Imperial College London

Department of Electrical and Electronic Engineering

Networks for sentiment analysis in the cryptocurrency market

Author:

Baptiste PROVENDIER

CID:

01553706

Project Supervisor:

Dr. Stefan VLASKI

Second Marker:

Dr. Wei DAI

June 2022

Final Report Plagiarism Statement

[illegible]

22/06/2022

Abstract

Machine learning and neural networks have found extensive uses in financial institutions for time series forecasting. Whilst getting information used to be slow and difficult, social media has revolutionised access to financial news. The rise of cryptocurrencies has offered investors a new market open 24 hours a day with an extremely reactive asset class due to its volatile nature. This project aims to build a complete algorithm capable of analysing sentiments of social media posts using neural networks and other machine learning techniques in order to predict cryptocurrency prices. Furthermore, a strategy will be developed in order to profitably trade cryptocurrencies and outperform current techniques. Some state of the art techniques have previously attempted to predict short term price movement in Bitcoin using machine learning [38], and even using twitter sentiment data to improve accuracy [75]. This project aims to build a complete algorithm from data collection to trading which will use customised sentiment features and financial evaluation metrics. Over the course of one month of testing, this method generated an average return between $10 - 150\%$ thanks to some optimisation techniques defined along the report.

Acknowledgements

I would like to thank Dr. Stefan Vlaski who supervised me during this interesting project and provided me great guidance and support throughout my research. Expressing ideas and discussing with him really helped me move forward in this project and has proved to be a precious experience for me.

I would also like to thank all my friends at Imperial College who, consciously and unconsciously, supported me throughout my studies and with whom I had insightful discussions.

Finally, I would like to thank my parents and family for their unconditional love and support both academically during my four years at Imperial College but most importantly emotionally.

Contents

1 Introduction

In 2013, an Associated Press tweet claiming an "explosion" had injured President Obama caused a stock market crash worth $130 billion in a matter of minutes [62]. AP said its Twitter account was hacked and the market later recovered that day, but it showed the power of news and social media on stock prices and the influence that it can have. Assets like stocks or cryptocurrencies are influenced by a variety of factors dictated by the supply and demand from buyers and sellers. Jim Simmons and his Renaissance fund is a famous example of using data to predict the short-term variability in stock prices. The strategy is to gather all the data available to make mathematical models that find repeated anomalies that can predict short term price movement. With the recent digitisation of media, information is spreading quicker than ever, and social media has completely changed the way investors get access to financial news. In a 2007 paper discussing the impact of investor's sentiment on stock market prices, Baker and Wurgler argued that "the question is no longer, as it was a few decades ago, whether investor sentiment affect stock prices, but rather how to measure investor sentiment and quantify its effects" [13]. In 2021, the S&P Dow Jones indices released an index designed to track the performance of the 200 companies that had the best Twitter sentiment which has consistently beat the market over the last couple of years [68]. Tweets sentiment has been a metric difficult to ignore when it comes to financial decisions and price predictions. If the stock market is influenced by the sentiment of financial tweets, there is another category of assets that is even more impacted by these: cryptocurrencies. The latter has often been described as the most volatile asset [63] due to its speculative nature and the fact that its market never closes meaning it is very reactive to news all over the world at any time of the day. As demonstrated in extensive recent works [9] [10] [41], sentiment analysis has been shown to be a powerful feature when it comes to cryptocurrency price forecasting. It is not hard to see the impact that tweets can have on cryptocurrency prices when an Elon Musk tweet about Dogecoin caused the price to rally over 30% in minutes [18] as seen in Figure 1.

Due to its instantaneity, Twitter has become the platform of choice for people to share their financial advice and opinions which makes tweets' sentiment analysis a valuable data when it comes to predicting the short-term price movement of cryptocurrencies [41] [9]. Previous state-of-the-art methods have achieved encouraging results yet a number of issues are still to be solved and further research effort is required. Some of these issues include noisy data and the use of basic binary classifiers for sentiment analysis, which does not include the polarity of the expressed opinions. Most previous works also tend to evaluate performances based on traditional error minimisation techniques while we will propose alternative financial evaluation metrics. The intent of this project is hence to use Twitter sentiment analysis as an input to neural networks in order to predict cryptocurrency short term price movement and develop

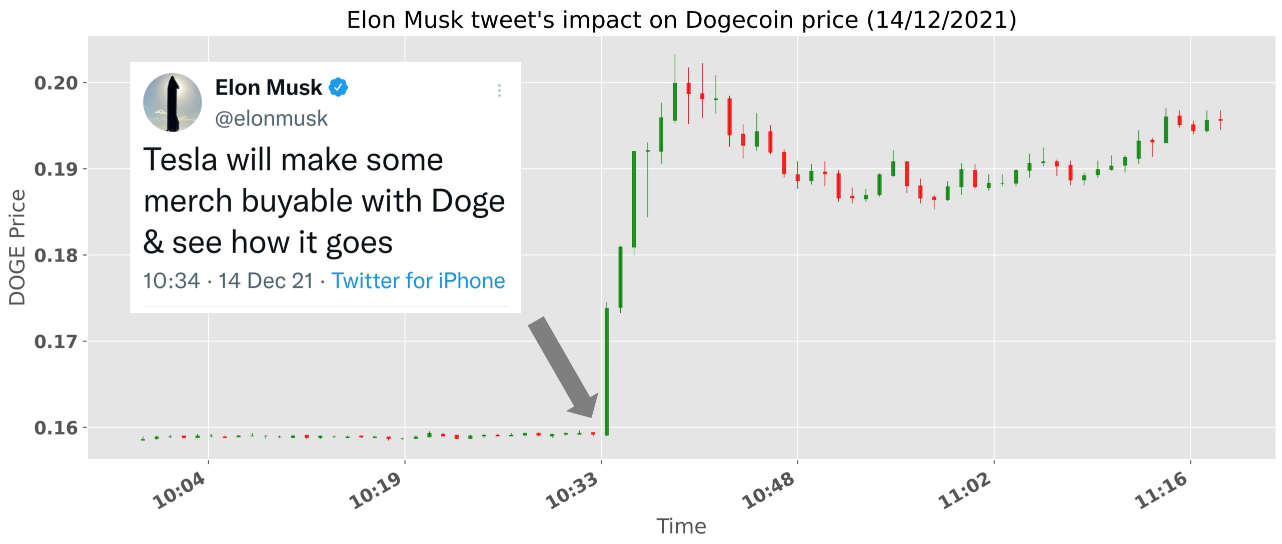


Figure 1: Elon Musk tweet's impact on Dogecoin price (14/12/2021).

a profitable trading strategy from it. Most models tend to use unreliable sentiment analysis methods with poorly selected datasets. Tweets are often duplicated for marketing purposes and may also be automated by bots [73]. Tweets also typically contain features that result in noise (when it comes to sentiment analysis) including hashtags, profile mentions and URLs [41]. Therefore, efforts have to be put to tackle the general problem of data gathering, sentiment analysis as well as pre-processing of tweets to reduce noise and obtain more meaningful features. Furthermore, some models tend to focus on reducing model errors [38] in order to conduct predictions and not on any real-life financial metrics to optimise their models. In this paper, we propose a complete algorithm which gathers tweets from Twitter, pre-processes them and analyses their sentiment. This feature is then used, along with other financial data, into a deep learning algorithm in order to predict not only the direction of price changes but also the magnitude and build a profitable trading strategy based on these recommendations. To the best of our knowledge, this paper is the first to propose such a complete state of the art system.

In the following sections we will cover related literature in section 2.1 and compare their approaches with our project, the key distinctions and how we aim to differentiate ourselves in section 2.2. Secondly, we will cover some historical and technical background on the different topics covered in the project in sections 2.3-2.9. Afterwards, we will set up the design analysis and cover the different sections of the project in 3. Next, we will study the implementation of these sections as well as the improvements made along the way in 4. In addition, an observation of the algorithm's results and performances will be performed as well as a critical evaluation of these results comparatively to the previously discussed strategies in sections 5 and 6. Finally, we will discuss the success of the projects along with the difficulties encountered and the future work considerations in sections 7 and 8.

2 Background

2.1 Related Litterature

This section will conduct a literature review of similar pieces of work that have been conducted in the past. We will highlight the techniques used and the conclusions observed from them. We will later study some weaknesses in the current approaches, the differences compared to the aim of this report and the way our work will overcome these weaknesses.

Social media has been the preferred way of sharing information about cryptocurrencies. Being the first cryptocurrency [53] and remaining the biggest of all by marketcap, most of the studies and research have focused on the impact of social media sentiments of Bitcoin. In 2017, Phillips et al. [60] used statistical models usually used to detect epidemic outbreaks of influenza (Hidden Markov Model) to predict cryptocurrency price bubbles. Using only social media usage related to selected cryptocurrencies and trading volume, their model was able to return over 1000% in one year, over double the overall benchmark returns during that period. However, their model was tested on selected cryptocurrencies which had experienced unusual exponential behaviour and cannot achieve similar performance in a more mature market with bigger market capitalisation cryptocurrencies such as 2022's Bitcoin. Colianni et al. [20] used a Naïve Bayes model to classify Twitter sentiments and find optimal trading time by achieving over 95% accuracy. Their analysis was focused on the price change movement and not the magnitude and was conducted over a short time period in a less mature cryptocurrency market. Georgoula et al. [31] used SVM (support vector machine), Twitter sentiment analysis and other regression models to forecast price fluctuations of Bitcoin. With a mean squared error of only 0.14%, they found strong short-term correlation between positive "measurements of collective mood based on sentiment analysis" and price fluctuation of Bitcoin. By using extreme gradient boosting regression tree model, Li et al. [42] showed that Twitter sentiment analysis and trading volume can be used as a leading indicator for price fluctuations in the cryptocurrency market. More importantly, the study shows that social media platforms are able to capture investor's sentiment, and that sentiment is an "early signal to future price fluctuations in cryptocurrencies". While this paper did not intend to predict prices, it highlights the correlation that exists between the different features that we are going to use. In a research paper from 2018, Mai et al. [46] found that social media sentiment analysis is "an important leading indicator of the future price swings". Finally, in 2015, Garcia and Schweitzer [28] demonstrated that changes in Twitter sentiment polarity preceded Bitcoin price fluctuations using vector autoregressive model. These confirm that tweets' sentiments and polarity can be used as leading indicators to predict changes in direction and magnitude. A recent research in May 2022 [76] aimed to use Twitter sentiment analysis as well as data volume to combine them with lagged price features in order to predict not only Bitcoin price direction but also magnitude of change with the use of different deep learning techniques. They managed to obtain over 60% accuracy in predicting magnitude of changes and we will be using their as a benchmark.

These studies not only show the strong correlation that exists between cryptocurrency, and in particular Bitcoin, price fluctuation and its related Twitter sentiment analysis, it also proves that the latter precedes the price change meaning that it is possible to predict the changes in price and build a profitable strategy from the analysis of sentiments of tweets and their overall polarity for a given period. This report aims to support that hypothesis and the following sections will explain the concepts covered in the project as well as the implementation and evaluation plans.

It is worth noting that these findings have recently been contested by new research. Findings from a recent 2022 research conducted by data scientists and machine learning engineers [43] have found conflicting evidence drawing no clear correlation between cryptocurrency price movement and their sentiment analysis. In section 6.2 we will investigate the role that sentiment features play in the model final predictions and draw conclusions on this correlation.

2.2 Motivations and Objectives

Financial time series prediction is naturally appealing due to the potential profits and it is an active field of research. As mentioned in section 2.1, extensive work has already attempted to forecast prices with the highest accuracy possible. Basic time series forecasting models, like the autoregressive integrated moving average (ARIMA) model, tend to perform well in terms of traditional error and accuracy metrics but are often simple lagging models. A lagging model is a model that reacts to price changes only after they occur, while a leading model is one with the ability to provide guiding insight to how a price would change before it occurs. In Figure 2, we can observe Bitcoin prices compared with a 1-period lag signal. We can evaluate this model using only the metrics used for most models and we obtain a MSE of less than 0.3%. However, in practice, such models are useless and cannot be used to make money as they offer no insight into future price action. This is why this paper will aim to create a model based on real-life financial metrics such as return on investment and sharpe ratio. This also leads to the choice of using neural networks in order to detect complex patterns in non-stationary data. Some papers also struggle with the sentiment analysis part which limits the quality of the data and hence the predictions. We will therefore use more recent (2018), state-of-the-art natural language processing (NLP) methods [22]. Finally, the main motivation of this project is to build a complete autonomous system from the data collection, to the sentiment analyser, to the price prediction and to the trading strategy and signals.

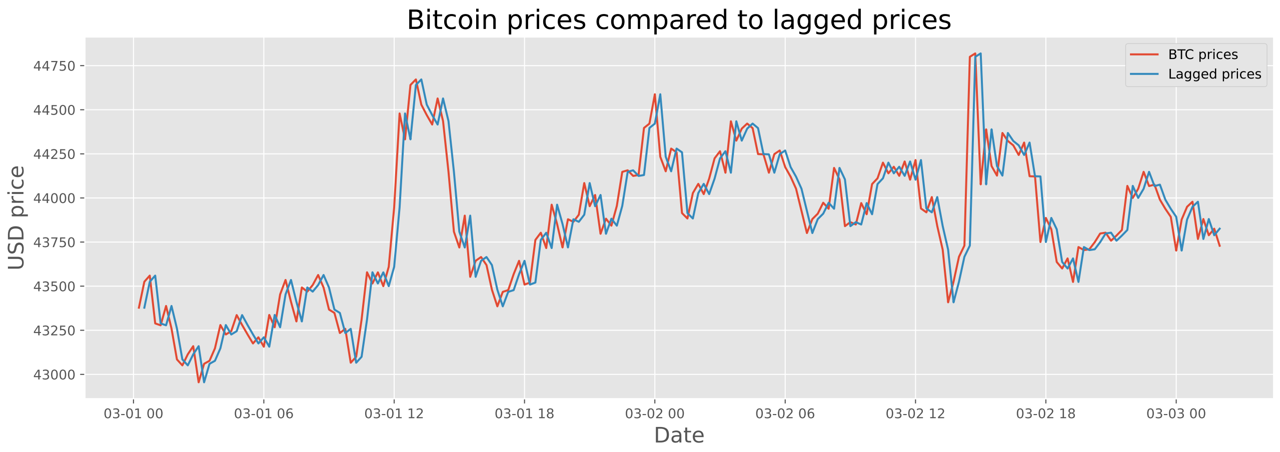


Figure 2: comparison of Bitcoin prices over 200 periods with a 1 period lagged signal.

The objective of this project relies on a profitable trading strategy using a complete price prediction algorithm. The project will then be evaluated on the completion of the end-to-end algorithm as well as the financial performance against a set of benchmarks. These benchmarks are going to be a buy-and-hold approach; which consists in buying the asset at the beginning of the test period and selling it at the end and evaluate performance, as well as some state of the art models implemented on the same time period.

2.3 Financial forecasting and technical analysis

An accurate prediction of asset prices or relevant financial data, commonly referred to as financial forecasting, has significant benefits in providing a competitive advantage to investors and decision makers. Therefore, financial signal processing and machine learning is an important research topic both in academia and in the industry lured by potential economic profits. In addition to classical models governed by analytic methods such as the Black Scholes equation [17], a large amount of research has been done using more data-driven models such as support vector machines (SVM) [37] and neural networks (NN) [12]. Market participants and investment banks have been working for decades to try and determine the direction of an asset price in order to trade profitably. Determining whether an asset price will go up or down is a challenging task due to the many uncertainties involved and many variables that influence the market. Therefore, a slight edge over a random guess, that is even a couple of percentages over a 50% accuracy, would give an investor a significant advantage in the market and considerable potential profits.

The most prevalent type of analysis used to perform financial forecasting is called "technical analysis" [76]. Technical analysis is defined to be the "study, practice, and analysis of chart patterns, indicators and oscillators, and the candlesticks themselves that make up price charts of assets" [8]. Born in the late 1800's from the Dow Theory [69], technical analysis is used as a tool to guide a trading decision based on past market data as well as technical indicator such as the relative strength index (RSI), which we will cover later in section 2.7.1. Since the beginning of the 1980's, algorithmic trading, or the process of trading using pre-programmed algorithms, has grown in popularity. Not only do these algorithms make faster and more accurate predictions, they also remove the human emotions out of the trading decision making [50]. The main emotions leading to irrational decisions in market trading are greed and fear. An algorithm which trades on unbiased technical data is hence desirable and has been widely used by the world's biggest investment banks like Goldman Sachs [5].

2.4 Neural Networks

Before introducing the different deep learning models for time series prediction, it is useful to recall the classical machine learning analysis that preceded them. Times series prediction has traditionally been performed by the use of linear models including metric autoregressive (AR), moving average (MA) or autoregressive integrated moving average (ARIMA) [61] [47]. The parameters estimated were accounting for the main characteristics of time series like seasonality and long-term trend in order to determine the autocorrelation embedded in the data. However, this type of process is linear and is not able to cope very well with nonstationary data. Financial data tend to be non-stationary time series as they have means and variances that vary over time due to a combination of trends and cycles in the market. Due to their high volatility, cryptocurrency prices and returns tend to have heavy tail representation [57] with strong non-linear behaviour. Hence, as big price movements like market crashes, happen more often than predicted given a Gaussian model, the probability distribution of cryptocurrency returns is non-Gaussian [52].

These difficulties in predicting time series pushed researchers to develop new models in the 1990s. Deep neural networks were invented to replicate the functioning of the human brain and recognising complex patterns in data. Neural networks approach time series prediction as a non-parametric task [49]. This means that it does not need to know any information regarding the process that generates the signal, and the prediction is entirely made by inference of future behaviour from past behaviour. In its most basic form, a recurrent system can be described using equations 1 and 2. $s^{(t)}$ is the model's state at time t and is a function of the previous state $s^{(t-1)}$, the external input signal $x^{(t)}$ and the network's parameters θ. Since recurrent networks use hidden states, denoted $h^{(t)}$ which can be thought to be intermediary connections between the inputs and outputs of a model, equation 2 is a simple mathematical representation.

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}, \theta) \tag{1}$$

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}, \theta) \tag{2}$$

A neural network is a network of interconnected group of perceptrons, a simple model that attempts to replicate the functioning of a neuron receiving information and binarily deciding whether to send that information forward or not, that transmit input values layer by layer until it reaches the output. Along the layers, the weights, which correspond to the importance of the features carried by the perceptron nodes, are updated to try and minimise the error. In 1994, Lizhong Wu had concluded that "for a non-Gaussian signal [...], nonlinear prediction will always achieve better predictive gain than linear prediction" [77]. Professor Mandic summarises that using neural networks for prediction rather than classical time series analysis was "computationally at least as fast, more accurate and able to cope with the nonlinearity and nonstationarity of input" [49].

However, traditional artificial neural networks or convolutional neural networks are not used for time series predictions as they miss some key properties. The first problem arises from the fact that traditional neural networks could only handle fixed size input, not varying size data which is the case of most time series. The second, and major, problem was the sequential data problem. Most of the real-life data, including cryptocurrency prices, include temporal correlation of the data and hence need a dynamic model to be able to perform predictions [72]. Regular neural networks are feed forward, meaning that the input is propagated layer by layer until the output without remembering the proceeding output values. Feed forward networks do not consider any dependencies in the data and the solution to that problem is recurrent neural networks [54]. Recurrent Neural Networks (RNN) introduce a circular feedback element which makes it appropriate for time series forecasting [49]. The decision outcome at time $t-1$ influences the decision at time t as the network maintains a reference historical analysis at any given point [54]. Figure 3 is a schematic of an unrolled recurrent neural network.

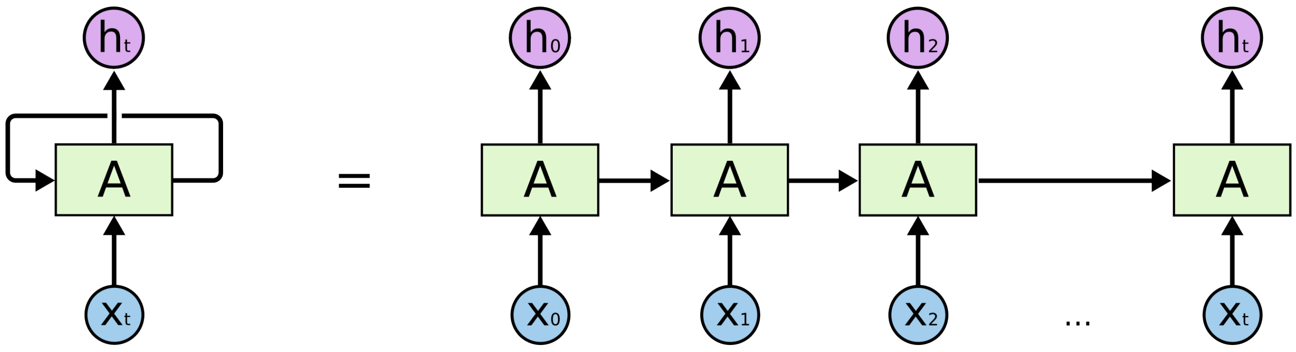


Figure 3: Schematic of an unrolled RNN [40]

We will now develop the forward propagation equations from the RNN depicted in figure 3 in their basic form. A recurrent neural network maps an input sequence x to a corresponding output sequence o by recursively computing a training loss L which measures how far each o is from the target value y [32]. The loss L internally computes $\hat{y}$, which simply applies an activation function to the output o. From the loss function, we can derive the gradients which are used to update the weights [66]. An activation function, or transfer function, is simply a mathematical function which decides how much the neuron should be activated or not and whether its should be passed onto the next layer of the network or not [30]. Some examples of popular transfer functions are the linear activation, the ReLU activation or the hyperbolic tangent activation function. First, the RNN has the input to hidden connections parameterised by a weight matrix $\mathbf{U}$ and the hidden-to-hidden recurrent connections are parameterised with another weight matrix $\mathbf{W}$ as seen in equation 3. Then, the hidden-to-output connections are parameterised by a final weight matrix $\mathbf{V}$ as shown in equation 5. Equation 3 defines the forward propagation of a recurrent neural network model.

$$a^{(t)} = b + \mathbf{W}h^{(t-1)} + \mathbf{U}x^{(t)}, \tag{3}$$
$$h^{(t)} = \sigma_1(a^{(t)}), \tag{4}$$
$$o^{(t)} = c + \mathbf{V}h^{(t)}, \tag{5}$$
$$\hat{y}^{(t)} = \sigma_2(o^{(t)}), \tag{6}$$

From the equations above, we notice that the current hidden state $h^{(t)}$ is defined by five parameters. The current input $x^{(t)}$, the previous hidden state $h^{(t-1)}$, the bias vector b as well as the two weight matrices $\mathbf{U}$ and $\mathbf{W}$. The fact that the current hidden state depends on the previous hidden state is the element that brings memory to the network and makes the model suitable for time series prediction. This equation is then passed into an activation function to obtain $h^{(t)}$. The current output $\hat{y}^{(t)}$ depends on the current hidden state as well as another bias vector c and a final weight matrix $\mathbf{V}$ through a second activation function [32].

Although recurrent neural networks solve many problems that most other neural networks are not able to solve, it faces some inconveniences too. The biggest of them being long-term dependencies and the so-called ”vanishing or exploding gradient problem” [32]. By repeatedly applying the same operation at each time step of a long temporal sequence, RNNs construct deep computational graphs with repeated applications of the same parameters. We will illustrate this problem with the following equations. The recurrent relation employed in recurrent neural networks can be summarised by equation 7. For visualisation simplification, we remove the nonlinear activation function and the inputs x.

$$h^{(t)} = \mathbf{W}h^{(t-1)} \tag{7}$$

Recurrently applying this relation up to $h(0)$ gives us equation 8:

$$h^{(t)} = \mathbf{W}^t h^{(0)} \tag{8}$$

If the weight matrix $\mathbf{W}$ admits eigendecompostion of the form:

$$\mathbf{W} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T \tag{9}$$

With $\mathbf{Q}$ being an orthogonal matrix, the recurrence may be simplified even further to equation 10:

$$h^{(t)} = \mathbf{Q}\mathbf{\Lambda}^t\mathbf{Q}^T h^{(0)} \tag{10}$$

The eigenvalues in the term $\mathbf{\Lambda}$ are raised to the power of t, causing the eigenvalues with magnitude less than one to decay to zero and the eigenvalues with magnitude greater than one to explode to infinity [32]. Any component of $h^{(0)}$ that is not aligned with the largest eigenvector will eventually be discarded. Because of this problem it was shown that whenever a model is able to represent long term dependencies, the "gradient of a long-term interaction has exponentially smaller magnitude than the gradient of short term interaction" [15]. To solve these issues, LSTM (Long Short Term Memory) were invented. The keys to LSTM are the cell state that moves information along without changing it, represented by the straight horizontal line at the top of the sketch, and the gates that control the information flowing alongside as shown in the schematic in Figure 4.

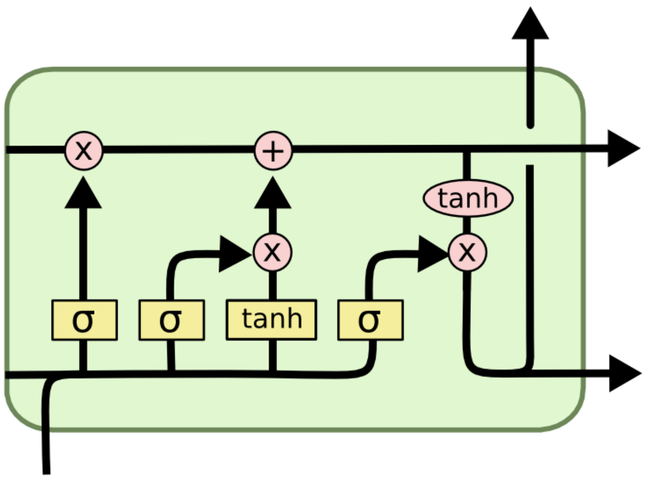


Figure 4: Schematic of an LSTM model [56]

The three gates, symbolised by σ from left to right are the forget, input and output gates, control which information are important to remember, and which are not, in order to only keep the necessary data. The key difference which LSTM use to get rid of vanishing gradient comes from the forget gate, the one on the left in Figure 4, that selects which information to be kept. When computing the gradient of the hidden state in a traditional recurrent neural network, one of the terms is going to be the weight of the state. If the weight is lower than 1, the gradient will converge to 0 over time. Looking back at vanilla RNNs described in equation 4, we can define the hidden state $h^{(t)}$ at time-step t to be:

$$h^{(t)} = \sigma(b + \mathbf{W}h^{(t-1)} + \mathbf{U}x^{(t)}) \tag{11}$$

Removing the input component for visualisation purposes [59], we can show that the gradient of the RNN hidden state is :

$$\frac{\partial h^{(t')}}{\partial h^{(t)}} = \prod_{k=1}^{t'-t} \mathbf{W}\sigma'(\mathbf{W}h^{(t'-k)}) \tag{12}$$

$$= \mathbf{W}^{(t'-t)} \prod_{k=1}^{t'-t} \sigma'(\mathbf{W}h^{(t'-k)}) \tag{13}$$

The first term $\mathbf{W}^{(t'-t)}$ in equation 13 is the reason for vanishing and exploding gradients in regular recurrent neural network. As we compute the gradient of the hidden state to perform error backpropagation, if the weights is not equal to 1, then it will either decay to zero exponentially fast in $(t' - t)$ or converge exponentially fast to infinity [14].

LSTMs introduce a self-loop to produce paths where the gradient can flow for long duration where the weight of this self-loop is gated. This means that even if the parameters of the LSTM are fixed, the time scale of integration can be changed based on the input sequence controlling the flow of information through a system of gating units [32]. In this case, the self-loop weight is controlled by the forget gate unit, denoted $f^{(t)}$, which sets this weight to a value between 0 and 1 via a sigmoid unit:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j \mathbf{U}_{i,j}^f x_j^{(t)} + \sum_j \mathbf{W}_{i,j}^f h_j^{(t-1)}\right) \tag{14}$$

where $x^{(t)}$ is the current input vector and $h^{(t)}$ is the current hidden layer vector containing all the outputs of all the LSTM cells, and b^f, $\mathbf{U}^f$, $\mathbf{W}^f$ are respectively biases, input weights and recurrent weights for the forget gates. The LSTM cell internal state is thus updated as follows, but with a conditional self-loop weight $f_i^{(t)}$:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j \mathbf{U}_{i,j} x_j^{(t)} + \sum_j \mathbf{W}_{i,j} h_j^{(t-1)}\right) \tag{15}$$

where b, $\mathbf{U}$, $\mathbf{W}$ respectively denote the biases, input weights and recurrent weights into the LSTM cell and $g_i^{(t)}$ is the external input gate unit which is computed similarly to the forget gate but with its own parameters. Within the cell state $s_i^{(t)}$, described in equation 15, the gradient of a state with respect to its immediate predecessor $\frac{\partial s^{(t)}}{\partial s^{(t-1)}}$ is the only part where gradients flow through time and we can observe if the phenomena of vanishing gradient still persists. It is described as:

$$\frac{\partial s^{(t)}}{\partial s^{(t-1)}} = \frac{\partial f^{(t)} s^{(t-1)}}{\partial s^{(t-1)}} \tag{16}$$

$$= \frac{\partial \left[\sigma\left(b^f + \sum_j \mathbf{U}^f_j x^{(t)}_j + \sum_j \mathbf{W}^f_j h^{(t-1)}_j\right)\right] s^{(t-1)}}{\partial s^{(t-1)}} \tag{17}$$

$$= s^{(t-1)} \frac{\partial \left[\sigma\left(b^f + \sum_j \mathbf{U}^f_j x^{(t)}_j + \sum_j \mathbf{W}^f_j h^{(t-1)}_j\right)\right]}{\partial s^{(t-1)}} + \tag{18}$$

$$\frac{\partial s^{(t-1)}}{\partial s^{(t-1)}} \left[\sigma\left(b^f + \sum_j \mathbf{U}^f_j x^{(t)}_j + \sum_j \mathbf{W}^f_j h^{(t-1)}_j\right)\right] \tag{19}$$

$$= \sigma\left(b^f + \sum_j \mathbf{U}^f_j x^{(t)}_j + \sum_j \mathbf{W}^f_j h^{(t-1)}_j\right) \tag{20}$$

and thus with $t' > t$:

$$\frac{\partial s^{(t')}}{\partial s^{(t)}} = \prod_{k=1}^{t'-t} \sigma\left(b^f + \sum_j \mathbf{U}^f_j x^{(t+k)}_j + \sum_j \mathbf{W}^f_j h^{(t-1+k)}_j\right) \tag{21}$$

the gradient of the cell state of an LSTM only depends on the activation vector to its forget gate [14] which helps the network to better regulate the gradient values at each time step by updating the forget gate's parameters, as described in equation 21. The activation of the forget gate allows the LSTM to select to what extend a particular information should be remembered at each time step and update the model's parameters accordingly. That way, any time step, the model can find an appropriate parameter update of the forget gate such that the gradient does not disappear. The existence of the forget gate's activation vector in the gradient term, and additive structure allows the LSTM to identify such a parameter update at each time step to prevent fast decaying behaviour. For these reasons, LSTM have become state of the art in terms of time series prediction and are used in a variety of contexts including voice assistants, instantaneous translators or weather prediction systems [6].

Finally, a more recent type of RNN has been developed which combines most the advantages of LSTMs while having fewer parameters and hence being much less computationally expensive. These are Gated Recurrent Units (GRU) [32]. The GRU introduces some elements of the LSTM with a forget gate but has fewer elements due to lacking the output gate. In most cases, GRU have shown lower to similar results as LSTM but can be practical due to being more efficient.

$$h^{(t)}_i = u^{(t-1)}_i h^{(t-1)}_i + (1 - u^{(t-1)}_i)\sigma\left(b_i + \sum_j \mathbf{U}_{i,j} x^{(t)}_j + \sum_j \mathbf{W}_{i,j} r^{(t-1)}_j h^{(t-1)}_j\right) \tag{22}$$

Equation 22 describes the update equation of the hidden state in a GRU. In this equation, $u^{(t)}$ is the update gate, which decides how much of the previous hidden state goes into the current hidden state. The second gate is $r^{(t)}$, the reset gate. The reset gate controls which parts of the current state gets used in the future state. They can be described using the following equations:

$$u_i^{(t)} = \sigma\left(b_i^u + \sum_j \mathbf{U}_{i,j}^u x_j^{(t)} + \sum_j \mathbf{W}_{i,j}^u h_j^{(t)}\right) \tag{23}$$

$$r_i^{(t)} = \sigma\left(b_i^r + \sum_j \mathbf{U}_{i,j}^r x_j^{(t)} + \sum_j \mathbf{W}_{i,j}^r h_j^{(t)}\right) \tag{24}$$

The reset gates control which parts of the state get used to compute the next target state, introducing an additional nonlinear effect in the relationship between past state and future state [32]. The main difference with the LSTM is that a single gating unit simultaneously controls the forgetting factor and the decision to update the state unit. A comparison of the model's performance using LSTMs and GRUs will be performed in order to determine which of the two is a better solution in this project's specific scope. Figure 21 is a schematic comparing the internal structure of RNN, LSTM and GRU to be able to compare the the three models and observe their differences.

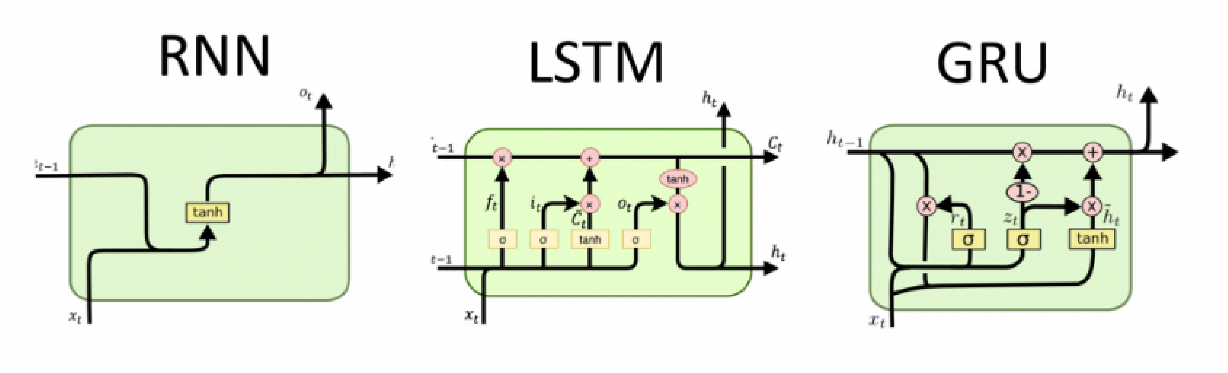


Figure 5: Schematic comparing RNN, LSTM and GRU models [71]

2.5 Sentiment analysis

The efficient market hypothesis states that stock prices already reflect all known information and are instantly adjusted in response to new information [24]. However, recent studies have rejected this hypothesis and showed the presence of "unexploited market opportunities for agents participating in the market" [39]. Over the last two decades, economists and computer scientists have shown that stock prices tend to under-react to news stories and company events and that investors "may be able to achieve abnormal returns by following signals conveyed in news articles" [26]. One area has attracted a lot of attention: NLP (natural language processing) and in particular sentiment analysis.

Also called opinion mining, sentiment analysis is a technique used to quantify and study the affective status of subjective information. The aim is to assign a score to text whether it is positive, negative or neutral. It has been widely used for customer reviews and social media posts about brands to understand customer satisfaction [1]. In the context of financial markets, prices are often directed by supply and demand and hence public opinion about a certain asset. More precisely, the emergence of social media at the beginning of the 21st century has transformed the access to information. Twitter is one the most popular social media platform for instantaneous exchange of information and a lot of financial opinions are expressed. Deep learning techniques have already shown "strong correlation exists between rise/fall in stock prices of a company to the public opinions or emotions about that company expressed on twitter through tweets" [58]. Professor Ronen Feldam wrote in a journal that the "utilisation of sentiment analysis techniques in stock picking can lead to superior returns" [25]. However, to be able to fully analyse the Tweets and predict prices using them, historical price data was necessary. Several machine learning models have been successful in analysing sentiments in a "regressive" way giving a number between 0 and 1 which is useful to capture nuances and polarity rather than simply classifying tweets as positive or negative.

In 2017, a group of researchers managed to predict stock price movements using only Twitter sentiment analysis as the only input [19]. They found an impressive 80% accuracy using simple machine learning models like support vector machine and linear regression in detecting the tweet's sentiment. The model was able to predict the price movement of the Apple stock with a mean average error of only 1.2. While these results appear very impressive and promising, there is one major flaw to consider which explains while their strategy cannot be used in practice. The experiment was done on a labelled dataset backtested over the past year. A labelled dataset means that the tweets are manually classified between positive and negative in order to train the model and perform a classification task between these two categories. Gathering tweets in real-time means that our data is unlabelled and cannot be treated in the same way. Our approach aims to not only be able to perform sentiment analysis on unlabelled data but also perform a regression method to capture nuances and polarity in the results.

Over the years, several deep learning models have succeeded each other in defining state-of-the-art in terms of sentiment analysis until a technological breakthrough by scientists at Google in 2018 [21]. Bidirectional Encoder Representations from Transformers, or BERT, is a transformer-based machine learning technique for natural language processing (NLP) pre-training developed by Google [3]. In its most basic form, a transformer is a deep learning attention mechanism composed a two parts. The first one is an encoder which reads the text input and the second one is a decoder that produces a prediction for the task. As opposed to more traditional directional models, which reads the text sequentially, BERT reads the entire sequence of words at once, enabling it to learn the context of the word on all of its surroundings. A 2020 literature survey concluded that "in a little over a year, BERT has become a ubiquitous baseline in NLP experiments" [64] with thousands of language detection software now using the model [3]. BERT is an unsupervised model pre-trained on unlabelled data containing over 3.5 billion words. However, the model was trained on a broad dataset and we expect our collected tweets to contain finance specific jargon and expressions. For this reason, we will be using FinBERT, a fine-tuned version of BERT designed with a deeper understanding of financial language and tailored for sentiment analysis [11]. This fine-tuned version was created by Prosus, the largest technology investment company in the world to achieve state-of-the-art performance in financial sentiment analysis. This model perfectly fits our task in determining the sentiment of finance related unlabelled tweets and we will use it for our sentiment analysis. Figure 6 shows the pre-training and tuning from the BERT white paper.

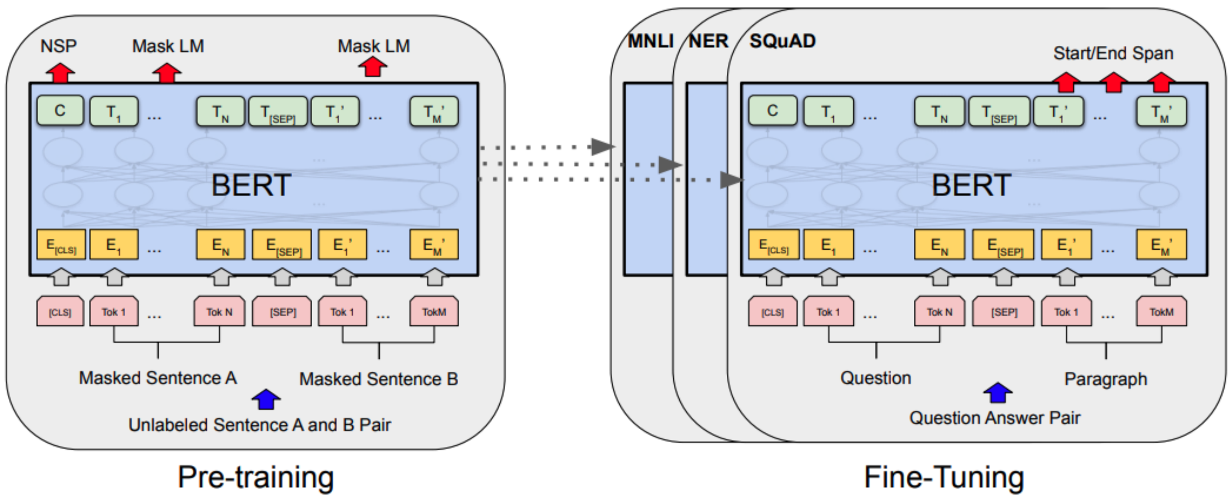


Figure 6: Overall pre-training and fine tuning procedure for BERT [21]

2.6 Cryptocurrency

Cryptocurrency is a virtual and digital currency secured by cryptography that makes it impossible to counterfeit or double spend [34]. Compared to traditional currencies, cryptocurrencies are not issued by a centralised authority or a central bank, but a set of decentralised procedures recorded in a blockchain, an extremely complex technology which aims to verify, approve and keep track of all transactions which makes impossible to hack or alter the information in the system [34]. The first of its kind, Bitcoin, was released in 2009 by Satoshi Nakamoto [53], in response to the global financial crisis caused by major American banks. Bitcoin aimed to create the first "peer to peer electronic cash system" [53], based on cryptographic proof instead of trust and would not require any third-party involvement. In recent years, cryptocurrencies have been seen as a new class of assets used for diversification purposes and have gained interest in terms of price prediction. In this section, we will briefly compare cryptocurrency markets to traditional financial markets like US equity and specify the interest in using sentiment analysis to predict the price of cryptocurrency. Traditionally in financial markets, brokers used to make money by charging large trading fees to customers. More recently, online brokers like Robinhood have popularised the concept of $0 commission fees. This policy, very popular among retail investors, has forced traditional brokers to find more ways of generating revenue. Brokers will often sell orders that don't specify an exchange in order to execute the trade on to market makers [29]. As a reward for directing the order to market makers, brokerage firms will receive a compensation called "payment for order flow". For zero commission brokerage firms, payment for order flow has become an important source of revenue to the point where brokers will usually route orders to a particular market maker for its own interest and not the investor's. Market makers provide liquidity to markets and make money on the buy-sell spread. Market makers are not very interested in institutional investors who have bargaining power over them and take advantage of retail investors to charge larger spreads, buying and selling stocks at prices differing their real value. This creates an issue for individual traders as trading with these market makers or high frequency firms on exchanges can severely limit their profit. Traditional financial markets are highly efficient, have low spreads on orders and news propagate quasi instantly.

In cryptocurrency markets, the biggest difference is that there are no longer brokers [29]. Instead of a centralised entity managing orders, investors handle the orders themselves in a peer-to-peer manner. Whereas in traditional markets, the settlement of buy and sell orders is handled by the National Securities Clearing Corporation, in cryptocurrency markets, they are handled by the exchange directly. If investors choose to use decentralised exchanges, the orders are handled by smart contracts, an electronic contract stored on the blockchain that run when pre-determined conditions are met. Unlike centralised exchanges where orders are

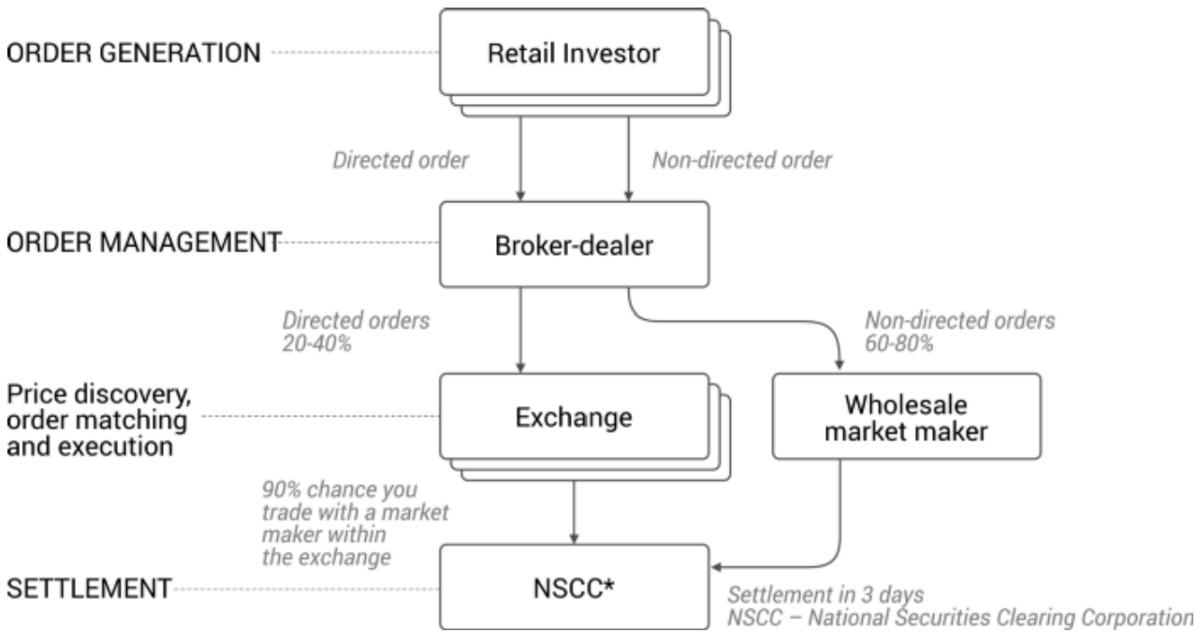


Figure 7: U.S. equities order flow [29]

paired by an order book, decentralised exchanges operate by matching individuals to exchange their assets directly. By removing third party involvement, high frequency trading firms lose their critical advantage between directed and non-directed orders. Decentralised exchanges can further impact high frequency traders as smart contracts usually have lower transaction speeds and higher fees. Due to these factors, it is often considered that cryptocurrency markets are far more inefficient, have wider spreads and news tend to propagate much slower. Although this may sound counter-intuitive, market inefficiencies create profit opportunities for investors. Established market such as the U.S. equities are often considered to follow a weak form efficiency [23], also called as the "random walk theory" [48]. This states that past prices, historical values and trends are all reflected in today's prices and cannot be used to predict future prices. This theory is generally accepted for equity markets due to their high liquidity, high number of participants and action and the overall market size. The American Securities Industry and Financial Markets Association estimates the total U.S. equity market capitalisation to be around $49 trillion [7]. This compares with a global cryptocurrency market of just over $1 trillion [4]. Although in practice most markets do show some inefficiencies, cryptocurrency's seem to be more pronounced due to its higher illiquidity, lower number of participants and slower news spreading speed which offer greater profit opportunities to traders and investors and support the idea that technical analysis can help predict future prices.

In addition, even the most established cryptocurrencies like Bitcoin are considered very volatile assets. A 2018 study by the American Institute of Economic Research has shown that Bitcoin's average daily percentage change and standard deviation are over six time greater than those of standard currencies [33]. A higher daily volatility increases the potential profits of an intraday trading strategy if correctly predicted.

Finally, contrarily to stocks and equities, cryptocurrencies are not backed by assets or cash flows but its prices are only due to speculation driven by investors' sentiments [65]. This is a key point motivating our research as sentiments have a much higher impact on cryptocurrency price changes than they do for other traditional asset classes.

The inefficiency of cryptocurrency markets coupled to its particularly high reactivity and sensitivity to investors' sentiments shifts opens greater opportunities for retail investors to make money. They also demonstrate that not only price changes can be predicted using technical analysis, but also that sentiment analysis must play a significant role in that prediction.

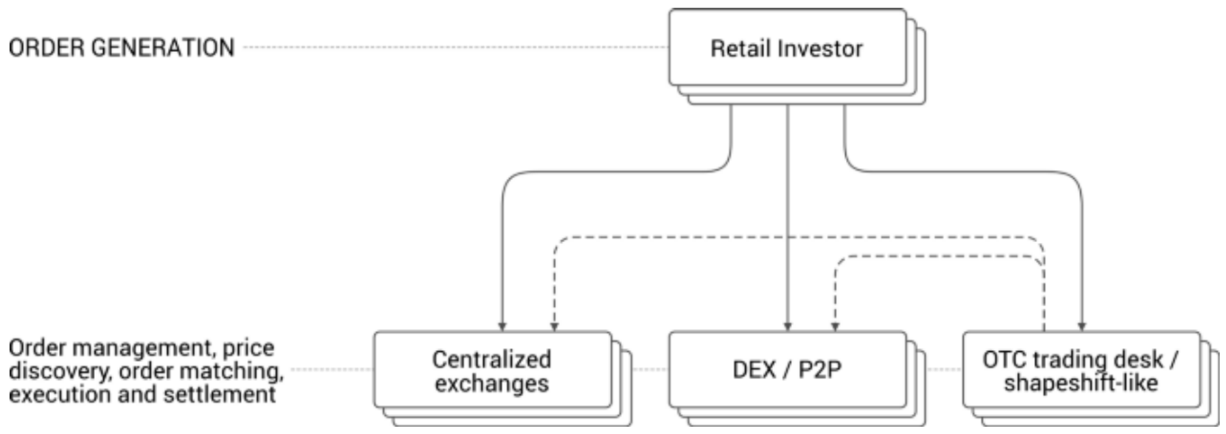

ORDER GENERATION
Retail Investor
Order management, price discovery, order matching, execution and settlement
Centralized exchanges
DEX / P2P
OTC trading desk / shapeshift-like

Figure 8: Cryptocurrency order flow [29]

2.7 Technical indicators

As opposed to fundamental analysis, trading and price predictive algorithms rely on technical analysis. Technical analysis uses indicators that mostly focus on price and volume actions to find a trend, a momentum or a pattern which could help a trading decision. The belief is that all relevant market information is reflected in the price and that the chart will mirror the "mood of the crowd" and perform an "analysis of human mass psychology" [16]. In this section, we will investigate some of the most used and famous technical indicators that we will use as features for our predictive model.

2.7.1 Relative Strength Index

The relative strength index (RSI) is a momentum indicator used in technical analysis to measure the magnitude of recent price changes to evaluate whether an asset have been overbought or oversold. The RSI is displayed as an oscillator and has values ranging between 0 and 100, which can interpreted as a percentage value. The most common interpretation of the indicator is that RSI values of 70 and above indicates that the security is becoming overbought or overvalued and might heading toward a trend reversal or a corrective pullback. On the other hand, RSI values of 30 and lower indicate that the security is oversold and might be undervalued. Equation 25 displays the calculation used for the RSI.

$$RSI = 100 - \frac{100}{1 - RS} \tag{25}$$

where

$$RS = \frac{\text{average gain over the last } x \text{ periods}}{\text{average loss over the last } x \text{ periods}} \tag{26}$$

It is most common to define the value of x to be 14. In this case, RS would be the ratio of the average gains over the last 14 periods by the average losses over the last 14 periods.

2.7.2 Stochastic Oscillator

The stochastic oscillator is another momentum indicator which compares the closing price of a security to a range of its prices over a certain period of time. Just like the RSI, the stochastic oscillator is commonly defined over the last 14 time periods. The belief is that price momentum occurs prior to the actual price action. The stochastic oscillator also defines overbought and oversold trading signals. Here, the overbought range happens for values of 80 and above, and the oversold range happens for values 20 or lower. Equation 27 indicates the formula for calculating the stochastic indicator.

$$\%K = \frac{C - L_{14}}{H_{14} - L_{14}} \times 100 \tag{27}$$

where

$$\begin{aligned}
C &= \text{current closing price} \\
L_{14} &= \text{lowest traded price over the last 14 periods} \\
H_{14} &= \text{the highest traded price over the last 14 periods} \\
\%K &= \text{the current value of the stochastic oscillator}
\end{aligned}$$

2.7.3 Williams Percentage Range

The Williams Percentage Range is another momentum indicator similar to the Stochastic Oscillator. Williams $\%R$ indicates the level of a price compared to the highest price over a certain time period, normalised by the high-low range over that same time period. Its values oscillates between 0 and -100, with a value of -20 and above indicating an overbought region, hence a sell signal, and values of -80 and below indicating an oversold region, hence a buy signal. Equation 28 displays the formula for the Williams Percentage Range.

$$\%R = -100 \times \frac{H_{14} - C}{H_{14} - L_{14}} \tag{28}$$

2.7.4 Moving Average Convergence Divergence

The moving average convergence divergence (MACD) is a trend following momentum indicator which tracks the relationship between two exponential moving averages of a security's price. The MACD is calculated by subtracting the 26-period exponential moving average (EMA) from the 12-period exponential moving average. Furthermore, the 9-period EMA of the MACD is calculated and serves as a signal line. A buying signal occurs when the MACD is crossing above the signal line and a selling signal is triggered when the MACD crosses below the signal line. Equation 29 displays the formula to calculate the MACD.

$$MACD = EMA_{12}(C) - EMA_{26}(C) \tag{29}$$

2.7.5 Price Rate of Change

The price rate of change (PROC) simply indicates the percentage change in price between the current price and the price a certain number of periods ago. Equation 30 shows the formula, with C_t being the price at time t and c_{t-n} being the security price at time $t - n$.

$$PROC_t = \frac{C_t - C_{t-n}}{C_{t-n}} \times 100 \tag{30}$$

2.7.6 On Balance Volume

Finally, the on balance volume (OBV) is a leading volume indicator which uses the cumulative traded volume of a security in order to predict the changes in its price. From a defined starting date, we iteratively follow equation 31 to update the value of the indicator. Essentially, we cumulatively add the volumes during periods where the price goes up and subtract the volumes on periods where the price goes down. If the price remains constant, then we do not update the OBV.

$$OBV(t) = \begin{cases} OBV(t-1) + Vol(t) & \text{if } C(t) > C(t-1) \\ OBV(t-1) & \text{if } C(t) = C(t-1) \\ OBV(t-1) - Vol(t) & \text{if } C(t) < C(t-1) \end{cases} \tag{31}$$

2.8 Model evaluation metrics

Although we argued in previous sections that traditional model evaluation or error metrics are not the best indicators to use when gauging of a trading model performances, we will still define them and use them to present the results. The reasons for that is that they are still useful to use alongside financial metrics to evaluate performances but they are also widely used in other papers. This means that they represent effective metrics to compare other state of the art methods and track performance improvements of our model to these methods. In this section, we will be defining the most commonly used error metrics used for model evaluation purposes.

2.8.1 Mean Squared Error

The mean squared error (MSE), is perhaps the simplest and most common loss function used to evaluate a model. The MSE is the squared difference between the predicted value and the true value averaged out across the whole model as outlined in equation 32. The MSE is easy to handle as it only has positive values. The main advantage of the MSE is that, since it squares the error, it is very sensitive to outliers. Putting larger weights on predictions with big errors, meaning far from the actual truth value, ensures that the model keeps the predictions rather close to the actual values. This also represents the main disadvantage of that loss function. If the model makes a single large bad prediction, the error is instantly magnifies resulting in a large error. However, in practice, it is often preferable to get a model that performs well the majority of the time even if it makes some bad predictions. Another issue to highlight is that due to squaring the error, the MSE ignores the direction of the error. This is critical is our case since the trading algorithm relies on the upward or downward direction of the price prediction in order to make a trading decision.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{32}$$

With N being the number of samples we are testing against.

2.8.2 Root Mean Squared Error

The root mean squared error (RMSE) is simple the square rooted MSE. It shares all the characteristics of the MSE however, since it is expressed in the same units as the original data. This makes it easier to compare to our values and can be easier to read when dealing with large error values.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{33}$$

2.8.3 Mean Average Error

The mean absolute error (MAE) presents similarity with the MSE but produces an opposite effects. The MAE is defined to be the absolute difference between the predicted value and the true value averaged out across the dataset. Since we are using the absolute value, all the errors will be weighted on the same linear scale. However, because we do not penalise large errors, we might end up with a model being right most of the time with some extremely large errors. The MAE can be more useful with data containing some outliers or unexpected values, and a dataset which does not contain too much noise.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{34}$$

2.8.4 Mean Average Percentage Error

Lastly, we define the mean average percentage error loss function (MAPE) which is a variation of the MAE except the difference between the predicted value and the true value is taken relative to the magnitude of the predicted value and expressed as a percentage. The advantage of the MAPE is the fact that, since it is a percentage, it is easily understandable and can be compared between different models to assess differences in performance.

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100 \tag{35}$$

Having defined all the main loss functions, let's discuss some drawbacks and the differences between them. Although the concept of the MAPE sounds very simple and convincing it contains severe drawbacks which makes it hard to use in practice. Firstly, we cannot have predicted values equal to 0 as the value of the MAPE would be undefined. This is not a problem to predict Bitcoin prices but will be when we try to estimate the changes in price instead of the prices themselves. Secondly, we can study what these loss functions try to minimise in practice. We will first study the MSE and RMSE in equation 36.

$$\frac{\partial MSE}{\partial y_i} = \frac{\partial \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\partial y_i} \tag{36}$$

$$\frac{2}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i) = 0 \tag{37}$$

$$\sum y_i = \sum \hat{y}_i \tag{38}$$

From these equations, we see that the MSE, and hence the RMSE, aim for the total forecast to be equal to the total demand [74] which means that these loss functions aim to produce a prediction that is correct on average. On the other hand, let's minimise the derivative of the MAE formula in equation 39.

$$\frac{\partial MAE}{\partial y_i} = \frac{\partial \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i|}{\partial y_i} \tag{39}$$

$$\text{with } \frac{\partial |y_i - \hat{y}_i|}{\partial y_i} = \begin{cases} 1 & \hat{y}_i < y_i \\ \text{indefinite} & \hat{y}_i = y_i \\ -1 & \hat{y}_i > y_i \end{cases} \tag{40}$$

$$\frac{\partial MAE}{\partial y_i} = \frac{1}{N}\sum \begin{cases} 1 & \hat{y}_i < y_i \\ -1 & \hat{y}_i > y_i \end{cases} \tag{41}$$

From these equations, we conclude that the forecast needs to be as many times higher than the true value as it is lower than this true value [74]. In other words, we are trying to split our dataset into two equal parts, hence minimising the median. On the other hand, since the MAE tries to minimise the median, it is very robust to outliers. Each metric has its own benefits and drawbacks and we will be using all of them as evaluation metrics. As far as the loss function is concerned, we will be using all of them to compare the performances.

2.8.5 Accuracy

We presented loss functions that will be used to train the model and quantify its regression performance. However, a lot of models only try to predict the direction of the price movement and hence tackle a classification problem. This will also be critical in our case when we incorporate the trading strategy which will be based on the movement of the price and not only its magnitude. In this case we might need other evaluation metrics used in classification problems. The first one is the accuracy. As described in equation 42, the accuracy is simply the ratio of correct predictions over the number of total predictions, or samples. This is a very straightforward metric that can be used to evaluate the overall performance. However, the main drawback of this metric is that it doesn't take the distribution of the samples in the dataset into account. This is especially critical since price data fatally include some kind of general upward or downward trend. For example, if our test set the price of Bitcoin goes from \$40,000 to \$30,000, by simply predicting a downward sloping line which only predict the price to go down, the model can expect very good accuracy since there is going to be more instances with the price going down than up. Hence, the accuracy does not necessarily gives a good idea of the model performance.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Number of total predictions}} \tag{42}$$

2.8.6 F1-Score

The F1-score aims to solve the issues encountered in the accuracy metric. It combines two metrics, the precision and the recall. The precision is the ratio of true positive predictions over the number of total positive predictions made.

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}}$$

The recall is the ratio of correct positive predictions over the number of all relevant samples, all the samples that should have been identified as positives.

$$\text{Recall} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}}$$

The F1-score is the combination of both described in equation 43. It describes both how precise the predictions are how many instances it classifies correctly, as well how robust the model is, it does not miss a significant number of instances. Whereas we try to minimise the previously presented loss functions, we always try to maximise the accuracy and the F1-score.

$$\text{F1-score} = 2 \times \frac{1}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} \tag{43}$$

2.9 Financial evaluation metrics

As previously discusses in the Section 2.1, traditional model evaluation metrics are often not the most effective way to evaluate a forecasting model involving a trading algorithm. In the Introduction, we presented a lagged input model which presents no predictive power but obtains remarkable results in terms of both MSE and accuracy. This "model" also achieves an F1-score of 78% for negative predictions and 61% for positive predictions. However, if we apply the trading algorithm, over the course of 2 days, we lose over 10% of our money using a lagged model. This means that we need to evaluate and select our model on financial based metrics.

2.9.1 Return on Investment

The easiest and most commonly used financial metric is the return on investment, the ROI. It evaluates the profitability of an investment to be able to compare it to other investments.

$$\text{ROI} = \frac{\text{Current value of the investment - Initial cost of the investment}}{\text{Initial cost of the investment}} \tag{44}$$

The ROI is a great and simple measure to help us instantly understand the profits made using our algorithm. However, the ROI does not give a full picture of the returns of an investment. Particularly in cyptocurrencies, a very high volatility is to be expected. In traditional finance, it is common practice to evaluate a return based on its risk, which is measured with the volatility. The return on investment metric completely fails to take the risk into account when measuring the returns of an investment.

2.9.2 Sharpe Ratio

The sharpe ratio was developed by Nobel laureate William F. Sharpe as a measure of the return of an investment compared to its risks. The ratio is the average return earned in excess of the risk-free rate per unit of volatility, or risk. The risk is defined to be the standard variation of the excess return.

$$\text{Sharpe ratio} = \frac{R_p - r_f}{\sigma_p} \tag{45}$$

$$\begin{aligned} \text{where } R_p &= \text{return of the portfolio} \\ r_f &= \text{risk-free rate} \\ \sigma_p &= \text{standard deviation of the portfolio's excess return} \end{aligned}$$

The sharpe ratio offers a better understanding how the returns relative to the risks. However, the main limitation comes from the fact that standard deviation of the returns in the denominator is used a proxy for the total risk and assumes a Gaussian distribution and no skewness.

In practice, financial returns are skewed away from the average due to the large number of surprising drops and spikes in prices. Because of this, the sharpe ratio tends to underestimate the risks associated with an investment and are influenced by the return of the underlying asset over a specific time period. Therefore it is impossible to use it to compare different algorithm's performances.

2.9.3 Value at Risk

"The objective of trading algorithms is to optimise the expected return of investments under the constraint of the risks generated by the investment" [70]. This is the aim of the value at risk (VAR) metric. It is used as an indicator to optimise the risk adjusted return of an algorithm. The metric by itself aims to quantify the potential risk, or losses, that can be engendered by an investment as a probability of the event occurring. VAR is very powerful as it takes into account the four moments of a distribution: the return, the volatility, the skewness and the kurtosis [70]. Although the VAR is a very robust metric, it is very hard to use in practice as it does not have an establish formula or protocol. The VAR can be defined using historical data, using a defined return distribution such as Gaussian or by implementing a Monte Carlo simulation to estimate all possible scenarios. In any case, it is difficult to apply to a model without prior assumptions and for these reason, we will not be using it as an evaluation metric.

$$\text{VAR ratio} = \frac{\text{Return}}{\text{Value at Risk}} \tag{46}$$

3 System Design

Figure 9 shows a diagram of the proposed framework for the system overall architecture. Although some modifications have been implemented in the implementation compared to the original design plan, the main building blocks have remained consistent in the final design. The system starts with a real time tweet extractor from the Twitter API and stored in a cloud based folder. Originally, this function was planned to be hosted on a virtual machine with the tweets stored in a database. However, this proved to be challenging to implement both due to time constraints and restrictive access to the necessary tools. The tweets are then cleaned and processed locally before being fed into a sentiment analyser. From there, multiple features are created and a general sentiment data file is created. Along with the corresponding price data, we proceed to feature engineering to generate new features that will be used by the price forecasting model. We then proceed to the most critical part of the system which is the model definition and training. As stated in Section 2.2, we differentiate ourselves from other state of the art methods by building a complete system from data collection and sentiment analysis to model definition and training to a complete trading algorithm relying on financial evaluation metrics in order to assess the model's performances rather than only traditional error metrics. Therefore, we define a trading algorithm and a set of financial metrics to judge the model's performance. We then use these results to evaluate the model as well as select and finetune some model parameters. We use these observations to refine the system and obtain improved results. In section 5.3 we will study the generalisation power of the model on unseen data to ensure that we are not overfitting on testing data either. When the parameters make theoretical sense and maximise the potential of the system, we present the results. These results will present the performances of the system as well as feature importance in order to assess whether the general thesis of the project is confirmed or not: do sentiment analysis help price forecasting and profit maximisation in cryptocurrency? Finally, we will be comparing our system performances to state of the art methods to quantify improvements and discuss further work considerations.

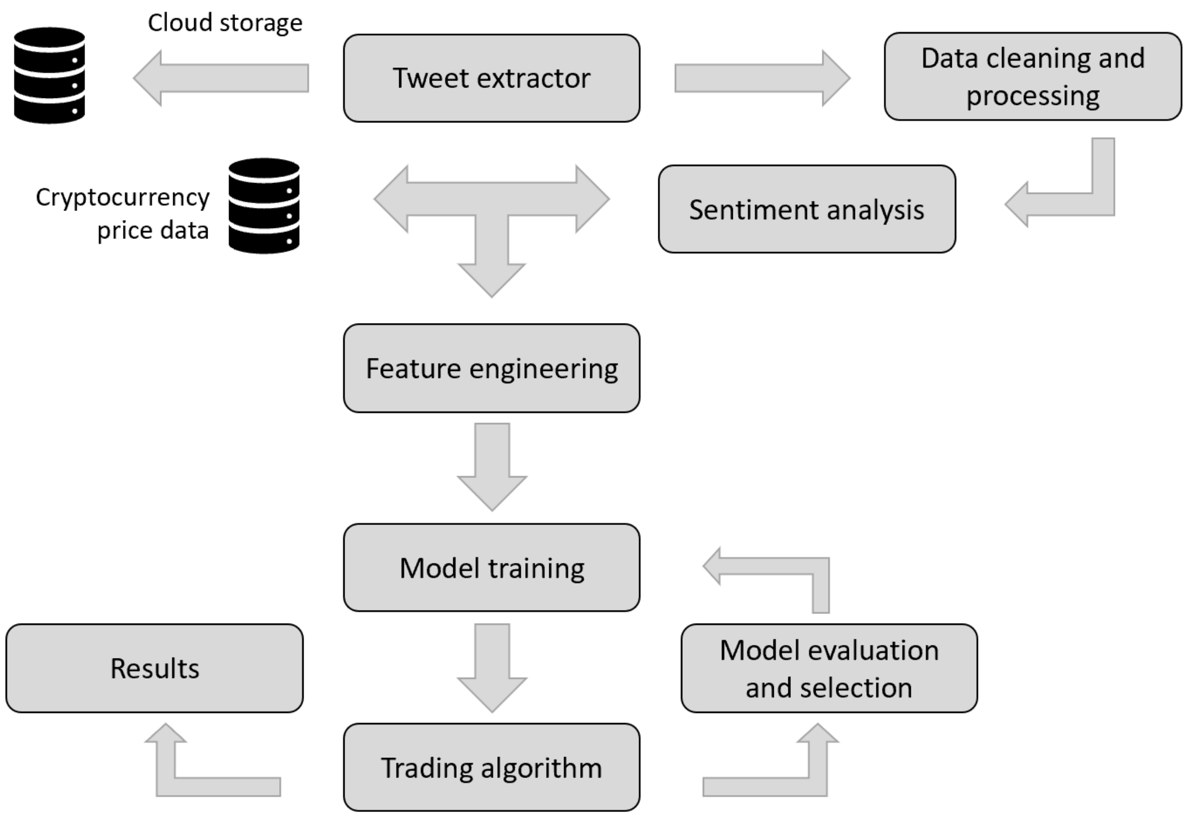


Figure 9: System architecture diagram

4 Implementation

In this section, we will discuss the implementation of the different modules used in the system used to predict cryptocurrency price movements and design a profitable trading algorithm. The first part will cover the data collection.

4.1 Tweet extraction

In order to perform sentiment analysis, we first need to extract and gather text data. Twitter is a platform of choice for individual investors to express their opinions and it offers several different ways to collect data. Since the start of the project, several limitations were encountered in terms of the quantity and quality of the collected data. The first limitation is that Twitter does not allow to collect historical data prior to the past week. This service is only available to professional working with the social media company. Although there exists alternative services and Python libraries capable of bypassing these limitations, these methods are strictly forbidden by Twitter's terms and conditions and therefore we will not make use of them. The chosen solution relies on Twitter's API service named Tweepy. This service offers two restrictions in terms of the data we can collect. The first one concerns the quantity of data collected. As part of this project, we applied for an elevated developer account which limits the number of Tweet requests to 2 million per month. Since we aim to request tweets every 15 min, this equals to just under 700 tweets/ 15 min. We maximise API's limits by requesting 300 tweets every 15 minutes for Bitcoin (BTC) and Ethereum (ETH), adding up to 600 requests per 15 min or 1.8 million requests per months for over 3 months. The second limitation comes from the fact that elevated developer accounts cannot request tweets filtering by cashtag. Cashtags, denoted by the dollar symbol $, is a symbol used on Twitter to talk about a particular ticker symbol. A ticker symbol is an abbreviation used to identify a particular security. It is most often used in the stock market but is also used in cryptocurrencies. The ticker symbol of Bitcoin is $BTC, and Ethereum's is $ETH. Cashtags make it easier to track when someone is talking about a certain cryptocurrency and often contain more relevant posts. However, Twitter does not allow to gather tweets including cashtags and we hence decided to use hashtags which includes either the ticker symbol, #BTC, or the name of the cryptocurrency, #Bitcoin, to collect our tweets. Table 1 gives an example of the data collected when requesting the Twitter API.

We use the recent tweet endpoint from the Twitter API 1.1 to search the most recent tweets. Some pre-filtering settings are applied in order to keep more relevant tweets and avoid duplicates or spam tweets. We do not allow tweets replies or retweets in our selection as well as filter tweets to be in English only. This choice is made since the BERT sentiment analyser is optimised to work best in the English language. Lastly, although the twitter API returns a lot of data, we only keep the categories presented in table 1 as they are the most relevant in our case.

Date	Tweet Text	Retweet Count	Favourite Count	Username	Follower Count
Sat Mar 26 01:23:05 2022	. Rio de Janeiro will allow residents to pay property taxes in #Bitcoin #cryptocurrency beginning in 2023.	19	49	Ape_Apecoin	9993

Table 1: Example of Twitter data collected via API request.

After retrieving the data in json format, we create a dataframe table with all the tweets and their associated details and create a file containing the day, the time and the name of the cryptocurrency. As mentioned previously, the original aim was to run the script on a virtual machine and store the file in a database. However this method proved to be more complicated than first anticipated and we had to found a different solution. The method used was to run the Python script on a Raspberry Pi 3b computer running 24/7 that would locally save the file and send it to an address email created for this project. Using Microsoft's automation tools, we can link the email address to a OneDrive folder. Each time an email containing an attachment is received, the file is then saved into the OneDrive folder and available in the cloud. We can set up the Linux tool CRON to run the python script every 15 minutes and make requests to the Python API. Although not the most efficient way, this method proved to be reliable and worked fine for our purpose.

4.2 Data cleaning & processing

The collected Twitter data contained a lot of noise and had to be cleaned and pre-processed before proceeding to sentiment analysis. First of all, a drawback of our tweet extraction system is the fact that we request the last 300 tweets regarding a certain cryprtocurency every 15 minutes without being to check that the tweets were actually published in the last 15 minutes or not. The first step is hence to filter the tweets in our dataframe to only contain tweets published in the 15 minutes prior to the request. Secondly we perform some manipulations to clean the text data of the tweets. Namely, we convert the text to lower case, remove any account names mentioned with @, remove any link or hashtags and replace any contractions with their longer form. For example, we transform ”can't” into ”cannot” or ”what've” into ”what have”. Then, we remove special characters as well as unwanted punctuation that comes from the json format and trailing spaces. These operations allow to get English text in the same form as the one that the BERT model has been trained on as well as eliminate small variations in text that can be used to create spam tweets. We then eliminate duplicates and create a file containing our cleaned text ready to be used in the sentiment analyser.

4.3 Sentiment analysis

As mentioned previously in the introduction, we use a version of BERT fine-tuned on financial data to get more precise results that would be able to understand financial jargon. There is not a version of BERT fine-tuned for cryptocurrency data or social media posts and after attempting to fine-tune the model on our collected data, no significant result improvements were observed.

Before using the model, the first step is to tokenize our inputs. Each transformer model has its own specific tokenizer which prepares the inputs for a model. This step is essential and is composed of multiple transformations. First, the text is tokenized, that is each string is split into sub-word token string. Then, each token string is encoded into integer ids in a format specific to the model at use. Finally, the tokenizer also manages special mask tokens by adding them, assigning them to attributes in the tokenizer for easy access and making sure they are not split during tokenization [2]. The way BERT works is that 15% of the words in a given inputs are replaced by a [MASK] token. Based on the context of the words surrounding it, the model then attempts to predict the original value of the masked words and calculate a probability for each prediction. This transformation gives us a BatchEncoding that we can fed into the classification model. The first limitation encountered when using the FinBERT sentiment analyser was a hardware constraint when inputting all the text. The model required a lot of memory usage which meant that we had to input the tweets individually and not together as a general text. This did not have significant impact on the result of the prediction scores. We then use a softmax function to transform our model output into a tensor in order to decode the prediction results. From that tensor we can obtain the positive, neutral and negative score for each tweet. We define the sentiment score as the difference between the positive score of the tweet minus the negative score of the tweet which lies between 1 and -1.

4.4 Feature engineering

This section will cover the feature engineering side of the project and the reasoning between them. We can start with the sentiment features. From sentiment scores, many attempts and discussions were made in order to determine the most significant features and quantify the sentiment analysis in the best possible way. In the end, six sentiment related features were created. The first is simply the *Number of Tweets* during the collection period. Although this figure has some limitations, such as the fact we collect only 300 tweets per 15 minutes and hence cannot account for the fact that there might have been more tweets posted during that time period, it gives a general idea into the number of tweets posted in the last 15 minutes. The second feature is the *Additive Score.* This figure aims to give a general score to the data in a file by adding 1 to the score per positive tweet and subtracting 1 to the score per negative tweet. The third and fourth features are the *Mean Score* and the *Standard deviation Score.* These two features are some of the most important among the sentiment analysis features. Intuitively, the mean score gives us the average sentiment from the tweets posted in the last 15 minutes while the standard deviation gives the polarity of these sentiments. Adding the polarity is a major differentiating factor and is critical to our model. Most previous works have failed to include that metric into their analysis while it may highlight indecisive sentiment in the market which is often preceding major breakouts. The last features are the *Number of positive* tweets and the *Percentage of positive.* Although these two features may seem redundant, they are not since the volume of tweets varies for each time period. The features are then stored a DataFrame in a local csv file. Table 2 displays a sample of the features collected for each file.

Date	Time	Number of tweets	Additive score	Mean score	Standard deviation score	Number of positive	Percentage of positive
28/03/2022	06:45	306	67	0.19	0.31	107	0.35
28/03/2022	07:00	296	55	0.19	0.29	81	0.27
28/03/2022	07:15	277	48	0.18	0.32	75	0.27

Table 2: Sentiment features sample.

Before continuing with the feature engineering process, we need to perform some data processing beforehand. The operations are relatively straight forward and allows us to merge the sentiment data with the collected price data. In order to collect historical price data, we choose to use Binance. Binance is the largest cryptocurrency exchange in the world, both in terms of daily trading volume and in terms of users [4]. The platform offers very competitive fees at only 0.1% per trade as well as two main sources of information to extract data. Binance offers both an API and a websocket in order to extract historical and real-time data from all the currency pairs available on the platform. For this project, we choose to use the API library optimised for Python as it is very easy to use for historical data. It is worth noting that the websocket offers more robust and reliable live stream data feeds which should be considered if the system is used on real-time data for live trading. We extract five features. Namely:

- Open: this the price at which the currency opens at the start of the time period.
- High: this is the highest price that the currency reaches during the time period.
- Low: this is the lowest price that the currency reaches during the time period.
- Close: this the price at which the currency closes at the end of the time period.
- Volume: this is the cumulative amount of tokens exchanged during the time period.

The sentiment data and price data have different time format by default, which we resolve in order to make sure we can merge the two files on the basis of that time column. We then merge the data frames left on the time column and sort the values. Finally, we notice that the sentiment data has some missing values. This is common in real-life applications to have missing or incorrect data. Because, in our case, the number of missing values is very restricted, we decide to interpolate through padding instead of using a linear or polynomial function. Padding simply assigns to the missing value the same value present above them in the dataset. This is acceptable since we do not have consecutive missing values. Other techniques can include defining a function, linear or polynomial, which estimates the missing values by defining a function using the previous and following values to estimate the missing value.

Once we have a complete dataset containing the price data and the sentiment data, we can perform the last part of the feature engineering process which includes computing the technical indicators presented in the Background section 2. To reiterate and add on to the list:

- Relative Strengh Index (RSI),
- Stochastic Oscillatore %K,
- Williams Percentage Range %R,
- Moving Average Convergence Divergence (MACD),
- Price Rate of Change (PROC),
- On Balance Volume (OBV),

- High Low Difference, which is the percentage difference between the High and Low prices and can be defined as a volatility indicator.

Table 3 gives an example from the financial features created in the dataset after the first 20 time periods. We note that the MACD indicator only starts having values after the 24^{th} value as its formula include the 24-period EMA. This means that there needs to be at least 24 values to be able to evaluate the MACD.

After dropping the rows with NaN values (-), we finally obtain a dataframe representing all the features for our selected cryptocurrency, in this case Bitcoin, for the time-frame of the analysis. The final dataframe is of shape (9600×19). We note that we have a column for the timestamp in addition of the features. The number of rows is 9600, which represents samples taken every 15 minutes for over 100 days.

Close	Volume	HighLow	Change	RSI	MACD	Stochastic	Williams	OBV
43492.96	224.07	47.05	0.45	57.95	-	24.71	-13.29	1125.31
43468.89	405.76	24.21	-0.06	56.79	-	66.14	-17.90	719.56
43506.49	1136.92	93.70	0.09	58.19	-	29.67	-31.07	1856.48
43563.59	488.21	47.53	0.13	60.30	28.59	50.67	-22.63	2344.69
43492.24	456.29	55.76	-0.16	56.47	23.95	50.62	-33.18	1888.40
43367.30	579.98	34.34	-0.29	50.42	11.42	39.11	-51.64	1308.43
43348.49	356.41	35.44	-0.04	49.56	1.35	28.88	-54.42	952.02
43234.30	636.00	48.76	-0.26	44.58	-12.74	-97.79	-83.38	316.02
43258.34	260.32	28.92	0.06	45.82	-19.76	-55.89	-79.23	576.34
43065.88	977.22	116.59	-0.44	38.44	-35.75	-196.11	-71.78	-400.88

Table 3: Features dataset sample.

To summarise Section 4.4, Table 4 lists all the input features used for the model's training and testing.

Data name	**Category**
Open	Price feature
High	Price feature
Low	Price feature
Close	Target feature
Volume	Price feature
Number of tweets	Sentiment feature
Additive score	Sentiment feature
Mean score	Sentiment feature
Standard deviation score	Sentiment feature
Number of positive	Sentiment feature
Percentage of positive	Sentiment feature
High-low percentage difference	Technical indicator
Percentage change	Technical indicator
RSI	Technical indicator
MACD	Technical indicator
Stochastic Oscillator	Technical indicator
Williams	Technical indicator
On balance volume	Technical indicator

Table 4: Summary of the model's input features used for training and testing.

4.5 Neural network

With all features created and a clean dataset, we now proceed to to the model training part. The first step is to normalise the data using a *MinMaxScaler*. Since all the data does not have the same range, some variables can have a bigger weight or influence in the model performance in one direction or the other. We use a scaler to transform all data into the same range to make it easier for the model to improve convergence for gradient descent in the learning process. Before proceeding, there is one critical step that needs to be performed: data separation. A common mistake observed in time series forecasting in to scale the entire dataset all together and then proceed to split the data and train the model. This leads to abnormal performances and results which can not be replicated in practice. The reason for this is called *look-ahead bias*, or data leakage. A MinMaxScaler transforms input features to scale them into a given range, which is usually 0 and 1. It uses the maximum input value as the upper range and the lowest as the lower range and scale all the values in between accordingly. If we normalise the data using a scaler on the whole dataset, then potentially the minimum or maximum values will belong to the test dataset and will be used to scale the values in the training dataset. This means that future information is introduced into the training data which leads to overconfidence in models built out of skewed results. Therefore we need to split the data into different sets before performing normalisation. We split the data into three datasets: the training set which will be used to train the model, the test set which will be used to test the model with new unseen data to simulate real life application of the model and represents 30% of the total dataset and finally the validation set. The validation set represents 10% of the non-test set and is used to evaluate the performance of the model while training by applying the model without updating the weights. This set is used to give an idea of the generalisation capability of the model and set early stopping to not overfit the training data. The validation and test sets are used to evaluate the generalisations capabilities of the model on new unseen data and therefore we scale those data using the training scaler and not define new scalers. Figure 10 shows the data distribution across the dataset. The training dataset contains 5218 rows, the valdiation set 1305 rows and the test dataset 2796. Due to the abundance of data, we reserve bigger portions for validation and testing than usual to be able to evaluate the model in more detail.

With data correctly separated, we now format the sets into formats suitable for neural networks inputs. The first step is to transform our tables into time series dataset. That function decomposes the dataset into subsequences of the full time series with each subsequence consisting of encoder and decoder timepoints for the given time series. It is a sophisticated PyTorch class which separates the x previous values from the x^{th} value, which is the one the model is trying to predict with x being the determined sequence length. We then use the DataLoader class which decomposes the dataset into batches. The batch size is a hyperparameter used to control the gradient descent. Instead of going through the entire dataset at once and cal-

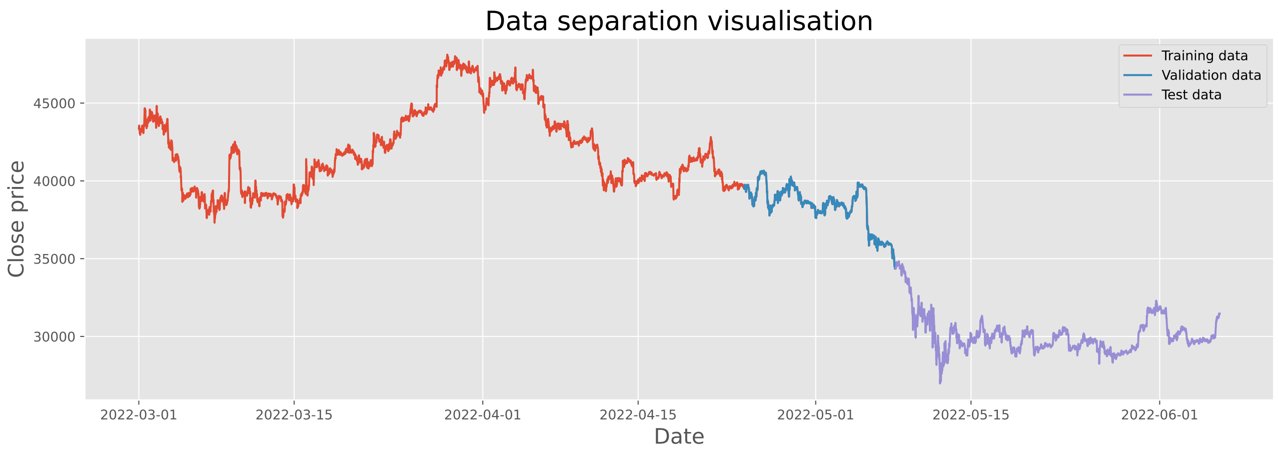


Figure 10: Train test split data separation graph.

culating the loss, we go through a smaller selected portion of the dataset, known as a batch, before updating the model's weights. This is a necessity in terms of memory usage and speed of convergence for any deep learning model. The disadvantage of using batches is that the model might be converge to a sub-optimal solution due to the noise introduced by only using a portion of the dataset. Hence, the batch size will be a hyperparameter to define in order to find a balance between speed of convergence and optimisation. Although other transformations are necessary during the training process, it is irrelevant to cite them all in this report.

We can now define and train our neural network. The model used contains 6 tunable hyper-parameters. These are parameters whose values control the learning process [55] and need to be tuned depending on the model and the task at hand. We will explain the choices of these parameters in section 5.1 but for now we just state them:

- Sequence length: 24
- Batch size: 32
- Hidden size: 32
- Number of layers: 2
- Dropout rate: 0.1
- Learning rate: 0.001

The sequence length corresponds to how many previous values should the model use to make the next prediction. In our case, it is equal to 24 which means the model uses the data from the past 6 hours to make a prediction for the price in 15 minutes. We decompose the dataset in batches of size 32 each which leads to a total of 162 batches. The hidden size corresponds to the number of features in the model's hidden state h. We choose to use a GRU in our model with

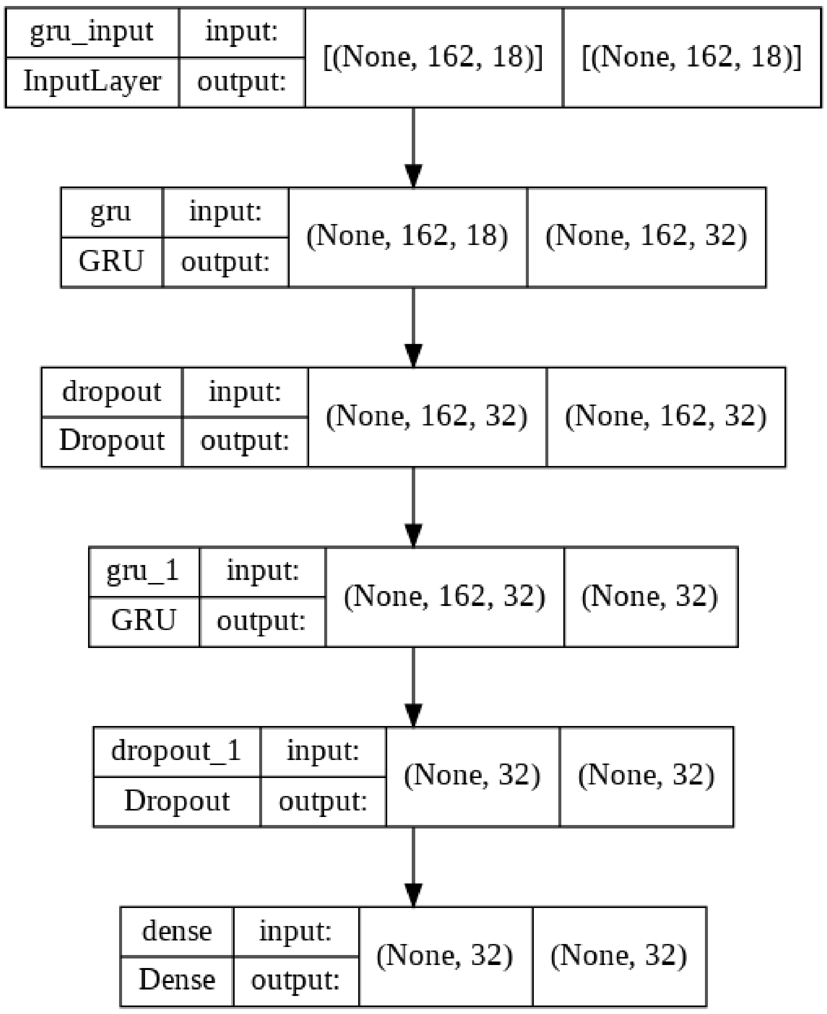


Figure 11: Model architecture plot.

two recurrent layers which creates a stacked GRU network. The second GRU takes as input the outputs of the first GRU in order to compute the final result. Stacking multiple layers on top of each other increases the depth and the complexity of the network. Benefiting from the GRU's memory capabilities, adding a second layer usually leads to much better performance as the is capable of more predictive power and achieves better pattern recognition. We define the dropout rate to be 0.1. Dropout is a technique used to prevent model overfitting which consists at randomly disabling some neurons and their corresponding connections at random in order to avoid the model relying too much on single neurons which forces all neurons to learn to generalise better. A dropout rate of 0.1 means that each epoch, the network will ignore 10% of the neurons. Finally the learning rate is set at 0.001 which corresponds to the step size of gradient descent at each iteration. As outlined in Figure 11, the model is therefore composed of 5 layers with the last one being a linear layer which converts the output into a single value. This is used to compute the loss function which was chosen to be the Mean Squared Error loss function presented in the Background section 2. The optimiser used is the Adam optimiser which is a very common and praised in the machine learning community.

As mentioned previously we perform early stopping on the model. Early stopping is another technique used to prevent overfitting of the model. Overfitting is often characterised by a model with a training loss that keeps improving but a validation loss that increases. This happens when the model does not learn important features anymore but only the training data noise and is not able to perform accurately on new unseen data. We locally store the model weights only when the validation accuracy has improved from the previous epoch. We set early stopping at 50 epochs, which means that if, after 50 consecutive epochs or iterations, the validation loss has not improved we stop the model training. This helps prevent overfitting and reduces the training time to only learn meaningful features.

Finally the last step involving the neural network is the prediction part. Similarly to the transformations performed on the training data before feeding the network, we transform the test data in the same way. Given the features in the test set, we make a prediction for the target label using the weights of the last saved model ensuring that the gradient is not updated in the process. The last step to be able to visualise the predictions against the actual values is to descale them. We load the scaler used in the pre-processing part of the implementation and use an inverse transformation to descale our data back to the the original range. We can then plot the predicted data and the labels and visualise the results.

4.6 Trading algorithm

As mentioned in the background section we will use the common error metrics to compare our model to previous works and state of the art methods. However the aim of a price prediction model if more often to profitably trade an asset rather than minimise an error function. For this reason, we develop a simple trading algorithm in order to evaluate the model in real life trading conditions and evaluate the financial performance of the model. We will rely mostly on three financial metrics which are the return on investment, the sharpe ratio and the prediction accuracy. For the latter we simply have to create a binary array which takes 0 when the price goes down and 1 when the price goes up. We create two arrays, one for the actual realised movement of the currency and one with the prediction values. We can then compare the two arrays and find F1-scores for each label as well as the overall accuracy of the model. We highlighted in the 2 background section that the accuracy can be a misleading indicator, however in our case since the data labels are well balanced in the set we will rely on it to evaluate performance of the model. The other two metrics are the returns on investment and the sharpe ratio. To be able to define these, we need to define a buy and sell strategy based on our prediction. We define a function which simulates a portfolio and the actions of buying and selling a cryptocurrency based on the predictive movement. Taking the predictions array chronologically, if we think Bitcoin is going to go up in the next period we buy it. We include a threshold hyperparameter to trigger a trading decision only if the predicted movement surpasses that threshold. This can be used to filter out smaller movements which can represent noise. Finally, to make the the trading algorithm more realistic, we also include fees in the calculations of the trade algorithm. Specifically, we apply a 0.1% fee when we buy or sell Bitcoin which will affect the performance but make the algorithm more robust.

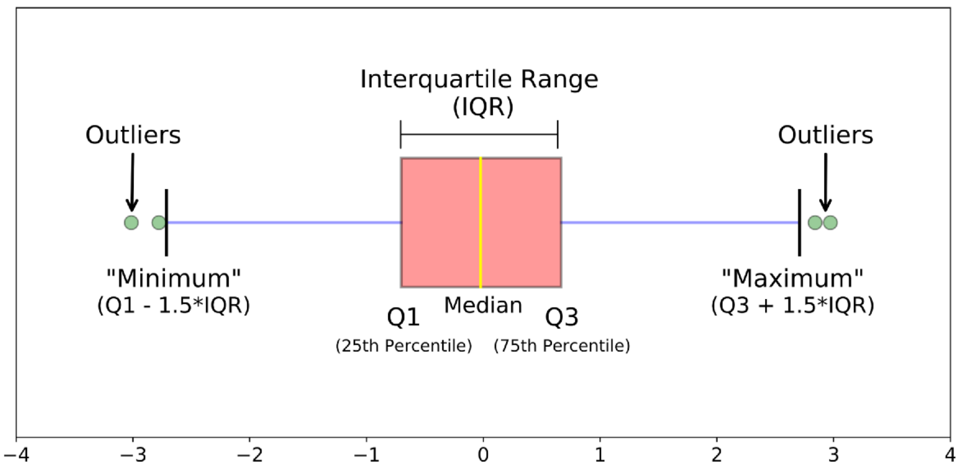


Figure 12: Different parts of a boxplot. [27]

In order to compare performances of different hyperparameters and the importance of sentiment features, we will use boxplot plots. Boxplots are graphical tools used to represent locality, spread and skewness groups of numerical data through their quartiles. As seen in Figure 12, a boxplot indicates the median of a series as well as a box delimited by the interquartile range and tails ending with the maximums and minimums. Dots on either side of the box represent outliers. We will use them to be able to compare the average performance and compare the different models.

4.7 Feature Importance

Neural networks are essentially black boxes making it really hard to determine the importance of a feature in the final model prediction. In order to determine the effect of the sentiment analysis features, we will implement the DeepLIFT algorithm [44]. This method approximates the SHAP values known from classic game theory in order to explain the output of a machine learning algorithm. The general idea is to evaluate how much a feature contributes to a model's prediction when it is included compared to when it is not thus deriving its importance in the final prediction. More specifically, the method decomposes the output prediction of a neural network on a specific input by backpropagating the contributions of all neurons in the network to every feature of the input. It then compares the activation of each neuron to its reference activation and assigns contribution scores, or multipliers, according to the difference. The reference input represents some default or neutral input that is chosen depending on the domain, e.g., zeros or feature averages. Once the multipliers have been computed based on a representative dataset, called background dataset, we can calculate the contributions of the input features to the model's output based on some sample inputs and rank the features in order of their largest contributions to get their importance [35] [67]. We then obtain a graph with the importance of each feature at at each time period prior to making the prediction. Using this method, we get more insight into the contributing factors of the model as well as detailed features interpretability.

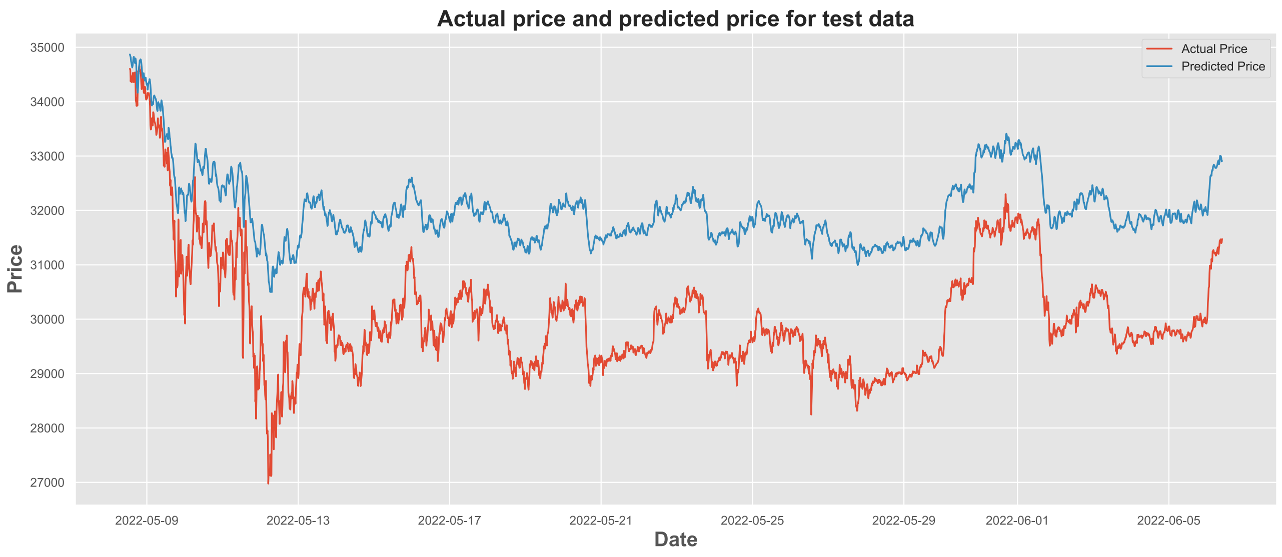


Figure 13: Predicted price against real price for test data.

5 Results

5.1 Hyperparameters selection

In this section, we will briefly cover the model hyperparameters selection and model choices. We recall that there are a total of seven parameters or architectural choices to make:

- Sequence length: this defines the number of previous inputs that the model should use in order to make a prediction for the next value. In other words how far back does the model look for forecasting.
- Batch size: the number of inputs to include in each set ”slice” given to the model. This method allows for higher gradient convergence speed.
- Hidden: the number hidden neurons between the neural networks layers.
- Number of layers: how many layers to include in the neural network.
- Dropout rate: the rate of neurons and their connections to ignore in each epoch. This is used to prevent model overfitting.
- Loss function: we presented multiple loss in the background sections, and will decide which one to use.
- Choice of model: we will need to determine whether to use LSTM, bi-LSTM or GRU.

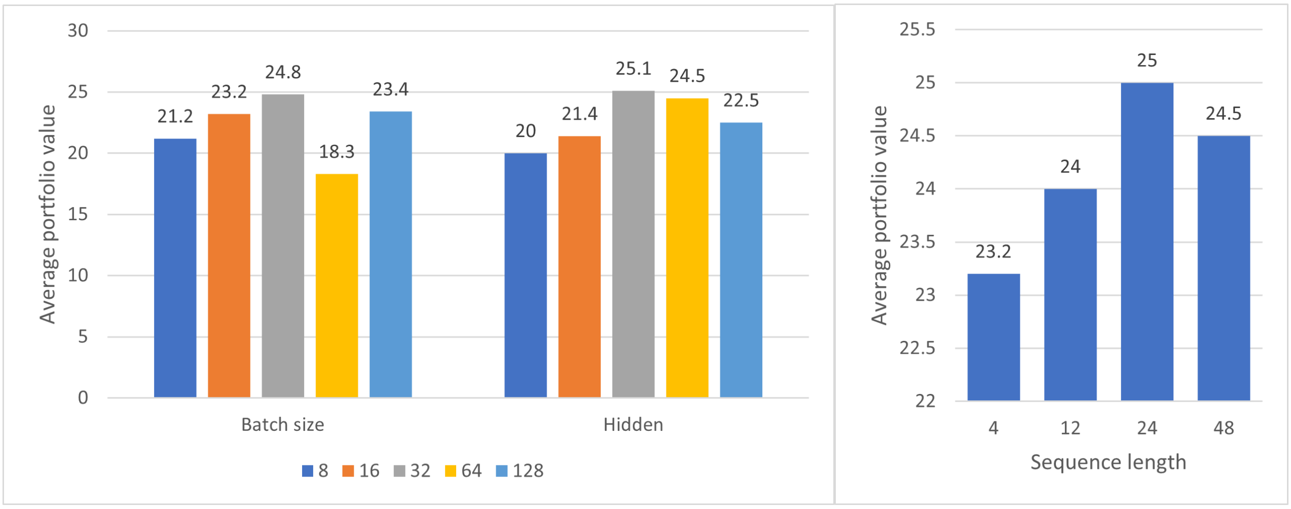
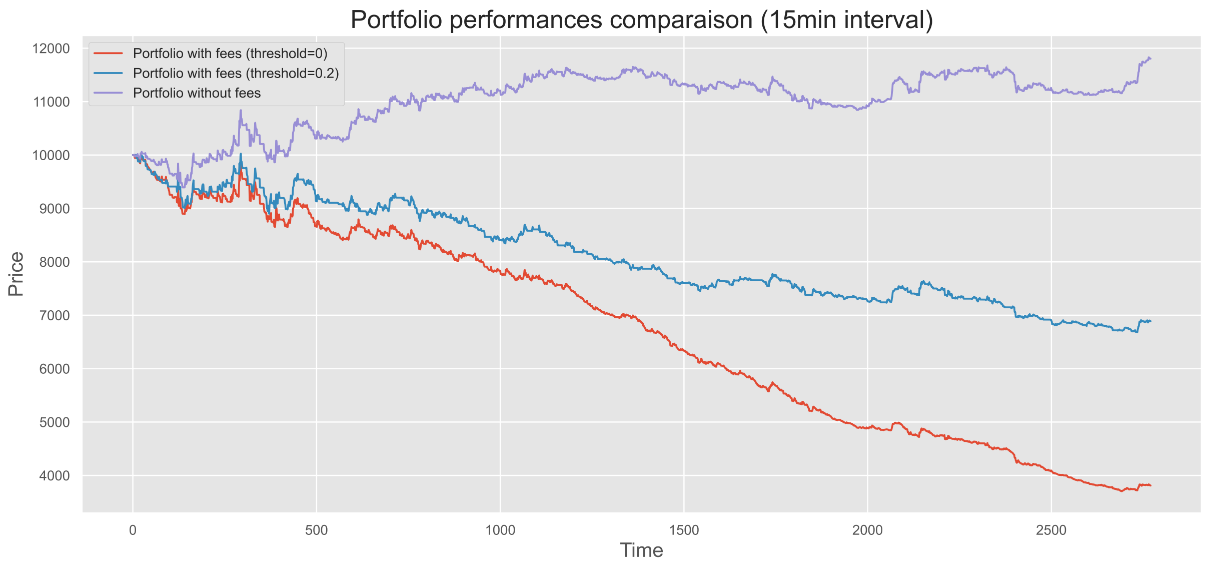


Figure 14: ROI for different values of selected hyperparameters.

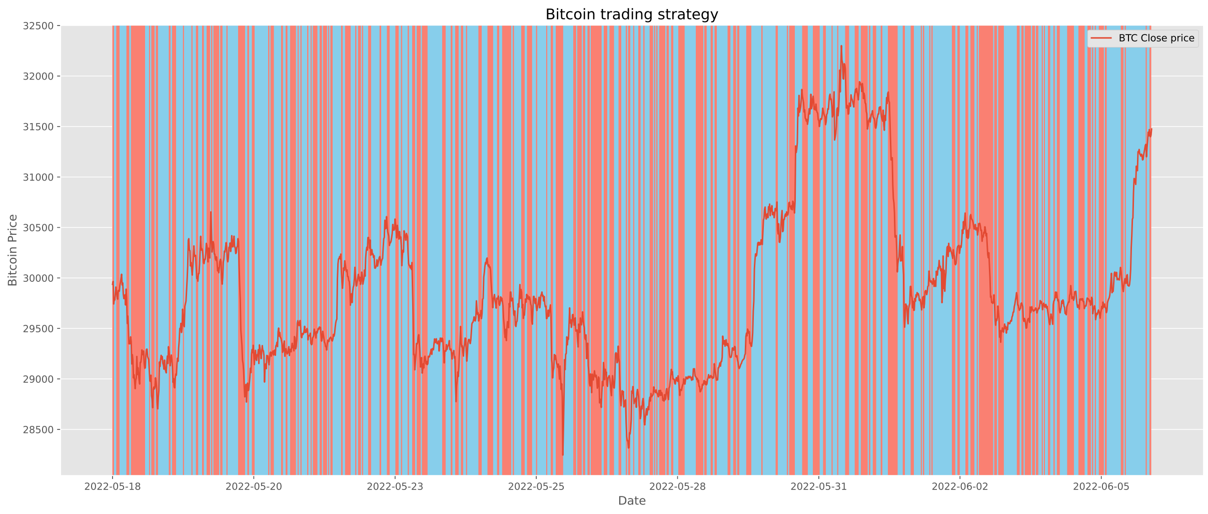
Grid search tests were conducted in order to fine tune all the hyperparameters. This is a technique which tests all the parameters for a given function and select the ones giving the most optimal output. In our case, we ran the trading function over 50 times for each parameters and took the ones with the best average return on investment. Figure 14 displays the average portfolio value using different hyperparameters at the end of the testing period with a starting capital of \$10,000. To determine the sequence length, it was found that going too far back was not improving the performance. This makes sense since data from one day ago may not be relevant to determine the price in the next 15 minutes. On the other hand, not going back far enough means that the model may not have enough information to determine the price in the next time period. It was found that $sequence\_length = 24$, which means using the last 6 hours, was the most optimal parameter with an average return on investment of 150% over the testing period. As discussed in section 4.5, the batch size parameter represents the size of the segments that we decompose the whole dataset in. A smaller batch size will improve computational efficiency but can result in sub optimal gradient minimisation solutions. The optimal batch size in terms of return on investment is found to be 32. Similarly the number of hidden neurons determine the model's complexity and depth of the hidden layers. The grid search tests found an optimal value of hidden to be 32 as well. The number of layers in the model also had to be determine. For equivalent performances, we prefer the smallest value as this improves computational efficiency. It was found that using only 1 layer does not use the full memory capabilities of neural networks while having 3 layers made the network overfit the data and lead to poor generalisation. The optimal number of layers was hence found to be 2. Changing the dropout rate did not lead to significant performance changes but a dropout rate of 0.1 was found to be optimal. This means that for each epoch 10% of the neurons and their connections are ignored in order to prevent overfitting and allow for a better model generalisation. Many loss functions were tried out even if not cited in this report including a MASE loss [36] or a Huber loss, which switches from a MAE to a MSE when the validation loss crosses down a pre-determined threshold. However, even for the drawbacks presented in the background section, the mean squared error loss function produced the best results. Finally it was found that LSTMs and GRUs produced very similar results in the task at hand. However, GRUs were found to be twice faster to train and hence were chosen for the final model.

5.2 Model performance

This section will present some results regarding the model performances as well as methods to improve them. We will evaluate the model to other approaches and state of the art methods in section 6.1. This section will use a Buy & hold portfolio with a starting capital of \$10,000 as a benchmark to compare the model performances. The test data lasts from the 08/05/2022 to the 06/06/2022. In this time frame, the price of Bitcoin has decreased over 10%, meaning a portfolio holding \$10,000 worth of Bitcoin at the start of the test time period would be worth \$8,941 by the end of the test time period. Figure 13 displays the predicted price against the realised Bitcoin price for the test period.



(a) Portfolio performances comparison (15 min interval)

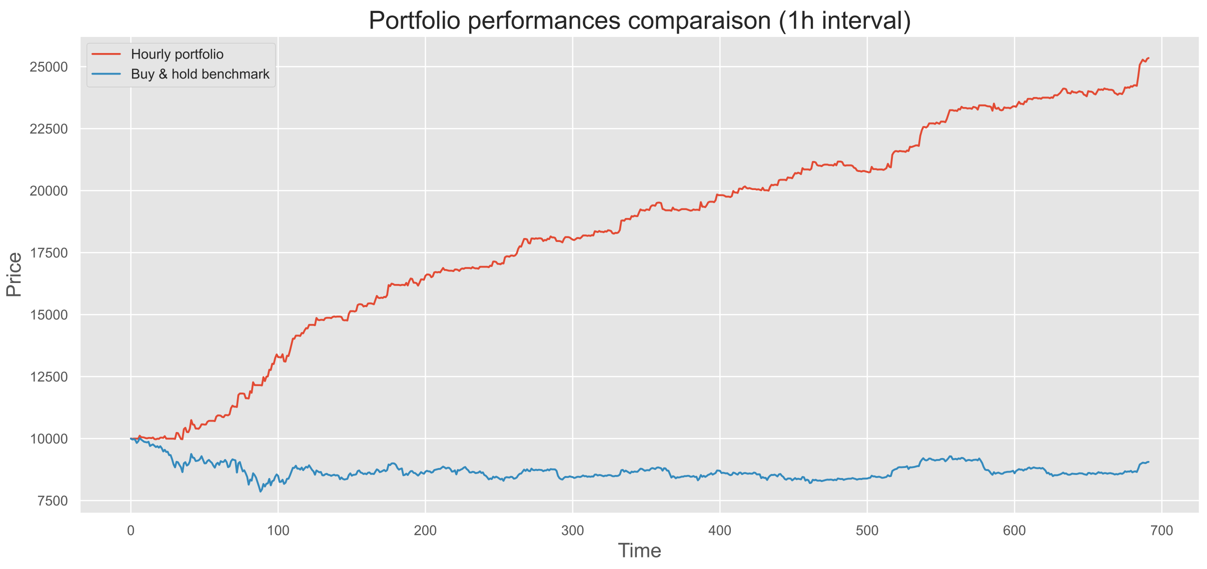


(b) Bitcoin 15 minutes interval buy and sell zones

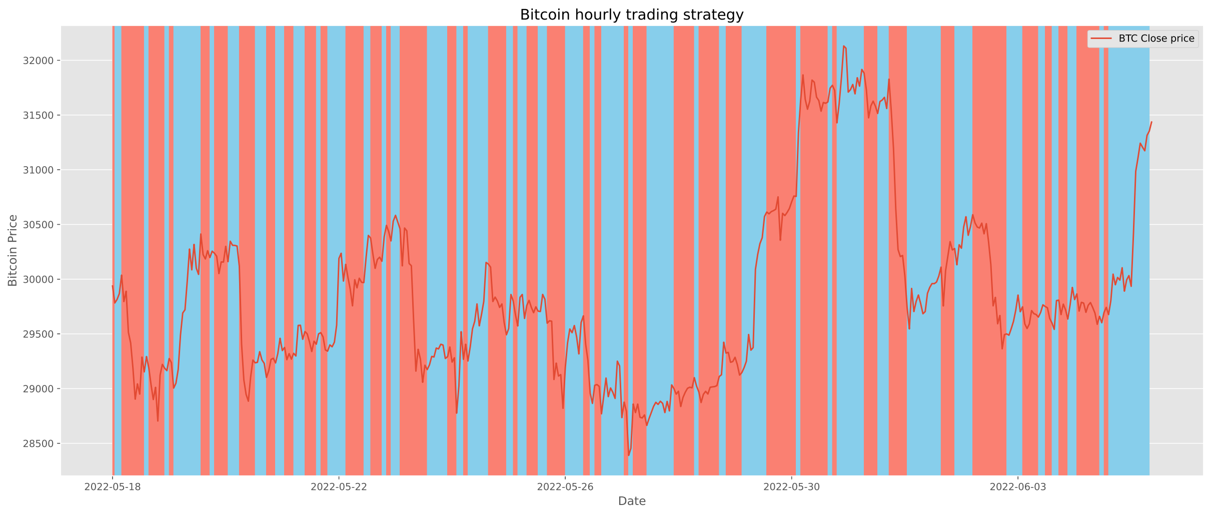
Figure 15: Portfolio performances plots (15 minutes)

We first analyse the most basic model which makes a prediction on the price every 15 minutes. The graph in Figure 15b represents the buying (blue) and selling (red) zones based on a predicted increase or decrease of the price every 15 minutes. We see from this graph that the zones are very scattered as the model is focused on short term movements. This model has a movement prediction accuracy of 53%. Although this is only 3% higher than a random guess, this offers considerable hedge to an investor especially in terms of short term prediction. We see from Figure 15a the purple line represents a portfolio which trades Bitcoin based on the model's decisions every 15 minutes without taking account of the fees. With almost a 20% return, this portfolio clearly outperforms the buy and hold approach and delivers high results. However, the main issue from this approach is the trade frequencies. The loss function used in the model is a MSE loss function which focuses solely on the distance of the prediction to the actual value but consider neither the direction of the prediction, which means whether the prediction rightfully predict the next price movement, nor the amount of trades made. These limitations will be discussed in sections 6.1 and 7. In a realistic environment we need to introduce trading fees and we use the rates from Binance, the world's largest cryprocurrency exchange platform. These rates are of 0.1% when buying a cryptocurrency and 0.1% when selling as well, totaling at a 0.2% fee per trade. The performance of the portfolio taking the fees into account is shown by the red curve in Figure 15a. With over 500 trades made in the course of one month, and a total 0.2% fee per trade, decomposed into 0.1% fee per transaction and a trade consists of two transactions one for buying the asset and one for selling it, the portfolio is highly underperforming the benchmark. Using this method, we lose about 60% of the portfolio's value in just one month. We then introduce a threshold before triggering a trade to filter out smaller noisy movements. The goal is to only trade trade when a significant movement is detected and thus reducing the amount of trades. We set this threshold to 0.2%, or double the transaction fee, for each action (buying and selling). Doing so reduces the number of trades by half and improves the performance resulting in a 30% loss. However, this is still not beating the buy and hold benchmark.

The main weakness if the first approach is that it contains a lot of noise and trading on such a short term basis does not give an indication on the longer term trend and is very sensible to short term movements in the market To decrease the frequency of trades and allow for a longer term vision approach, we expand the predictions on an hourly basis instead of the previous 15 minutes basis. This will give more robust results and is in line with previous works which have proven that predictive power tend to increase for longer forecast horizons [38]. We observe from Figure 16b that increasing the time frame to 1 hour largely reduces the amount of trades to about one third of what it was in the 15 minutes time horizons, or about 150 trades. Looking closely at this graph, we see that holding periods are longer and the model is able



(a) Portfolio performances comparison (1 hour interval)



(b) Bitcoin 1 hour interval buy and sell zones

Figure 16: Portfolio performances plots (1 hour)

to identify major movements and capitalise on them. The binary predicting accuracy jumps to 70% which is a huge improvement compared to our shorter term approach. We reproduce the same experiment trading with a portfolio of \$10,000 on the predicted hourly movement and after about a month of testing, the portfolio ends up with a capital of over \$25,000 which represents a 150% return on investment. With a sharpe ratio of 3, the model achieves excellent performance, largely beating the buy and hold benchmark as shown in Figure 16a.

5.3 Model generalisation

Performances reported in section 5.2 are unsustainable over the long term and raise the question of model generalisation and overfitting. The latter is particularly relevant since hyperparameters finetunning and model selection were performed based on test data results. Ideally, the model selection based on financial metrics should be done during the training process in order to avoid any kind of overfitting on the test data. However, as discussed in section 5.1, custom loss functions tailored for our trading task were found to perform worse than traditional loss functions. In this situation, the only way to evaluate the model with financial performance metrics is using test data. Furthermore, the performances of the model are subject to the market's conditions and cannot be guaranteed under all circumstances. Particularly in the period following the test data, the global cryptocurrency market experienced a steep sell-off period with Bitcoin prices falling over 30% in less than two weeks. This section will evaluate the performances of the model during the time period lasting from the 06/06/2022 to the 20/06/2022. During this period, Bitcoin prices fell from \$31,417 to \$20,525 which corresponds to a 35% decline. This means that a portfolio buying \$10,000 at the start of this test period would be worth \$6,533 by the end of it. Figure 17 displays the predicted price over the test period from a model trained on the same training data used previously and defined in section 4.5.

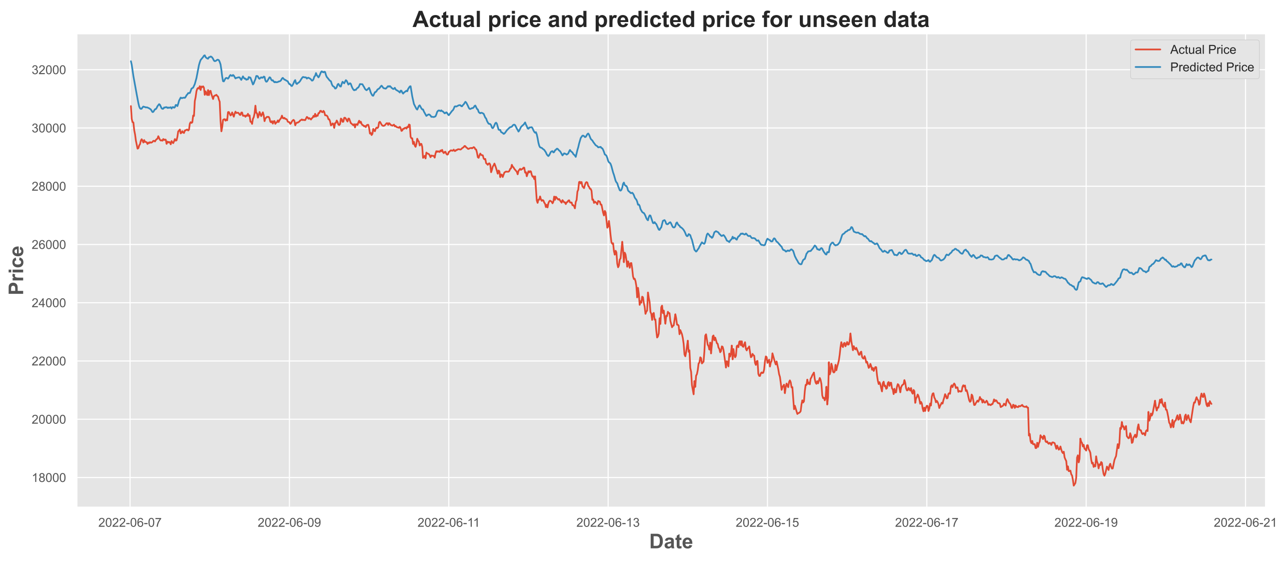


Figure 17: Predicted price against real price for unseen test data.

Similarly to the tests performed in section 5.2, we first apply the 15 minutes strategy onto the data, comparing for different threshold values in Figure 18. Impressively, even on the 15 minutes strategy and accounting for the fees, the model portfolio outperforms the buy and hold approach. It can be noticed that the portfolio starts outperforming the buy and hold as the price falls sharply. This is due to the fact that the employed strategy is a buy-sell strategy and not a long-short strategy. This means that if the price is predicted to go down, the strategy does not trade Bitcoin and simply stays out. A long-short strategy could benefit from both the rise and decline in price but would need to do more trades and hence could suffer further from transaction fees. Steep downward moves decrease the number of trades which attenuates the impact of transaction fees and the model is able to outperform the price action benchmark. Increasing the threshold to 0.2% and 0.5% further decreases the number of trades and results in higher performance once again.

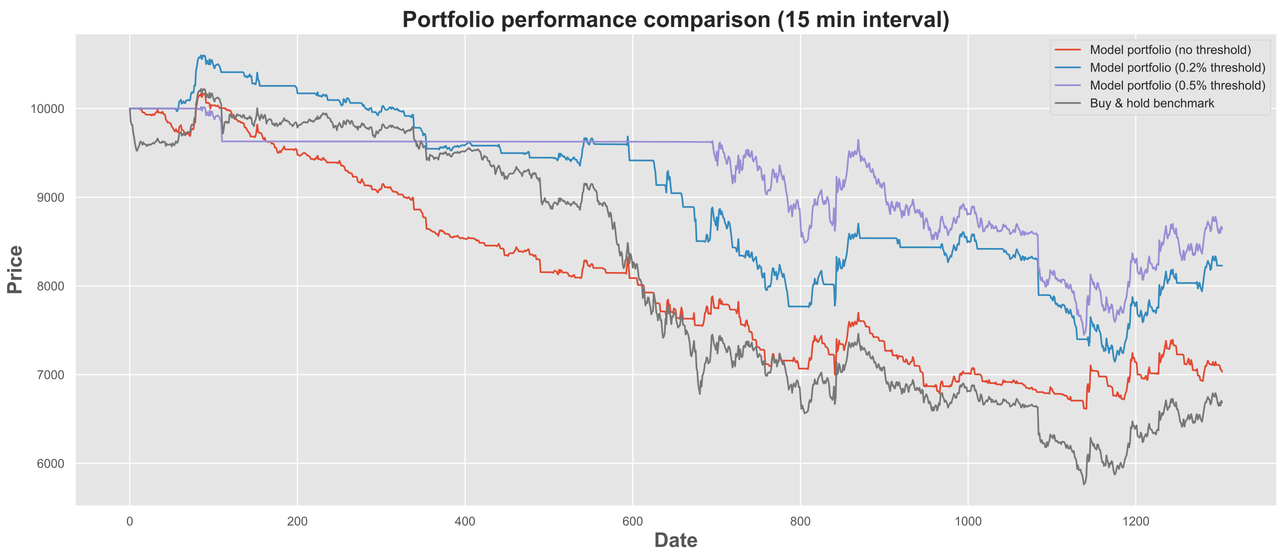


Figure 18: Portfolio performances comparison on 15 minutes timescale for unseen test data.

Finally, Figure 19displays the results of the portfolio on the hourly strategy to allow for a longer term strategy and receive confirmation for the trading signals. Once again, the strategy significantly outperforms the buy and hold benchmark returning over 60% during the testing taking advantage of the large upward retracement movements.

These observations on test data confirm the generalisation power of the model on different market conditions. However, it is worth noting that these kind of returns are unsustainable over the long term and, although a lot of precautions have been put in place to ensure no data leakage, look ahead bias or data overfitting was performed, more tests and evaluations are needed in order to establish the true predictive power of the model and the strategy.

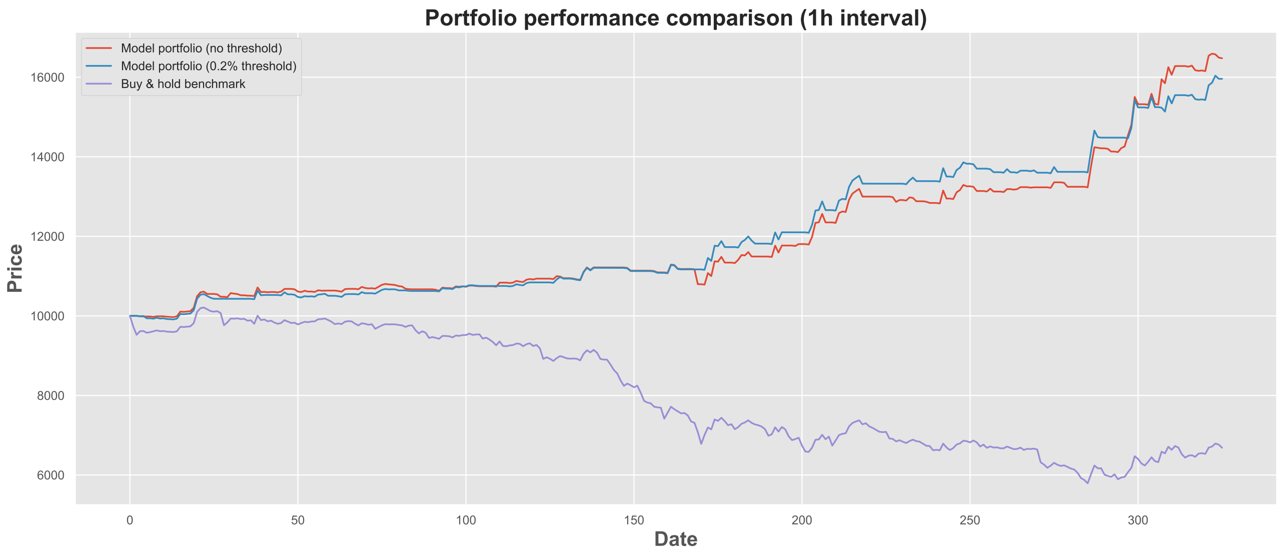


Figure 19: Portfolio performances comparison on 1 hour timescale for unseen test data.

6 Evaluation

6.1 Previous works

In this section, we will compare our model to three approaches. The first one is an ARIMA model, Autoregressive integrated moving average, which is a frequent model used for time series forecasting. The second model is defined in a 2021 paper named "Stock Price Prediction Using Machine Learning and LSTM-Based Deep Learning Models" [51] which uses a univariate deep stacked LSTM model to perform price prediction. It has achieved state of the art performance in terms of RMSE and correlation metrics but has not been tested on financial metrics. As discussed in section 5.2, a common problem of short term trading strategies is the large number of trades operated which affect the profits when taking transaction fees into account. The third model attempts to overcome this issue by designing a custom loss function which takes into account the movement of the predicted price compared to the realised movement. As opposed to a MSE loss which focuses on the the distance between the predicted value and the real value, this loss function also attempts to penalise predictions which forecast the price going up when it is in fact going down, in order to obtain results better suited for trading. The test period for these performance evaluations is shorter than the one in section 5.2 due to some of the models' characteristics and this is the reason why some returns might be different.

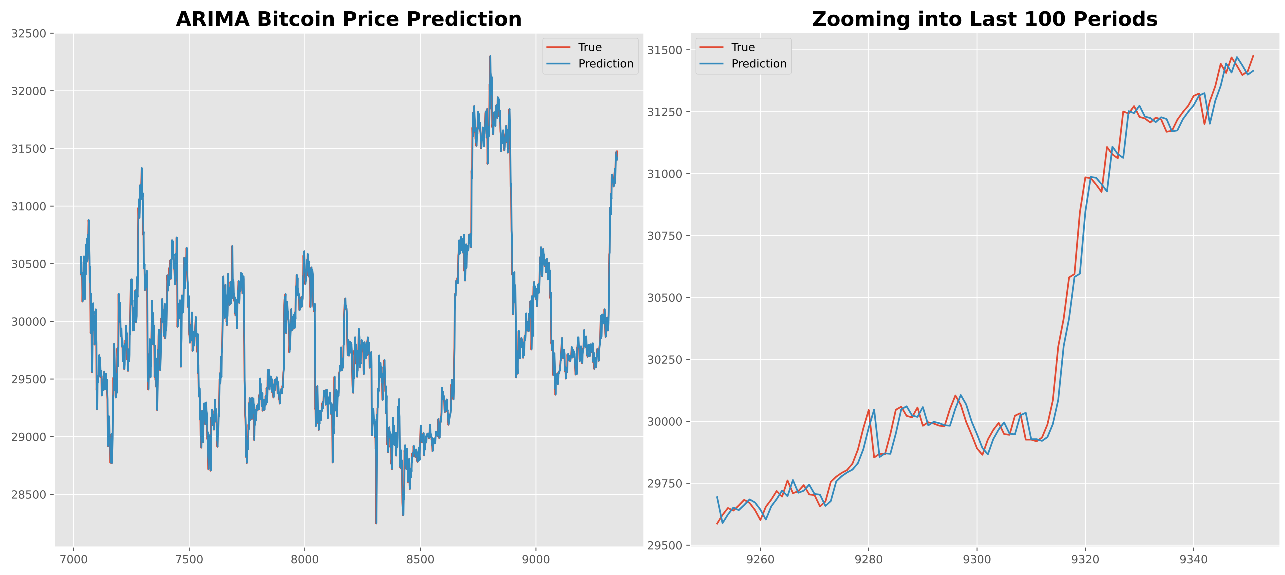


Figure 20: ARIMA model price prediction.

Figure 20 shows the predicted price against the real price using an ARIMA model. The data from the test set is used to perform prediction. From the graph on the left, it looks like the model stays really close to the actual price and the model has great forecasting power. However, zooming onto a specific period, we notice that the ARIMA model actually lags the price data. Due to the non-gaussian nature of price data ARIMA models are not well suited to predict such type of data and simply act as a 1 period lag model as presented in section 2.2. We used the buy and sell function to evalute the performance of such a model. Not surprisingly, the ARIMA model performs very poorly losing over 70% of the portfolio's value over the course of one month marked by an important number of trades, just under 700. Just like we did for our model, we set the threshold to trigger a trade to 0.2% in an attempt to eliminate some noise on small movements. The model performs better and still loses over 40% of the capital in one month. This demonstrates once again the inability of traditional time series prediction methods for our problem.

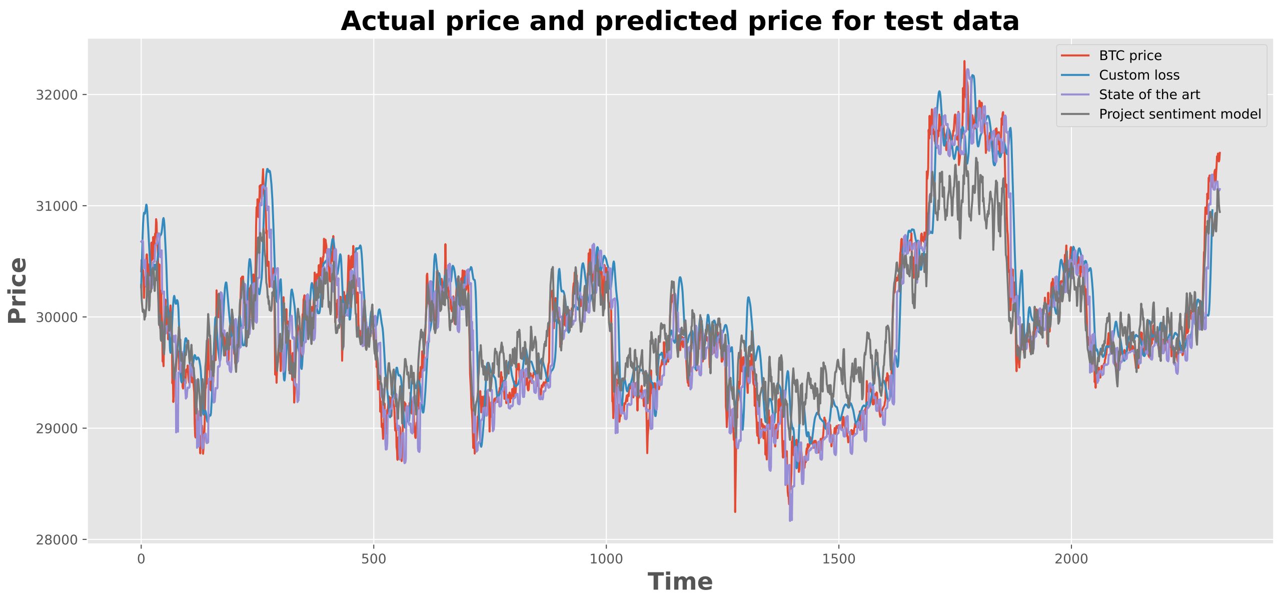
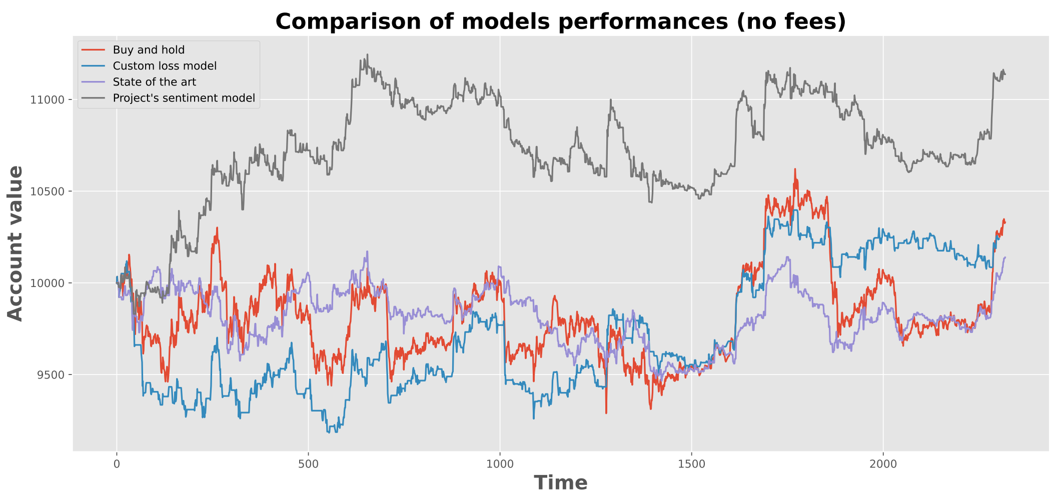


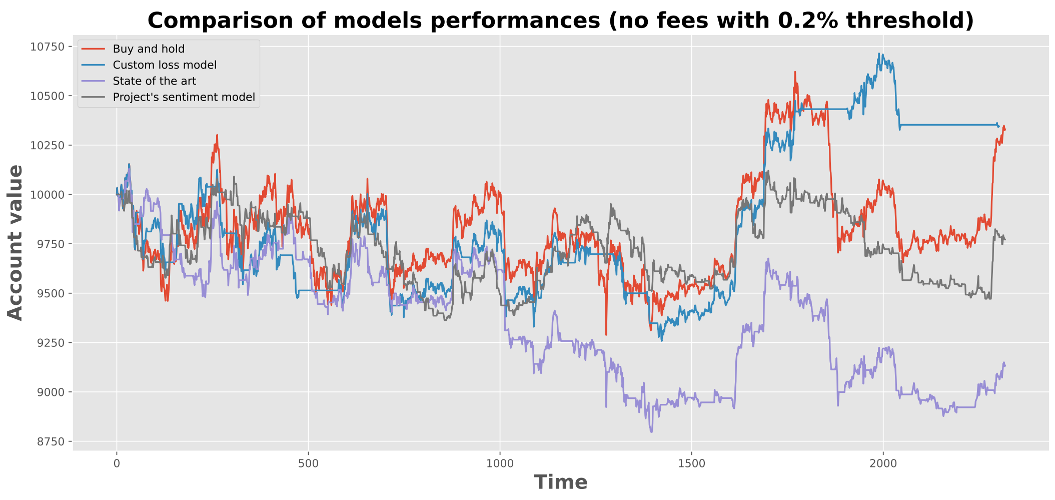
Figure 21: Predicted price against real price for different models.

Figure 21 displays the real Bitcoin price during the test period against the predicted price for the different models being tested. We observe that all the models seem to to pick up the trends in the price data nicely and follow the same patterns. Reiterating the performance evaluation steps performed in section 5.2, we observe in figure 22a the performance of portfolios following the strategy of each model when fees are not taken into account and in figure 22b when applying a 0.2% threshold. When fees are not taken into account, all models are profitable, however only the strategy using our model is able to beat the buy and hold portfolio. Since fees are not taken into account, reducing the number of trades by applying a threshold does not increase performance and performs worse. We can notice that the custom loss model portfolio is flat sometimes has it performs significantly less trades than other models as seen in table 5.

In order to perform a more realistic comparison of the models' performances, fees are now incorporated in the strategy as seen in figures 23a and 23b. With over 700 trades performed during the test period, the state of the art model performs very poorly when transaction fees are included compared to our own model with 462 trades, while the custom loss model performs



(a) Portfolio performances comparison for different models without transaction fees

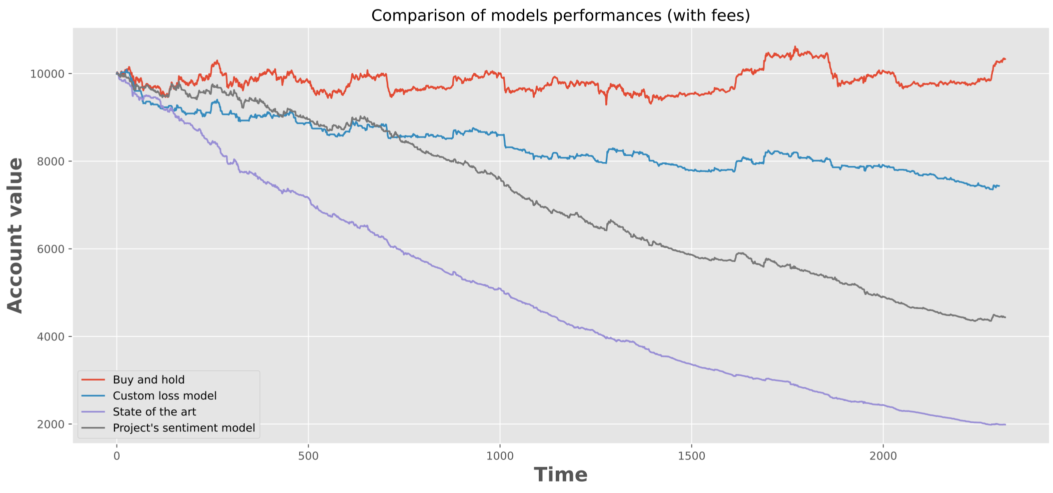


(b) Portfolio performances comparison for different models without transaction fees and a 0.2% buying and selling threshold

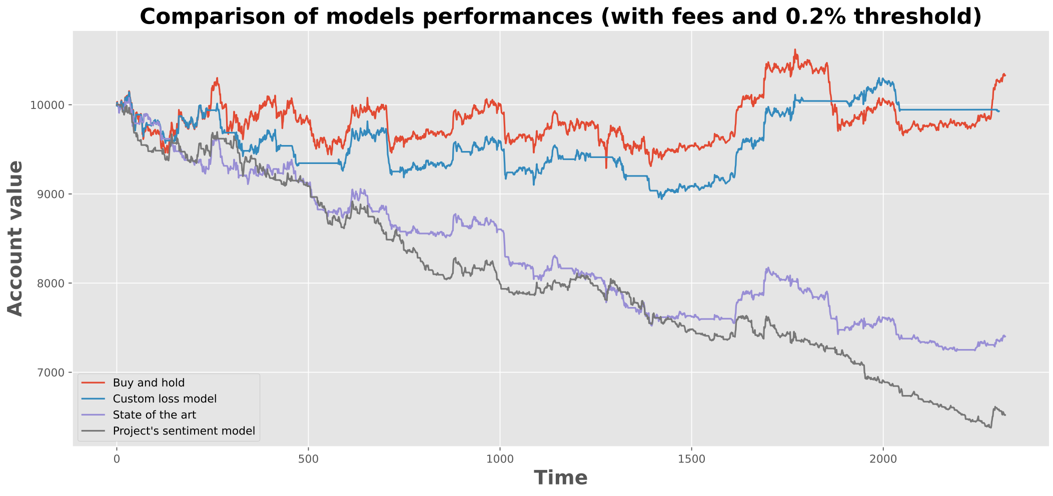
Figure 22: Portfolio performances comparison for different models

better with only 165 trades. In order to reduce the amount of trades, we apply the 0.2% threshold once again which significantly improves performance now that the fees are taken into account. Our model is the one with the most amount of trades even with the 0.2% threshold and is one with the worst performance over the test data.

There are two main weaknesses in our approach in a realistic trading environment. The first one is the fact that the model loss function does not penalises the model for making a lot of trades as it does incorporate the trading fees, or a threshold, in its loss function. The second one that because the model on data every 15 minutes, it contains a lot of noise and trading on such a short term basis does not give an indication on the longer term trend and is very sensible to short term movements in the market. For these reasons, we evaluate the models on an hourly basis to obtain predictions focused on a larger time scale.



(a) Portfolio performances comparison for different models with transaction fees



(b) Portfolio performances comparison for different models with transaction fees and a 0.2% buying and selling threshold

Figure 23: Portfolio performances comparison for different models (with fees)

Figure 24 displays the models' performances for the hourly strategy when fees are taken into account. Although all models perform similar amount of trades during the test period, between 100 and 150 trades made, our model is the only one able to run profitably and beat the buy and hold approach. Over the course of the 25 testing days, our model return a net profit of over 80% while all other strategies return negative performances. All models also have similar hourly accuracy which means our model does a better job at capturing longer term movement and trend and we are able to outperform previous state of the art methods trading profitably in a realistic environment.

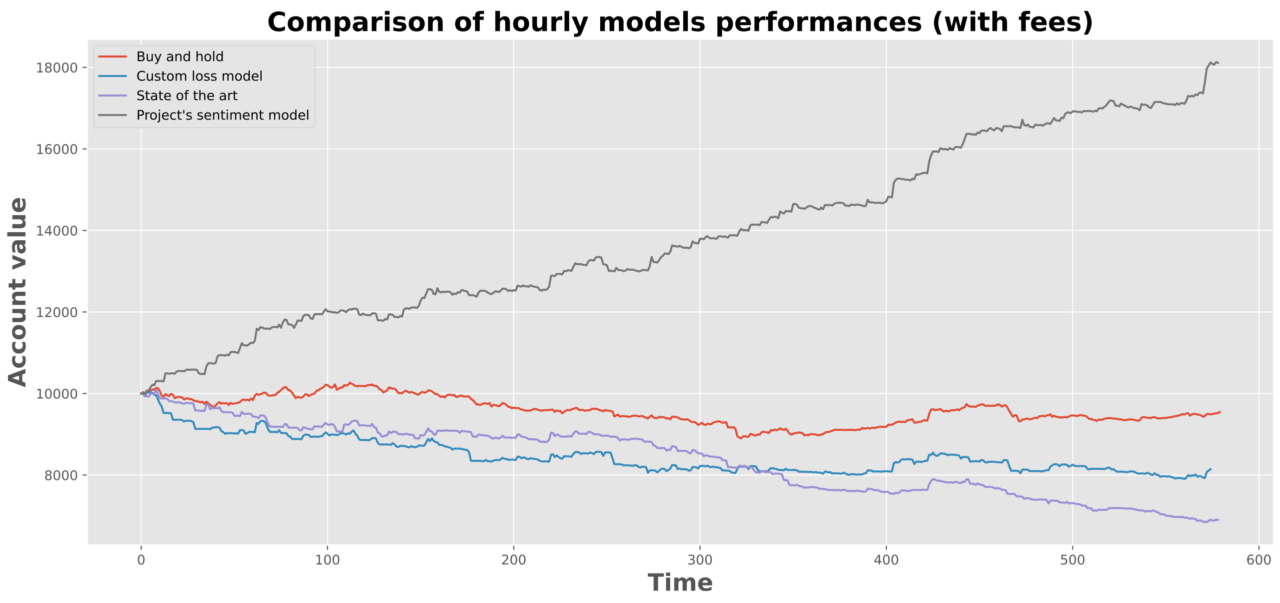


Figure 24: Portfolio performances comparison for different models on hourly interval

Model	**Number of trades**		
Name	15m with no threshold	15m with threshold	Hourly interval
Custom loss model	165	16	104
State of the art model	703	137	157
Project sentiment model	462	203	157

Table 5: Evaluating MSE and accuracy of different methods on Cifar-10 dataset

6.2 Sentiment features importance

As mentioned in section 5.2, the model achieves remarkable accuracy and financial performances on an hourly basis. This can lead us to question the main thesis of this report which is the role of the sentiment analysis features in this result. In this section, we aim to determine whether these features play a significant role in this result and compare the performances of models using sentiment analysis features and the ones who do not. To do so, the DeepLIFT algorithm is implemented as mentioned in section 4.7. Deep learning models are often treated as black boxes. These models are hard to explain and their outputs are often hard to interpret. The algorithm attempts to calculate the weight of each input in the final result by backpropagating the contributions of all neurons in the network to every feature of the input. Feature importance is an active field of research in machine learning and such techniques have limitations but they can give a general understanding of the feature's importance of a model. As seen from Figure 25 the sentiment analysis features seem to represent only part of the result, far behind the price related features. Looking at the combined numbers, sentiment analysis features represent a total of 26% of the model's prediction. The graph also informs us that more recent features are more important in the model predictions that older ones as we approach the prediction time.

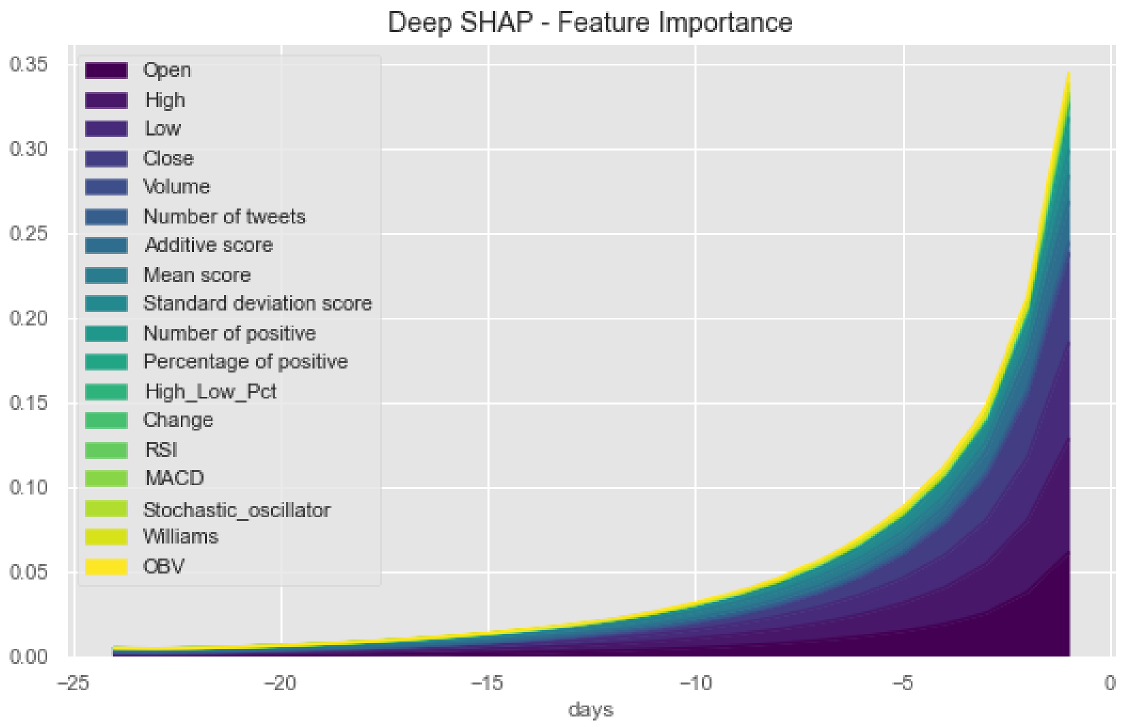
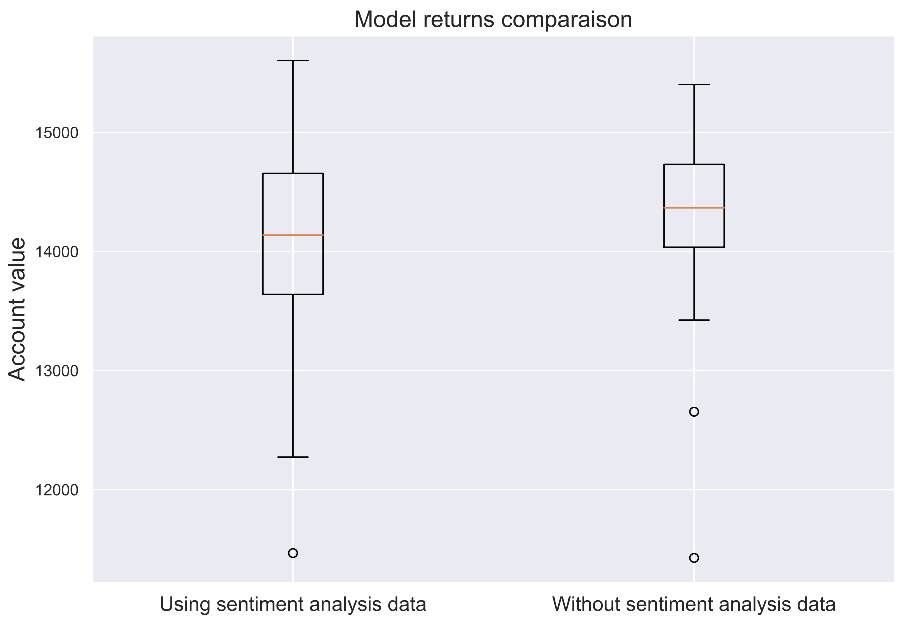
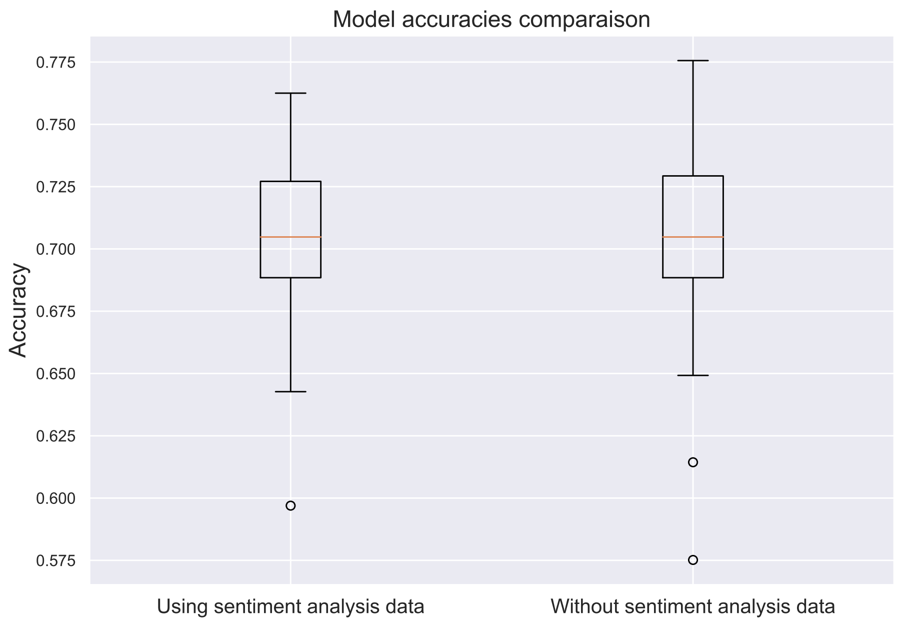


Figure 25: Features importance graph using DeepLIFT algorithm.

Although Figure 25 reveals that sentiment analysis features do play a role in the predictions, it does not tell the full story. We use boxplot graphs in Figures 26a and 26b to compare the average performances of models using sentiment features and models simply using price and volume related features. These box plots were made on smaller test datasets because of computational constraints by running 100 iterations. These models were then tested using the same trading algorithm used in section 5.2 on the hourly strategy. We see that the two types of model offer similar accuracy which tends to be over 70% with similar distribution. However, looking at the model returns, the models running without sentiment analysis tend to perform better than the model using sentiment features. The potential reasons behind this result will be discussed in section 7. This observation seems to be in line with findings from a recent 2022 research conducted by data scientists and machine learning engineers [43] which found no clear correlation between cryptocurrency price movement and their sentiment analysis. However, the research highlights clear correlation between the volume of tweets and price volatility and that "tweet frenzies precede price change and are likely influencing value" which suggests that Twitter data can provide leading indications regarding price movement and direction. Furthermore a recent research [45] has shown that a large feature set can negatively impact performance results both in terms of profits and accuracy results which can indicate that sentiment features actually acts like noise to the model and provide worse predictions.



(a) Returns comparison between 100 models using sentiment features and 100 models not using sentiment features.



(b) Accuracy comparison between 100 models using sentiment features and 100 models not using sentiment features.

Figure 26: Boxplot comparing performances on models using sentiment analysis compared to models not using sentiment analysis (for 100 models).

7 Future work

The current project and work frame presents limitations to be investigated and improved for future and further works. The main considerations comes from the data collection. The collected Twitter data contained a lot of noise and spams contributing to incomplete and noisy sentiment analysis features. In particular, a lot of tweets ended up being neutral or descriptive which did not provide meaningful information concerning the general sentiment. It has been observed and proven [43] that the volume of tweets is a critical information regarding cryptocurrency volatility and acts as a leading indicator for prediction purposes. As mentioned in section 4.1, we experienced some limitations regarding the quality of the data in the data collection phase. Precise tweet volume and collection of tweets based on cashtags and not hashtags would improve the quality of the sentiment data and can lead to better forecasting results. Another point to consider is the noise in the fluctuations of the sentiment data collected. Future works should consider using smoothing functions such as moving averages, as demonstrated in Figure 27. Using moving averages on the mean sentiment score with 100 and 1000 periods improves the correlation with the close price data from 0.5 with raw mean score to 0.9 and 0.97 respectively with the moving averages.

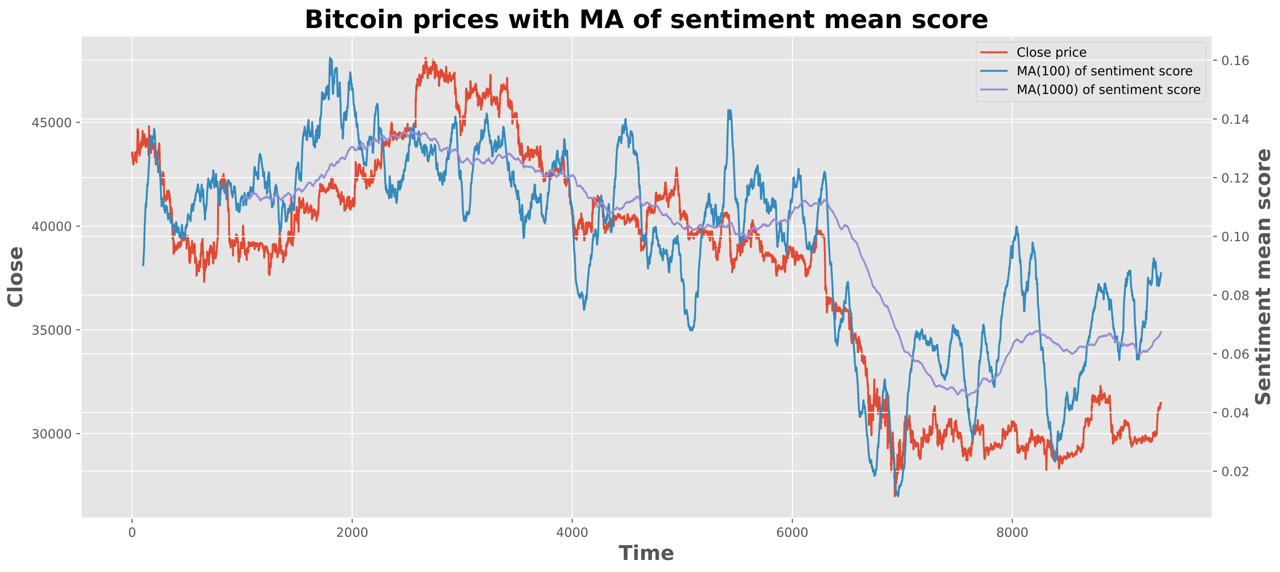


Figure 27: Bitcoin close prices compared to moving averages of sentiment features.

As discussed in sections 5.2 and 5.3, another limitation of the algorithm designed in this project is that the model selection is only done during the evaluation part with the test data and not during training. We highlighted the importance of using financial metrics in order to evaluate a model suitable for profitable trading but ideally this should be done during the training process. It was investigated that these limitations came from the fact that the traditionally used loss functions do not take into account the direction of the prediction compared to the direction of the data but simply focused on the distance to update the weights. Lastly, section 5.2 highlighted that the model performances were highly impacted by the high frequency of trades. Designing a trading algorithm which penalises frequent trades can significantly improve performance overtime when accounting for transaction fees.

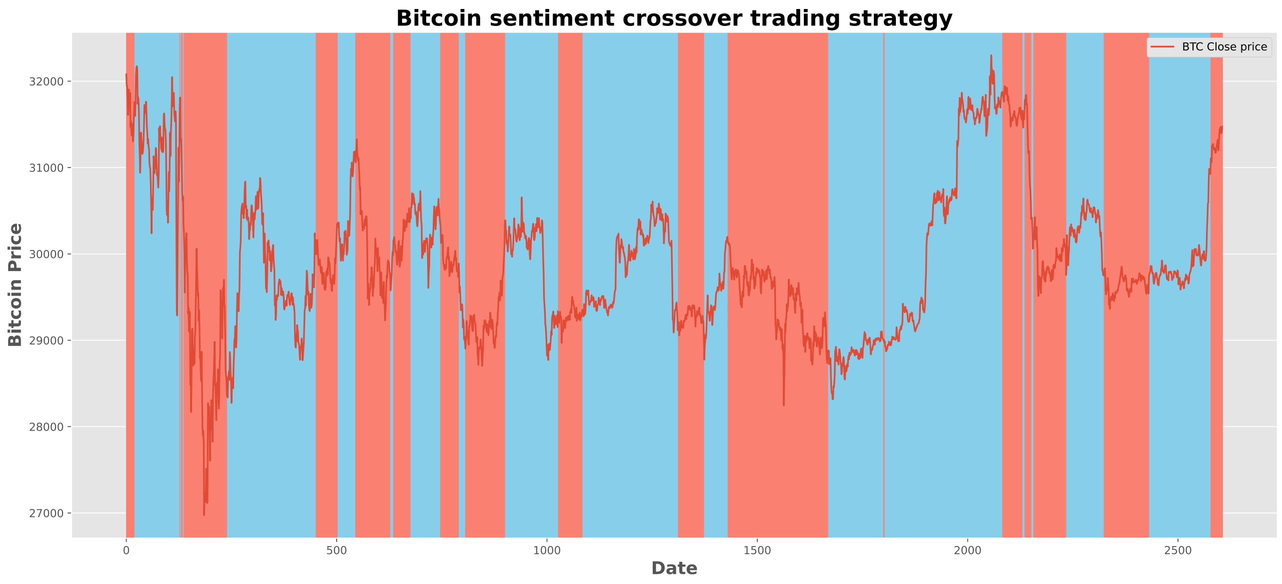


Figure 28: Bitcoin sentiment crossover strategy.

Finally, another consideration would be regarding the trading strategy. We have applied a simple trading strategy based only the next period predicted movement. However, this can be improved using multi step predictions in order to confirm the trend or supported by other indicators in order to build a more robust strategy. An example could be to use the sentiment features as indicators in the trading strategy. A common strategy is called a moving average crossover. Buying signals are triggered when a short term moving average crosses above a longer term moving average. Figure 28 displays the buying and selling zones applying the strategy with a 50-period moving average applied on the *Mean score* and the 200-period moving average. Using this strategy over the test period returns \$11,715 on a starting capital of \$10,000, accounting for fees, while a buy and hold would only be worth \$8,048. A more advanced trading strategy would be beneficial to maximise the results.

8 Conclusion

We have introduced a complete autonomous trading system using sentiment analysis fetching Twitter data as an input feature for neural networks in order to predict future prices of cryptocurrencies and make trading decisions profitably. The underlying hypothesis of this work is that opinions expressed on social media can function as useful predictors of price fluctuations, especially in cryptocurrency due to their volatile and reactive nature.

This work have shown that the model benefits from making predictions over longer forecasting horizons which gets rid of some of the inherent short term movement noise and is able to achieve a prediction accuracy of over 70%. This leads to a 150% return on investment over the course of one month of testing, significantly outperforming benchmarks and state of the art methods. Finally we have seen that both LSTM and GRU models are suitable for time series forecasting as they present great predictive powers due to their memory units with GRU benefiting from faster execution time. We have discussed the limitations and potentials future improvements of this work and cautiously evaluated the generalisation power of our model.

Regarding the general thesis of this report, we have found that sentiment analysis features do not offer increased performances and tend to add more model complexity without any seen benefits. This can largely be attributed to noisy inputs and data selection which can be investigating in future works.

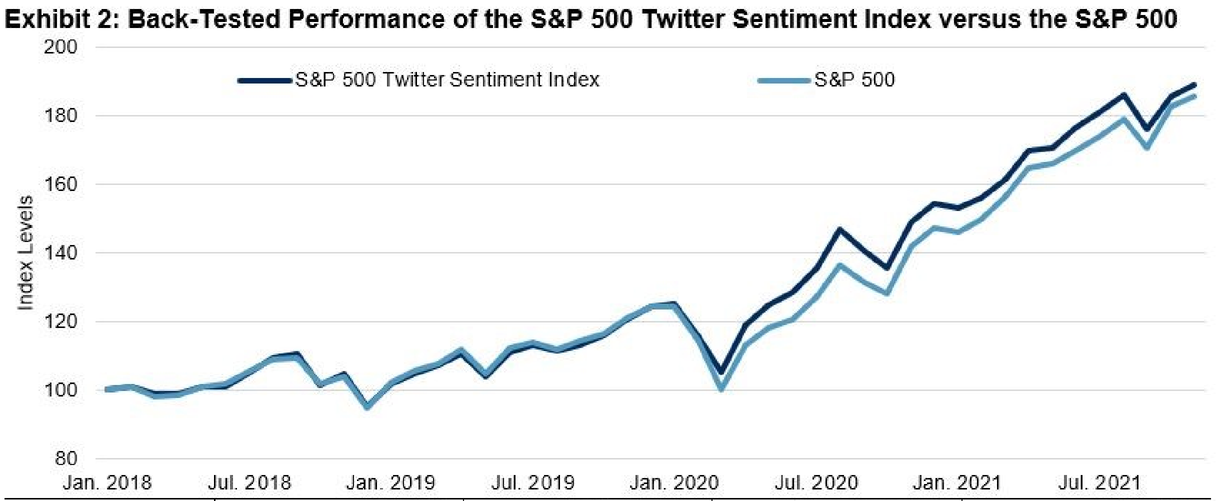
9 User guide

All the raw Twitter and sentiment data as well as all the code necessary to run the project and reproduce its results are available on:

`https://github.com/bprovendier/NN-for-Sentiment-Analysis`.

In order to create the combined file for the sentiment and price data, it is necessary to enter Twitter developer account credentials in the `credentials.py` file. Running the `tweet-extractor.py` file would result in collecting Twitter data every 15 minutes as detailed in section 4.1 Using `sentiment-analysis.py` would create the files containing all the sentiment features as well as specific files detailing the sentiment scores of each tweet. Finally `data-collection.py` creates the combined file with both the price data and the sentiment data. After that, once can simply run the `forecast.ipynb` notebook which will output all the results and graphs detailed in this report.

Appendix



INDEX	RETURN (ANNUALIZED, %)	VOLATILITY (ANNUALIZED, %)	RISK-ADJUSTED RETURN	MAXIMUM DRAWDOWN (%)
S&P 500 Twitter Sentiment Index	18.57	21.24	0.87	-30.54
S&P 500	18.00	21.52	0.84	-33.79

Figure 29: Performance of the S&P500 Twitter Sentiment Index compared to the S&P500 over the last three years.

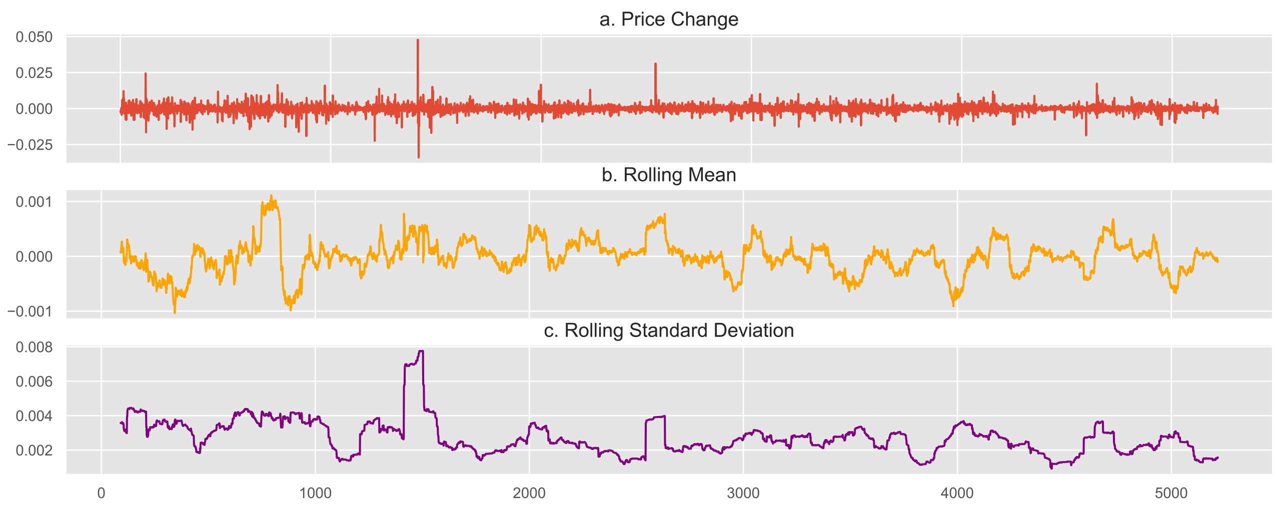
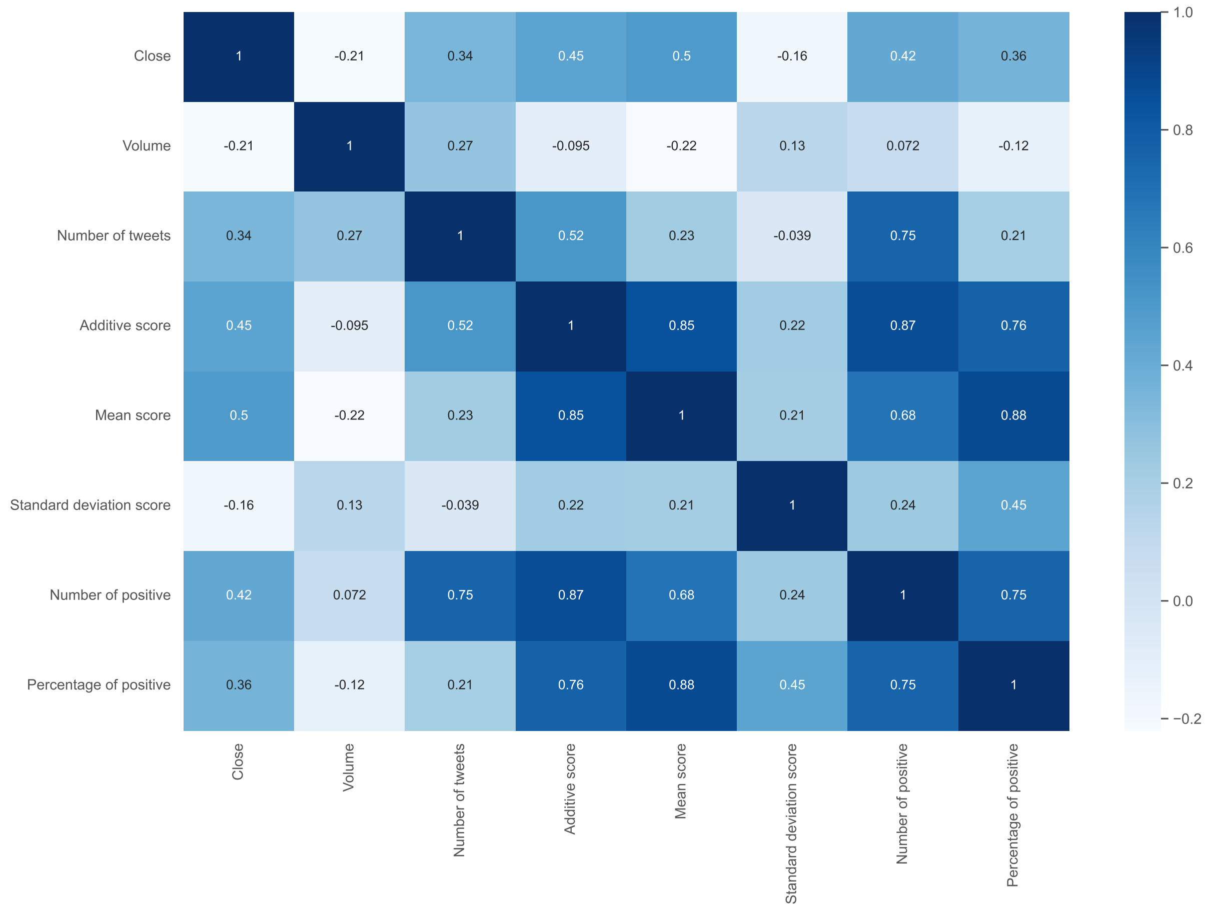


Figure 30: Dataset characteristics highlighting non-stationary nature of the task.



	Close	Volume	Number of tweets	Additive score	Mean score	Standard deviation score	Number of positive	Percentage of positive
Close	1	-0.21	0.34	0.45	0.5	-0.16	0.42	0.36
Volume	-0.21	1	0.27	-0.095	-0.22	0.13	0.072	-0.12
Number of tweets	0.34	0.27	1	0.52	0.23	-0.039	0.75	0.21
Additive score	0.45	-0.095	0.52	1	0.85	0.22	0.87	0.76
Mean score	0.5	-0.22	0.23	0.85	1	0.21	0.68	0.88
Standard deviation score	-0.16	0.13	-0.039	0.22	0.21	1	0.24	0.45
Number of positive	0.42	0.072	0.75	0.87	0.68	0.24	1	0.75
Percentage of positive	0.36	-0.12	0.21	0.76	0.88	0.45	0.75	1

Figure 31: Model features correlation matrix.

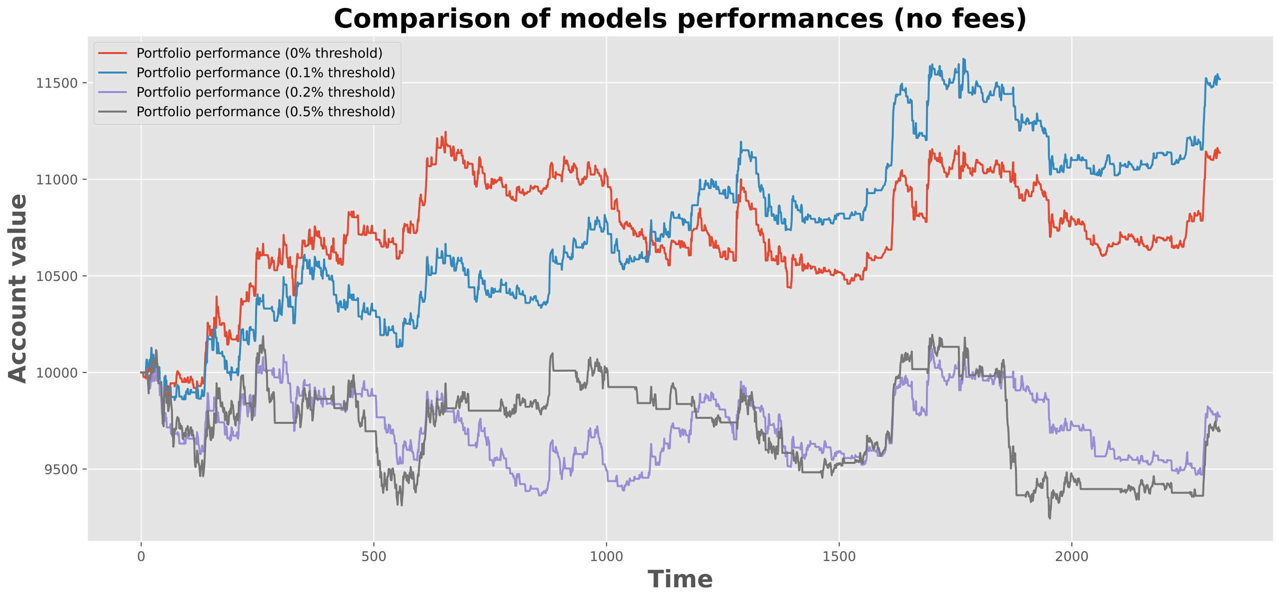


Figure 32: Performance comparison for models on 15 minutes time frame with threshold and no fees.

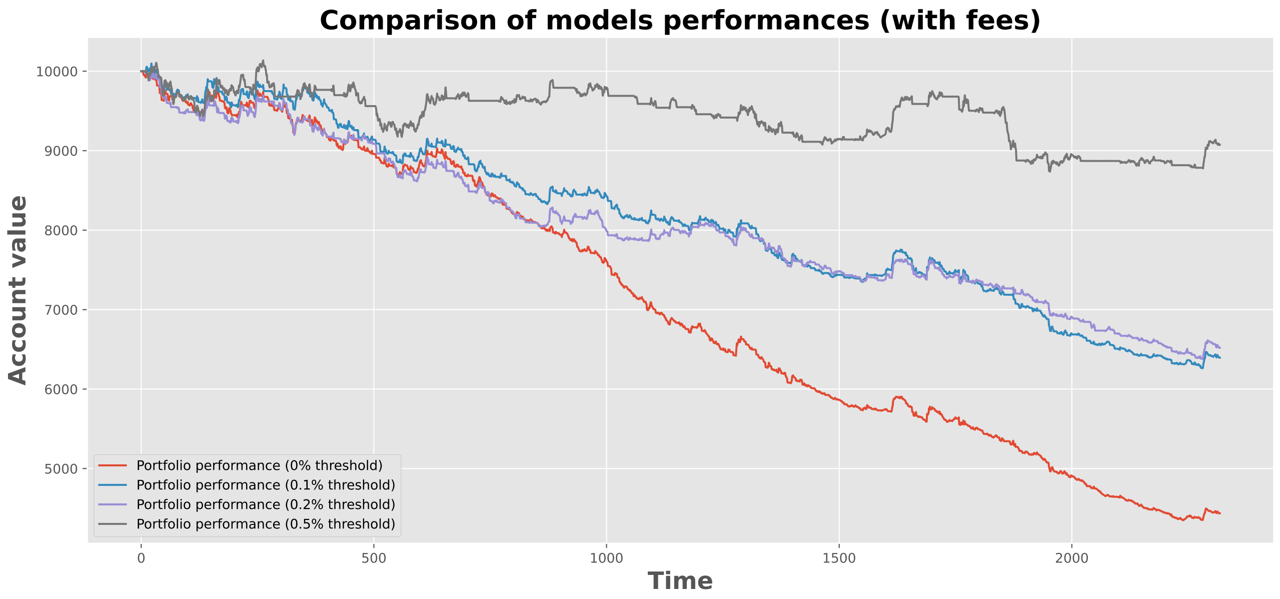


Figure 33: Performance comparison for models on 15 minutes time frame with threshold and with fees.

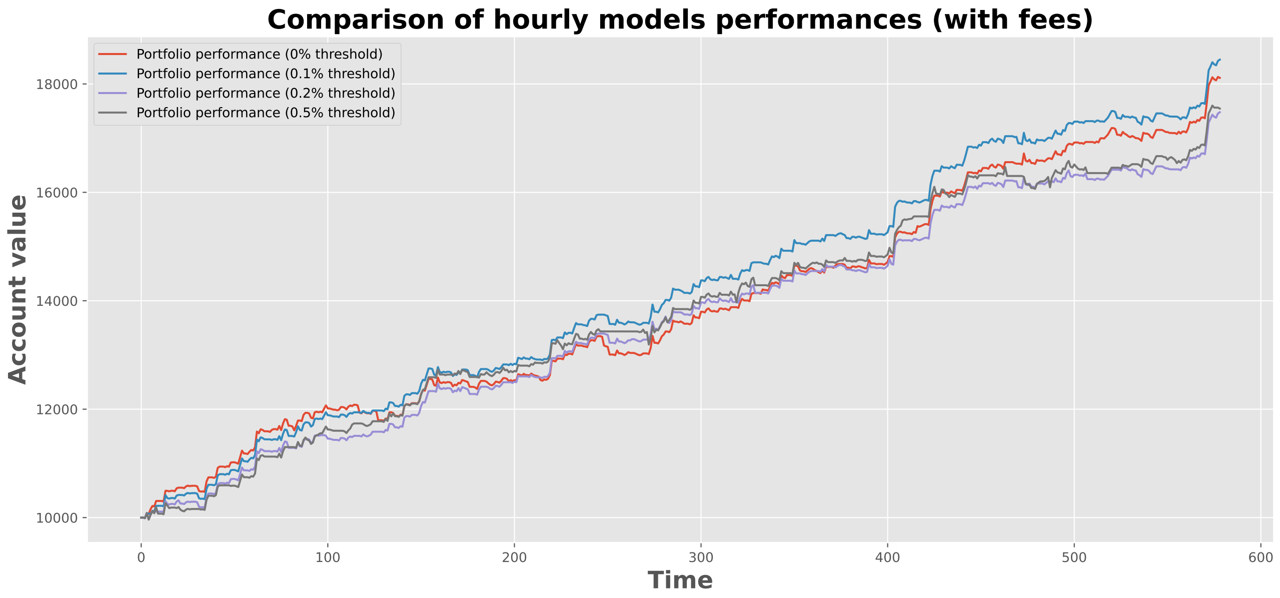


Figure 34: Performance comparison for models on 1h time frame with threshold and with fees.

References

[1] Sentiment analysis guide, 2021. URL: https://monkeylearn.com/sentiment-analysis/.

[2] Tokenizer, 2021. URL: https://huggingface.co/docs/transformers/main_classes/tokenizer.

[3] Bert (language model), Apr 2022. URL: https://en.wikipedia.org/wiki/BERT_(language_model).

[4] Cryptocurrency prices, charts and market capitalizations, 2022. URL: https://coinmarketcap.com/.

[5] Global markets - gset equities, 2022. URL: https://www.goldmansachs.com/what-we-do/global-markets/gset-equities.html.

[6] Long short-term memory, May 2022. URL: https://en.wikipedia.org/wiki/Long_short-term_memory#Applications.

[7] Research quarterly: Equities, Apr 2022. URL: https://www.sifma.org/resources/research/research-quarterly-equities/#:~:text=The%20U.S.%20equity%20markets%20are,market%20cap%2C%20or%20%2449%20trillion.

[8] Top cryptocurrency trading strategies 2022, Jun 2022. URL: https://primexbt.com/for-traders/cryptocurrency-trading-strategies/.

[9] Jethin Abraham, Danny W. Higdon, Johnny Nelson, and Juan Ibarra. Cryptocurrency price prediction using tweet volumes and sentiment analysis. 2018.

[10] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau. Sentiment analysis of Twitter data. In *Proceedings of the Workshop on Language in Social Media (LSM 2011)*, pages 30–38, Portland, Oregon, June 2011. Association for Computational Linguistics. URL: https://aclanthology.org/W11-0705.

[11] Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models, 2019. URL: https://arxiv.org/abs/1908.10063, doi:10.48550/ARXIV.1908.10063.

[12] E. Michael Azoff. *Neural Network Time Series Forecasting of Financial Markets*. John Wiley amp; Sons, Inc., USA, 1st edition, 1994.

[13] Malcolm Baker and Jeffrey Wurgler. Investor sentiment in the stock market. *Journal of Economic Perspectives*, 21(2):129–152, June 2007. URL: https://www.aeaweb.org/articles?id=10.1257/jep.21.2.129, doi:10.1257/jep.21.2.129.

[14] J.S. Bayer, P. van der Smagt, and J. Schmidhuber. *Learning Sequence Representations*. Universitätsbibliothek der TU München, 2015. URL: `https://books.google.co.uk/books?id=GRHLzQEACAAJ`.

[15] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. `doi:10.1109/72.279181`.

[16] Rolf Bertschi. Technical analysis - explained - credit-suisse.com, 2010. URL: `https://www.credit-suisse.com/pwp/pb/pb_research/OLDV2_technical_tutorial_en.pdf`.

[17] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973. `doi:10.1086/260062`.

[18] Ryan Browne. Dogecoin spikes more than 20% after elon musk says tesla will accept it as payment for merch, Dec 2021. URL: `https://www.cnbc.com/2021/12/14/dogecoin-price-surges-after-elon-musk-tweet-about-tesla-merch.html`.

[19] Pranjal Chakraborty, Ummay Sani Pria, Md. Rashad Al Hasan Rony, and Mahbub Alam Majumdar. Predicting stock movement using sentiment analysis of twitter feed. In *2017 6th International Conference on Informatics, Electronics and Vision 2017 7th International Symposium in Computational Medical and Health Technology (ICIEV-ISCMHT)*, pages 1–6, 2017. `doi:10.1109/ICIEV.2017.8338584`.

[20] Stuart G. Colianni, S. Rosales, and Michael Signorotti. Algorithmic trading of cryptocurrency based on twitter sentiment analysis. 2015.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL: `https://arxiv.org/abs/1810.04805`, `doi:10.48550/ARXIV.1810.04805`.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. URL: `https://aclanthology.org/N19-1423`, `doi:10.18653/v1/N19-1423`.

[23] Lucas Downey. Efficient market hypothesis (emh), Feb 2022. URL: `https://www.investopedia.com/terms/e/efficientmarkethypothesis.asp#:~:text=The%20efficient%20market%20hypothesis%20(EMH)%20or%20theory%20states%20that%20share,low%2Dcost%2C%20passive%20portfolio`.

[24] Eugene F. Fama. The behavior of stock-market prices. *The Journal of Business*, 38(1):34–105, 1965. URL: `http://www.jstor.org/stable/2350752`.

[25] Ronen Feldman. Techniques and applications for sentiment analysis. *Commun. ACM*, 56:82–89, 04 2013. `doi:10.1145/2436256.2436274`.

[26] Ronen Feldman, Benjamin Rosenfeld, Roy Bar-Haim, and Moshe Fresko. The stock sonar - sentiment analysis of stocks based on a hybrid approach. volume 2, 01 2011.

[27] Michael Galarnyk. Understanding boxplots, Jul 2020. URL: `https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51`.

[28] David Garcia and Frank Schweitzer. Social signals and algorithmic trading of bitcoin. *Royal Society Open Science*, 2, 06 2015. `doi:10.1098/rsos.150288`.

[29] Jack F. Gerulskis, Dante J. Knight, and Sean P. Dandeneau. Cryptocurrency trading program. Technical report, 100 Institute Road, Worcester MA 01609-2280 USA, April 2021.

[30] Snehal Gharat. What, why and which?? activation functions, Apr 2019. URL: `https://medium.com/@snaily16/what-why-and-which-activation-functions-b2bf748c0441`.

[31] George Giaglis, Ifigeneia Georgoula, Dimitrios Pournarakis, Christos Bilanakos, and Dionisios Sotiropoulos. Using time-series and sentiment analysis to detect the determinants of bitcoin prices. 10 2015. `doi:10.2139/ssrn.2607167`.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[33] Max Gulker. Bitcoin's price volatility is trending in the wrong direction, Feb 2018. URL: `https://www.aier.org/research/bitcoins-price-volatility-is-trending-in-the-wrong-direction/`.

[34] Mohammad Hamayel and Amani Owda. A novel cryptocurrency price prediction model using gru, lstm and bi-lstm machine learning algorithms. *AI*, 2:477–496, 10 2021. `doi:10.3390/ai2040030`.

[35] Daniel Herkert. Multivariate time series forecasting with deep learning, Jan 2022. URL: `https://towardsdatascience.com/multivariate-time-series-forecasting-with-deep-learning-3e7b3e2d2bcf`.

[36] Rob Hyndman. Another look at forecast accuracy metrics for intermittent demand. *Foresight: The International Journal of Applied Forecasting*, 4:43–46, 01 2006.

[37] Kyoung jae Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1):307–319, 2003. Support Vector Machines. URL: https://www.sciencedirect.com/science/article/pii/S0925231203003722, doi:https://doi.org/10.1016/S0925-2312(03)00372-2.

[38] Patrick Jaquart, David Dann, and Christof Weinhardt. Short-term bitcoin market prediction via machine learning. *The Journal of Finance and Data Science*, 7, 03 2021. doi:10.1016/j.jfds.2021.03.001.

[39] Manolis G. Kavussanos and Everton Dockery. A multivariate test for stock market efficiency: The case of ase. *Applied Financial Economics*, 11(5):573–579, 2001. doi:10.1080/09603100010013006.

[40] Kabir Kohli. Recurrent neural networks(rnn's) and time series forecasting, Mar 2020. URL: https://medium.com/analytics-vidhya/recurrent-neural-networks-rnns-and-time-series-forecasting-d9ea933426b3.

[41] Olivier Kraaijeveld and Johannes De Smedt. The predictive power of public twitter sentiment for forecasting cryptocurrency prices. *Journal of International Financial Markets, Institutions and Money*, March 2020. doi:10.1016/j.intfin.2020.101188.

[42] Tianyu Li, Anup Chamrajnagar, Xander Fong, Nicholas Rizik, and Feng Fu. Sentiment-based prediction of alternative cryptocurrency price fluctuations using gradient boosting tree model. *Frontiers in Physics*, 7:98, 07 2019. doi:10.3389/fphy.2019.00098.

[43] Monica Lin, Monica Lin, Christoph Meier, Matthew Parker, and Kiran Ravella. How to correlate the trend in crypto prices to a twitter sentiment model using databricks delta - the databricks blog, May 2022. URL: https://databricks.com/blog/2022/05/02/introduction-to-analyzing-crypto-data-using-databricks.html.

[44] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc.

[45] Vahur Madisson. Forecasting and trading financial time series with lstm neural network. 2021.

[46] Feng Mai, Zhe Shan, Qing Bai, Xin (Shane) Wang, and Roger H. L. Chiang. How does social media impact bitcoin value? a test of the silent majority hypothesis. *Journal of Management Information Systems*, 35:19 – 52, 2018.

[47] J. Makhoul. Linear prediction: A tutorial review. *Proceedings of the IEEE*, 63(4):561–580, 1975. `doi:10.1109/PROC.1975.9792`.

[48] Burton. G. Malkiel. *A Random Walk Down Wall Street.* Norton, New York, 1973.

[49] Danilo P. Mandic and Jonathon A. Chambers. *Recurrent neural networks for prediction - learning algorithms, architectures and stability.* John Wiley amp; Sons Incorporated.

[50] Publisher Shift Markets. Advantages of algorithmic trading. URL: `https://www.nasdaq.com/articles/advantages-algorithmic-trading-2019-06-07`.

[51] Sidra Mehtab, Jaydip Sen, and Abhishek Dutta. Stock price prediction using machine learning and lstm-based deep learning models. *Communications in Computer and Information Science*, page 88–106, 2021. `doi:10.1007/978-981-16-0419-5_8`.

[52] Yuichi Nagahara. Non-gaussian distribution for stock returns and related stochastic differential equation. *Financial Engineering and the Japanese Markets*, 3(2):121–149, 1996. `doi:10.1007/bf00868083`.

[53] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[54] Tshepo Chris Nokeri. Implementing machine learning for finance. 2021. `doi:10.1007/978-1-4842-7110-0`.

[55] Kizito Nyuytiymbiy. Parameters and hyperparameters in machine learning and deep learning, Mar 2022. URL: `https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac`.

[56] Christopher Olah. Understanding lstm networks, Aug 2015. URL: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[57] Joerg Osterrieder and JULIAN LORENZ. A statistical risk assessment of bitcoin and its extreme tail behavior. *Annals of Financial Economics*, 12:1750003, 03 2017. `doi:10.1142/S2010495217500038`.

[58] Sasank Pagolu, Kamal Reddy, Ganapati Panda, and Babita Majhi. Sentiment analysis of twitter data for predicting stock market movements. pages 1345–1350, 10 2016. `doi:10.1109/SCOPES.2016.7955659`.

[59] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of*

Machine Learning Research, pages 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL: `https://proceedings.mlr.press/v28/pascanu13.html`.

[60] Ross C. Phillips and Denise Gorse. Predicting cryptocurrency price bubbles using social media data and epidemic modelling. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7, 2017. `doi:10.1109/SSCI.2017.8280809`.

[61] Lawrence R. Rabiner and Ronald W. Schafer. Introduction to digital speech processing. *Foundations and Trends® in Signal Processing*, 1(1–2):1–194, 2007. `doi:10.1561/2000000001`.

[62] Kenneth Rapoza. Can 'fake news' impact the stock market?, Jun 2021. URL: `https://www.forbes.com/sites/kenrapoza/2017/02/26/can-fake-news-impact-the-stock-market/?sh=46455c472fac`.

[63] Eric Reed. Crypto vs. stocks: Which is better?, Oct 2021. URL: `https://finance.yahoo.com/news/crypto-vs-stocks-better-201720889.html`.

[64] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020. URL: `https://aclanthology.org/2020.tacl-1.54`, `doi:10.1162/tacl_a_00349`.

[65] James Royal. Cryptocurrency vs. stocks: What's the better choice for you? URL: `https://www.bankrate.com/investing/crypto-vs-stocks/`.

[66] Seb. An introduction to neural network loss functions, Apr 2022. URL: `https://programmathically.com/an-introduction-to-neural-network-loss-functions/`.

[67] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3145–3153. PMLR, 06–11 Aug 2017. URL: `https://proceedings.mlr.press/v70/shrikumar17a.html`.

[68] Therese Simberg. Introducing the s&p 500 twitter sentiment index series, Nov 2021. URL: `https://www.indexologyblog.com/2021/11/18/introducing-the-sp-500-twitter-sentiment-index-series/`.

[69] William C. Spaulding. Technical analysis (ta). URL: `https://thismatter.com/money/technical-analysis/technical-analysis.htm`.

[70] Ralph Sueppel. Measuring the value-added of algorithmic trading strategies, Oct 2021. URL: `https://www.sr-sv.com/measuring-value-added-of-algorithmic-trading-strategies/#:~:text=These%20metrics%20include%20mean%20squared,effective%20return%20computed%20ex%2Dpost`.

[71] Sik-Ho Tsang. Review: Empirical evaluation of gated recurrent neural networks on sequence modeling (gru), Nov 2021. URL: `https://sh-tsang.medium.com/review-empirical-evaluation-of-gated-recurrent-neural-networks-on-sequence-modeling-gru-2adb86559257`.

[72] Ah Chung Tsoi. Recurrent neural network architectures: An overview. *Adaptive Processing of Sequences and Data Structures*, page 1–26, 1998. `doi:10.1007/bfb0053993`.

[73] Franco Valencia, Alfonso Gómez-Espinosa, and Benjamin Valdes. Price movement prediction of cryptocurrencies using sentiment analysis and machine learning. *Entropy*, 21:1–12, 06 2019. `doi:10.3390/e21060589`.

[74] Nicolas Vandeput. Forecast kpi: Rmse, mae, mape amp; bias, Jul 2021. URL: `https://towardsdatascience.com/forecast-kpi-rmse-mae-mape-bias-cdc5703d242d`.

[75] Jacques Vella Critien, Albert Gatt, and Joshua Ellul. Bitcoin price change and trend prediction through twitter sentiment and data volume. *Journal of Financial Innovation*, 8, 05 2022. `doi:10.1186/s40854-022-00352-7`.

[76] Jacques Vella Critien, Albert Gatt, and Joshua Ellul. Bitcoin price change and trend prediction through twitter sentiment and data volume. *Journal of Financial Innovation*, 8, 05 2022. `doi:10.1186/s40854-022-00352-7`.

[77] Lizhong Wu, M. Niranjan, and F. Fallside. Fully vector-quantized neural network-based code-excited nonlinear predictive speech coding. *IEEE Transactions on Speech and Audio Processing*, 2(4):482–489, 1994. `doi:10.1109/89.326608`.