work, the performance gain is likely due to bi-directional LSTMs ability to learn the input data in both directions and hence make better use of the somewhat limited training data. However BLSTMs do seem to perform worse for the CNN architecture.

Surprisingly the split sentiment model, at least the version that used BLSTMs, beat the performance of the base LSTM model. Of all the architectures tested it seemed that the split sentiment model was the least likely to improve performance as the idea behind it was based on the least evidence. Although, interestingly it did achieve slightly worse validation loss when combined with CNN architecture. Based on the similar issue with BLSTMs it's likely due to the fact that both of these significantly increase the model complexity such that the training data available isn't sufficient to train the model to generalise well and hence is more likely to overfit the data.

The LSTM + CNN architecture saw the greatest performance across board for all the variations tested, specifically the LSTM + CNN model using uni-directional LSTMs achieved the best of those tested. This suggests that there is a trade off between the performance gained by adding to the architecture and how much the model overfits due to higher model complexity. This then implies that LSTM + CNN ensemble has the best ratio of prediction accuracy to model complexity and so there is a greater benefit to using this architecture and while the other two architectures improve performance on their own, combining all three adds too much complexity to the model hence the higher validation loss.

6.5 Stock Prediction - Hyperparameters

The full results of the hyperparameter tuning process can be found in the csv files submitted alongside this. Some highlights from this were that the top performing models favoured having fewer LSTM layers, 2 layer in particular provided the best performance. This is consistent with the findings that adding too much complexity to models caused them to overfit, increasing the validation losses. On the CNN side using stride lengths of anything above 1 tended to cause a loss in performance most likely because it resulted in skipping over too much of the data since the input matrix was only $13x21$. A few parameters weren't included in this testing such as the learning rate and batch size so as to keep the number of parameter combinations low enough to test in a reasonable length of time. However these parameters were still individually optimised, for example in the case of the learning rate this was done by inspecting plots of training loss to get a "healthy" looking learning curve alongside generally attempting to minimise validation loss.

6.6 Sentiment Features

While enabling/disabling different features was not extensively tested, the Stocktwits and or Twitter sentiment features were disabled for a few dozen runs to examine whether or not they contributed to making predictions.

Figure 37: Table indicating the best performance achieved for the different sentiment feature sets tested

Sentiment Features	Average Validation Loss (RMSE)	Testing Loss (RMSE)
Twitter and Stocktwits	0.1095	0.0841
Only Stocktwits	0.1264	0.0916
Only Twitter	0.1323	0.0822
None	0.1364	0.0819

The results in figure 37 clearly show that the sentiment features have a positive impact on the overall prediction accuracy which was expected and consistent with related work for intraday predictions. Similarly, from the data, the Stocktwits sentiment seems to have a greater impact compared to the Twitter data when used on it's own which makes sense considering that it was also used as training data for the sentiment analysis algorithm and as such the Stocktwits sentiment is likely and shown to be more accurate, on top of this a good portion of the Stocktwits sentiment used is from the data tagged by the users themselves. What wasn't expected was the roughly 25% increase in performance when using both Stocktwits and Twitter sentiment as features. One possible reason behind this is that these datasets seemed quite noisy, hence on there own, spikes or drops in sentiment might just be random noise, however, if this occurs in both datasets simultaneously it much more strongly indicates a change in the share price.

7 Discussion

The results of this project have shown that RoBERTa can be used to classify social media sentiment to a degree of accuracy which matches or exceeds that of previous work such as (Renault 2020). Unfortunately, it's not possible to make a direct comparison of performance due to slight differences in the datasets used. For example the fact that they don't use posts from a single stock but rather a mix of many which might effect results as seen by the difference in prediction accuracy between AMD and TSLA posts. More importantly however (ibid.) predominantly uses a balanced dataset to produce their results meaning that accuracy gives them a good measure of performance. In the case of this project this was not possible due to a lack of data available for individual stocks, resulting in unbalanced datasets hence the use of average macro F1 measure to compare results. However, the paper does also give some accuracies of around 82% for tests on a unbalanced dataset, which, while it might not exactly match the class balance of the datasets tested in this project, strongly suggests that RoBERTa outperforms the algorithms tested in the paper. Similarly, (Sohangir, Petty, and Wang 2018) suggests that VADER is the best algorithm for this task, however we have shown that RoBERTa outperforms this algorithm by a wide margin although there is still a lot of room for improvement.

The results also corroborates the finding of (Renault 2017) that intraday investor sentiment can help predict future stocks prices as indicated by the 25% decrease in loss when using the Twitter and Stockstwits data as opposed to just financial data.

On the model architecture side, while the results do show that bi-LSTMs outperform their uni-directional counterparts for stock prediction, which is consistent with the findings of (Althelaya, El-Alfy, and Mohammed 2018), however when incorporated into more complex models these performance gains disappear. We also found that CNNs in general perform better for the task of stock prediction than LSTMs which agrees with the findings of (Mehtab and Sen 2020) although an ensemble of these two neural network architectures provides the best performance. However as more data is collected it might be the case that the data is sufficient enough to train the more complex models e.g. combination of BLSTMs and CNNs and so experiments with these should not be ignored in future work on the topic.

Related works which similarly present the stock prediction task as a regression problem, such as (Persio and Honchar 2016) often show the same "horizontal offset" or lag in the results when directly predicting the close price, indicating that this is simply a very challenging problem to solve. While the solution that this project proposes in predicting the simple moving average price is still potentially useful to investors, as the 2 day window on the simple moving average values predicted effectively provides information on how the initial 1 day of known prices compares to the unseen latter half of that window. That being said, in future it might be better to tackle stock prediction as a classification problem e.g. is the stock price going to be higher 1 day ahead? Although perhaps a better solution might be a combination of the two with the classification side giving information on whether the stock price will increase or decrease and the simple moving average affirming that as well as giving an indication as to the scale of the price change.

Another focus for future work could be to use a language model for sentiment analysis that is specifically pre-trained on social media data rather than a general text corpus. This would give such a model a better initial understanding of language used in the posts being classified, TweetBERT is an example of such a model (Qudar and Mago 2020) which shows considerably better performance on a range Twitter datasets as compared to the RoBERTa model they tested. Unfortunately the model is yet to be released for public use so couldn't be used presently for this project. Regardless of the language model used, it would also be a good idea to try training the model on a balanced dataset of Stocktwits posts so as to improve the predictions of the negative sentiment class, which results showed were consistently worse. On top of this it could also give a better indication of performance which could be better compared to models tested in (Renault 2020).

Other than generally collecting data over a longer time period it would also be interesting to incorporate sentiment data from other sources such as the Yahoo Finance forums or Seeking Alpha considering the performance gained by combining just the Twitter and Stocktwits data. Similarly news headline sentiment has also been used in this context and while news headlines were collected for each of the stocks tested in this project there wasn't sufficient time to explore the use of this data.

Finally a more rigorous strategy for evaluating the performance of the various models would give a clearer picture of the results. While in this project over a thousand different parameter combinations were tested across the various models and the use of cross fold validation did help to minimise any effects of the variations in results, running tests multiple times for the same parameters would help improve the correctness of the results.

8 Conclusion

To conclude, this project has demonstrated that social media sentiment for specific stocks can be used to aid one day ahead, intraday share price prediction with a 25% difference in loss as compared to models using only financial data. In the process of this, two sentiment analysis algorithms were evaluated, VADER and RoBERTa. It was shown that RoBERTa substantially outperforms VADER, and based on related work strongly indicates that it also outperforms other machine learning approaches such as SVM's and Naive Bayes for this particular dataset. Likewise, different neural network architectures were tested for the actual task of stock price prediction and it was concluded that an ensemble approach using uni-directional LSTMs and CNNs achieves the best performance whereas more complex models tended to overfit the data, hurting the models ability to generalise.

This project has also shown that, while market volatility, particularly for intraday data makes direct predictions of share price an extremely difficult task, indirect representations of this price such as the simple moving average of the share price can be predicted to a high level of accuracy while still providing useful information to potential investors.

Finally, while this project does illustrate the potential use of social media sentiment in predicting stock prices and takes some steps towards producing results directly usable by investors to help their decision making, it does not present a finished trading strategy based on these results. Further work to achieve this, outside of collecting data from more sources over a longer time period and across more stocks, should focus on improving the performance of the sentiment analysis algorithms by testing different language models more specific to this task. Also, classification models of stock prediction have been shown to be quite accurate in other works and could be incorporated with the results from this project to present results with more confidence.

9 Appendices

9.1 Other sources of data

Outside of the Stocktwits and Twitter posts used in this project other data was also collected but wasn't used in the final product either due to time constraints or simply because that data wasn't relevant. The first of these was news headlines collected using the Bing News API individually for each of the stocks in section 5.1.5. Headlines were collected for the previous week every Saturday to avoid making too many API requests resulting in being rate limited. While the collection of this data was completed and still ongoing as of the end of this project it wasn't used due to time constraints as the focus was on testing the existing data and models.

The other main source of data collected, but not used, were comments from reddit posts on the r/wallstreetbets subreddit collected from late January 2021 to March. The motivation behind collecting this data was due to the explosion in popularity of this particular subreddit in relation to the trading of stocks such a Gamestop (GME) and AMC and it's self moderation in that any posts negatively talking about these stocks would be heavily down voted by users. This meant that any mention of these stocks with non negative up votes was most likely going to have positive sentiment and vice versa. While past prices of most of the popular stocks on the subreddit hadn't been collected during the course of the project and as such weren't really usable, the hope was that there would still be some discussion of more popular stocks such as TSLA. However while there were some mentions of stocks, particularly TSLA and AMD, it wasn't enough to warrant it's use in the project considering the data was only available for a small portion of the overall dataset collected. On the other hand, this did bring to the forefront the use of bots to manipulate such data, which could be a focus of future work i.e. detection of bots making posts or comments to push a specific agenda which could add a significant degree of noise to such datasets.

9.2 Trading Strategy

As mentioned in section 6.1.2, a simple trading strategy was developed originally for use when directly predicting the close prices of stocks. This strategy worked by starting with an arbitrary amount of $10,000 and then for each prediction calculating the potential loss/gain from buying shares at their current price compared to their value given the prediction. If it was determined that a profit could be made including accounting for a trading fee set at $6 then it would simulate buying the maximum number of shares possible within the $10,000 budget, likewise if a lose could be avoided then a sell signal was generated. When directly predicting the close price the strategy made trading decisions based on the weighted predicted profits from the past 3 predictions with more recent predictions being more heavily weighted. This was implemented to prevent the constant selling/re-buying of shares due to volatility in the data however it was later discarded when predicting the simple moving average as that output effectively achieved the same thing but much more successfully. In the end this strategy wasn't used to help evaluate the different models primarily due to the switch to predicting simple moving average prices meaning that the strategy didn't work as well but also because there was a significant degree of luck involved, however in general it did appear that using this strategy with the predictions provided by the models did more often than not outperform a buy and hold strategy.

References

Althelaya, K. A., E. M. El-Alfy, and S. Mohammed (Apr. 2018). “Evaluation of Bidirectional LSTM for Short-and Long-Term Stock Market Prediction”. In: *2018 9th International Conference on Information and Communication Systems (ICICS)*. 2018 9th International Conference on Information and Communication Systems (ICICS), pp. 151–156. DOI: 10.1109/IACS.2018.8355458.

Bollen, Johan, Huina Mao, and Xiao-Jun Zeng (Mar. 2011). “Twitter Mood Predicts the Stock Market”. In: *Journal of Computational Science* 2.1, pp. 1–8. ISSN: 18777503. DOI: 10.1016/j.jocs.2010.12.007. arXiv: 1010.3003. URL: http://arxiv.org/abs/1010.3003 (visited on 03/26/2021).

Chen, Chung-Chi, Hen-Hsen Huang, and Hsin-Hsi Chen (2018). *NTUSD-Fin: A Market Sentiment Dictionary for Financial Social Media Data Applications*. URL: /paper/NTUSD-Fin%3A-A-Market-Sentiment-Dictionary-for-Social-Chen-Huang/d97ee125f7f92741e6d05549fc4 (visited on 10/28/2020).

Devlin, Jacob et al. (May 24, 2019). *BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805 [cs]. URL: http://arxiv.org/abs/1810.04805 (visited on 01/18/2021).

Fama, Eugene F. (1965). “The Behavior of Stock-Market Prices”. In: *The Journal of Business* 38.1, pp. 34–105. ISSN: 0021-9398. JSTOR: 2350752.

Hutto, C. and E. Gilbert (2014). “VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text”. In: *ICWSM*.

INTERSPEECH 2014 Abstract: Fan et Al. (2021). URL: https://www.isca-speech.org/archive/interspeech_2014/i14_1964.html (visited on 04/13/2021).

K.Nirmaladevi and N. Krishnamoorthy (Dec. 17, 2019). “Social Media Aided Sentiment Analysis for Stock Prediction”. In: *International Journal of Innovative Technology and Exploring Engineering* 9, pp. 112–116. DOI: 10.35940/ijitee.A5062.129219.

Karpathy, Andrej, Justin Johnson, and Li Fei-Fei (Nov. 16, 2015). *Visualizing and Understanding Recurrent Networks*. arXiv: 1506.02078 [cs]. URL: http://arxiv.org/abs/1506.02078 (visited on 03/31/2021).

Liu, Yinhan et al. (July 26, 2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. arXiv: 1907.11692 [cs]. URL: http://arxiv.org/abs/1907.11692 (visited on 03/23/2021).

Mehtab, Sidra and Jaydip Sen (Oct. 21, 2020). *Stock Price Prediction Using CNN and LSTM-Based Deep Learning Models.* arXiv: 2010.13891 [cs, q-fin]. URL: http://arxiv.org/abs/2010.13891 (visited on 04/08/2021).

Mittal, Anshul and Arpit Goel (2012). “Stock Prediction Using Twitter Sentiment Analysis”. In:

Oliveira, Nuno, Paulo Cortez, and Nelson Areal (May 1, 2016). “Stock Market Sentiment Lexicon Acquisition Using Microblogging Data and Statistical Measures”. In: *Decision Support Systems* 85, pp. 62–73. ISSN: 0167-9236. DOI: 10.1016/j.dss.2016.02.013. URL: http://www.sciencedirect.com/science/article/pii/S0167923616300240 (visited on 10/17/2020).

Persio, L. and O. Honchar (2016). “Artificial Neural Networks Architectures for Stock Price Prediction: Comparisons and Applications”. In: *undefined.* URL: /paper/Artificial-Neural-Networks-architectures-for-stock-Persio-Honchar/41487776364a6dddee91f1e2d2b (visited on 03/29/2021).

Prakash, Anushka and Harish Tayyar Madabushi (Dec. 2020). “Incorporating Count-Based Features into Pre-Trained Models for Improved Stance Detection”. In: *Proceedings of the 3rd NLP4IF Workshop on NLP for Internet Freedom: Censorship, Disinformation, and Propaganda.* COLING-NLP4IF 2020. Barcelona, Spain (Online): International Committee on Computational Linguistics (ICCL), pp. 22–32. URL: https://www.aclweb.org/anthology/2020.nlp4if-1.3 (visited on 04/12/2021).

Qudar, Mohiuddin Md Abdul and Vijay Mago (Oct. 16, 2020). *TweetBERT: A Pretrained Language Representation Model for Twitter Text Analysis.* arXiv: 2010.11091 [cs]. URL: http://arxiv.org/abs/2010.11091 (visited on 03/30/2021).

Renault, Thomas (Nov. 1, 2017). “Intraday Online Investor Sentiment and Return Patterns in the U.S. Stock Market”. In: *Journal of Banking & Finance* 84, pp. 25–40. ISSN: 0378-4266. DOI: 10.1016/j.jbankfin.2017.07.002. URL: https://www.sciencedirect.com/science/article/pii/S0378426617301589 (visited on 03/29/2021).

— (Sept. 1, 2020). “Sentiment Analysis and Machine Learning in Finance: A Comparison of Methods and Models on One Million Messages”. In: *Digital Finance* 2.1, pp. 1–13. ISSN: 2524-6186. DOI: 10.1007/s42521-019-00014-x. URL: https://doi.org/10.1007/s42521-019-00014-x (visited on 11/05/2020).

Sezer, Omer Berat, Murat Ozbayoglu, and Erdogan Dogdu (Jan. 1, 2017). “A Deep Neural-Network Based Stock Trading System Based on Evolutionary Optimized Technical Analysis Parameters”. In: *Procedia Computer Science.* Complex Adaptive Systems Conference with Theme: Engineering Cyber Physical Systems, CAS October 30 – November 1, 2017, Chicago, Illinois, USA 114, pp. 473–480. ISSN: 1877-0509. DOI: 10.1016/j.procs.

2017.09.031. URL: https://www.sciencedirect.com/science/article/pii/S1877050917318252 (visited on 04/13/2021).

Sohangir, Sahar, Nicholas Petty, and Dingding Wang (Jan. 1, 2018). *Financial Sentiment Lexicon Analysis*, p. 289. 286 pp. DOI: 10.1109/ICSC.2018.00052.

Understanding LSTM Networks – Colah's Blog (2021). URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 03/31/2021).

Vaswani, Ashish et al. (Dec. 5, 2017). *Attention Is All You Need*. arXiv: 1706.03762 [cs]. URL: http://arxiv.org/abs/1706.03762 (visited on 03/24/2021).