Prediction of Cryptocurrency Price Movements from Order Book Data Using LSTM Neural Networks

April 2019

Andrei Alexandru Maxim

BSc. Computer Science

Supervisor: Dr. Denise Gorse

Abstract

The purpose of this work is to investigate the use of a long-short term neural network (LSTM) for time series prediction, more specifically for cryptocurrency price variation forecasting. The Bitcoin cryptocurrency market is here used as a testbed for a newly-proposed trend prediction machine learning algorithm that uses smoothed input data extracted from the limit order book of the market. The novelty of this project will reside not only in using LSTM networks in a cryptocurrency context (a relatively unexplored area in the academic literature), but also in its use of the above-mentioned ingenious input smoothing strategy, proposed by one of the studied papers.

Contents

Chapter 1 – Introduction

1.1 Early Financial Markets and the Dawn of Market Prediction

Even though "banking" as a concept has been known to mankind since the earliest of days, from the ancient societies of Greece, China, India and even The Roman Empire (workers there getting grain loans from the sovereignty), the birth of a financial market similar to what we are familiar with nowadays comes no sooner than the late 17th century: London goldsmiths offered to store the gold of customers in their secured vaults for a small fee, exchanging the physical quantity for a signed paper stating the exact amount they had in store – this was the beginning of banknotes. In the same period, Amsterdam was the first city to witness the emergence of a modern stock market layout, trading capital, bonds and stocks. Understandably, people immediately tried to seek advantage from these newly established markets and were hungry for profits. Although the movement on the markets has always been hugely influenced by an entanglement of factors—political, demographical, philosophical and so on—modern approaches to understanding economical movements relate preponderantly on a mathematical interpretation of the situation as a whole.

1.2 In Search of a Forecasting Model

Market forecasting can be scientifically approached from two perspectives:

- **Fundamental analysis**, where a company's past performance is studied, and the potential investor makes predictions as to whether an investment may be profitable or not, counting on the dependence between capital reward and performance. This kind of decision making may also be carried out in a top-down manner, starting from objective analysis of global economy, down to country specifics, domain perspectives, leaving company position and particularities as the final layer of investigation.
- **Technical analysis** on the other hand, disregards the company's financial profile and makes predictions only on the basis of past stock prices, looking for common

patterns that supposedly repeat themselves and therefore could be predicted to some extent. **Quantitative analysis**—the construction of mathematical and statistical models that find patterns in the markets—has been a huge focus of contempory interest in trading. Quantitative analyses draw upon large historical records of prices and require a human professional, skilled in mathematics and experienced in economics, to construct a model from the patterns he/she may be able to pick up.

1.3 Motivation for the Use of Automated Inference

Scalability is an inevitable problem for quantitative analysis—no matter how many experts a team of analysts might consist of, there is a limit as to how much information can be shared efficiently between humans and how much integrity the resulting solution would then have. It is obvious that any approach of this kind would be at a disadvantage with the ever-increasing number of parameters that realistic financial decisions must consider. In addition, the importance of granular-level data—the small-scale patterns—is completely overlooked because of the impossibility of processing these large amounts of data. The vast availability of data in the modern era is thus in fact creating a problem for quantitative analysts. A machine is needed with greater concentrated computational power, more controllable and larger memory, as well as undisturbed attention to detail in order to find the complex patterns that govern the market movements of today.

1.4 Artificial Intelligence, Machine Learning and Neural Networks

Artificial intelligence (AI) is a broad term covering all attempts to create machines that, if they were human, would be judged to be exercising some degree of intelligence. In earlier years it predominantly meant rule-based automatic inference systems which would be able to grant human-made machines the ability of reasoning. The domain has since split into several fields that nowadays function mostly independently, such as robotics, machine learning, natural language processing, human-computer interaction etc.

Machine learning (ML) is a term that refers solely to the ability of a computer program to acquire the ability to perform a given task with no code having been explicitly written for performing the task, but with numerous examples given which would allow a

learning algorithm to create its own solution (e.g. to a classification problem) based on a large number of trial and error attempts performed on the shown set (the *training set*).

There are numerous algorithms used for machine learning, however a specific subset of them are in the current research spotlight: **artificial neural networks (ANNs)**. These are algorithms that have drawn their inspiration from the inner workings of the brain, although, artificial neurons and the networks constructed from them only represent real biological brains at a superficial level. The neurons, which assess information coming from the outside world, and from their neighbours, are the central part of these algorithms, and in modern technologies, their complex landscape of interconnections reaches back through several hidden layers (similar to neuronal clusters) that create a considerable depth (number of layers) of the network. This is why this niche of AI has come to be known as "deep learning".

1.5 Structure of this Report

The remainder of this report will proceed as follows. Chapter 2 will give a brief background to cryptocurrency markets and the use of neural networks for time series prediction (focusing on the LSTM model), as well as surveying some relevant literature, Chapter 3 will cover the methodology of our proposed approach, and Chapter 4 will explore our findings. The concluding Chapter – 5 – will discuss our results and draw conclusions as to the effectiveness of our methods, as well as suggesting potential future work.

Chapter 2 – Background and Literature Survey

2.1 Cryptocurrency Markets

2.1.1 Cryptocurrency Basics

Cryptocurrencies are decentralised ledger-based assets not issued by any government, not covered in physical countervalue and untransferable to the physical world. They only exist in the digital world and their physical existence is replaced by a system of mutual trust between users, in the form of a register that is being updated simultaneously over numerous computers throughout the world. The form in which this market is digitally stored on computers is a called a *blockchain*, which is a structure similar to a linked list, where each node is actually an encrypted sequence of transactions.

Despite numerous flaws in time and space performance, anonymity preservation and inconsistencies in the rewarding system for miners, Bitcoin remains the most widely used cryptocurrency, perhaps mainly because of the cryptocurrency's notoriety and age. The Bitcoin blockchain receives a new block approximatively once every eight minutes, consisting of around 3000 transactions. While the design intention was for the market to be decentralised—meaning there would be no unique authority that regulates the traffic, or, more specifically, there would not be a single register stored somewhere that exerts a higher privilege over the others—there have appeared factors that undermine this desire. Because of the way that Bitcoin offers rewards to the users that lend their computational power for block construction, it has become profitable for so called “mining farms” to emerge onto the market. The operation of these server farms has the ironical effect of re-centralising the network due to the physical clustering of connections that occurs. The larger the capabilities of such a mining farm, the stronger its confidence in providing preferential services to customers that are willing to pay more, i.e. prioritizing a single transaction, one that payed a fee depending on how fast the client wanted their transaction to be processed. It thus becomes hard to predict what priority a new transaction will be given on the network.

2.1.2 Cryptocurrencies as Financial Assets

Although the cryptocurrency market seems to be so different technologically compared to any other market known so far, it is still governed by the principles of offer and demand. Wherever an asset is eligible to be traded, the relationship between the ones that sell it and the ones that wish to buy it will always be quantifiable. One tool for representing a market at a specific moment in time is the *limit order book*.

Figure 1 - Bitcoin Limit Order Book snapshot (by Courtesy of the *CoinbasePro* platform)

The contents of this representation are very intuitive to grasp. The image shown above (fig. 1) is a cumulative graph of offer and demand. On the X-axis there are values expressed in US dollars ($), so naturally, on the right side, in orange, relating to a higher price of Bitcoin, there are the positions of sellers that want to sell Bitcoin at a value more than the market's mean, and on the left side, in green, there are the positions of buyers that want to buy Bitcoin for a cheaper price. Logically, the place where those two come close together is the current market value. More specifically, the *arithmetic mean* between the lowest *bid* (leftmost orange point) and highest *ask* (rightmost green point) is regarded as the market price, since the two sides of the order book never intersect. The Y-axis represents cumulative volumes for each position—this is why the height is uniformly increasing to the edges, not because there are increasingly many positions that want to buy or sell at that price, but because the previous depth values (closer to the mean) are also included in the current height. This means that a good indication of how many positions are up for a specific price is to analyse the so-called "walls," the straight upward shifts in coloured areas at certain price values.

2.2 Neural Networks, Deep Learning, and LSTMs

The Introduction chapter explored the motivations for using machine learning, focusing on the increased granularity of data combined with the incapacity of the human mind to account for all these highly-detailed movements. Although machine learning has only recently become very popular, the theory that underlies this technology is not at all new.

Neural networks have been around from the early 1960s and have undergone several waves of enthusiasm interspersed with losses of interest (e.g. corresponding to the rise of interest in rule-based A.I in the 1970s). Most recently, there has been a wave of enthusiasm for what is known as *deep learning*. Technology developments driven mainly by the needs of the gaming industry (the evolution of graphical processing units to have thousands of computational cores) have made possible the efficient implementation of many-layered neural networks. Multiple layers are a critical requirement for solving even some simple logic problems, such as the parity problem, and so it is not surprising to find many such intermediate ("hidden") layers are needed for to solve complex problems in areas like image processing. The arrangement of the neurons and layers within a network is known as its *architecture* and there are many possibilities, with the choice dependent on the task. For instance, the operation of *convolution* over multidimensional grid-like data (such as images) has been found to be exceptionally efficient in recognizing shapes when applied multiple times, at various levels of abstraction; this currently constitutes the state of the art solution for most computer vision problems such as image segmentation and image recognition.

Temporal data, for instance in signal processing or text analysis, where the syntax and semantics of previous words have a large influence over what is to come next, may best be handled using **recurrent networks (RNNs)** to simulate the memory attribute of the human mind. These networks at each moment in time take as input their own previous outputs from previous timesteps, typically alongside some newly-presented external input. There are a number of variants of recurrent networks but one of the most important RNNs used nowadays is known as an **LSTM** (Long-Short Term Memory) net.

Although the phrase “deep learning” was originally applied only to nets with a very large number of layers, such as the previously mentioned convolutional nets, the phrase is also applied nowadays to networks of a more modest size, if a recent learning methodology is being used within the network. One example of this expanded use of "deep learning" is the (Tsantekidis et al., 2017) paper which will be discussed in the following section, that uses a LSTM network—LSTMs being among the methodologies associated with deep learning—even though in this case only a single, relatively small hidden layer is being used.

LSTM networks have however, independently of the size of the net which they are incorporated into, revolutionised the approach for time series forecasting in finance, which previously was done predominantly using sliding windows of a fixed length as inputs to a *multi-layer perceptron* (MLP), trained to predict price change. These sliding windows were used because former types of recurrent networks (prior to LSTMs) suffered from a problem of “vanishing gradients,” a very damaging consequence of a net's using as inputs its own outputs from different moment in time, which we will now briefly examine.

The problem arises fundamentally via the use of **backpropagation** to calculate the appropriate weight changes after each trial and error learning attempt. During training, when the result of a prediction is wrong, a step in the space of weights is taken so as to decrease the error of the network, and this weight change step, through the mechanism of derivation of the backpropagation weight update rule, contains a derivative value which is usually a fraction. Many layers—or equivalently, in an RNN, many passages of the same information through the network—multiply these derivative factors so that the magnitude of the associated weight changes becomes very small. Weights are less and less altered through training because the gradient is constantly diminishing until it completely disappears. This may lead to the earlier layers being completely left untrained, or, in the RNN context, to potentially important information from earlier times being ignored. This "vanishing gradient" problem within a multilayer network is partly solved by using a different activation function (using rectified linear units (RELU) instead of the tangent function or the logistic function). However, in the case of RNNs there is not only a potential problem within the network, if it contains many layers, but more importantly with the connections through time the network forms with itself by using outputs from its previous

states. A common way to visualise the problem is by “unrolling” the network, as in the figure to follow (fig. 2):

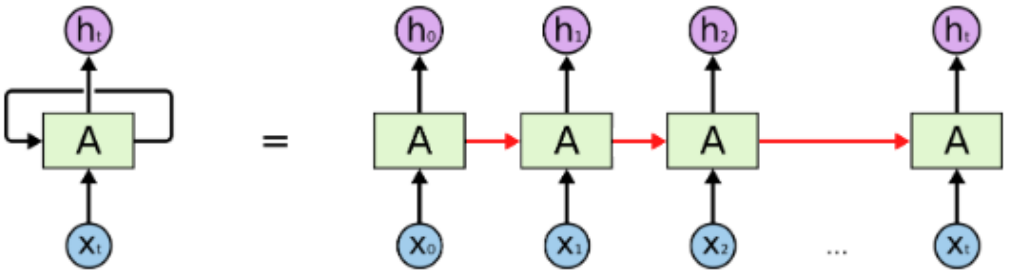


Figure 2 - An unrolled recurrent neural network, as described in the popular blog on LSTMS[1]

The gradient that is sent back through backpropagation gets altered considerably each time it passes one of the (red) links. Thus, a gradient with magnitude less that 1 would lead to vanishing weight changes (approaching closely to the 0 value), while in the case of a gradient of magnitude greater than 1, the weight changes would be amplified to infinity (exploding gradient). A vanishing gradient, such as would be found in an earlier-style RNN such as an *Elman net*, leads to only the most recent input activity being responded to, making the recurrent network insensitive to associations in the distant past i.e. not able to make correlations between desired outputs and events that happened further in the past.

By design, the *LSTM unit* (as illustrated on the next page in fig. 3) preserves the gradient as it is, without diminishing it, and does so by introducing the special pipe (shown in red) which gives unrestricted access to the entire history of states without altering the gradient. The functionality of LSTM cells is determined by implementation of different gates that oversee all the state changes, controlling the alteration of the weights themselves. The great breakthrough of LSTM design is that association of distant events in time with target outputs becomes possible using this architecture.

[1] http://colah.github.io/posts/2015-08-Understanding-LSTMs/ - accessed throughout (February) 2019

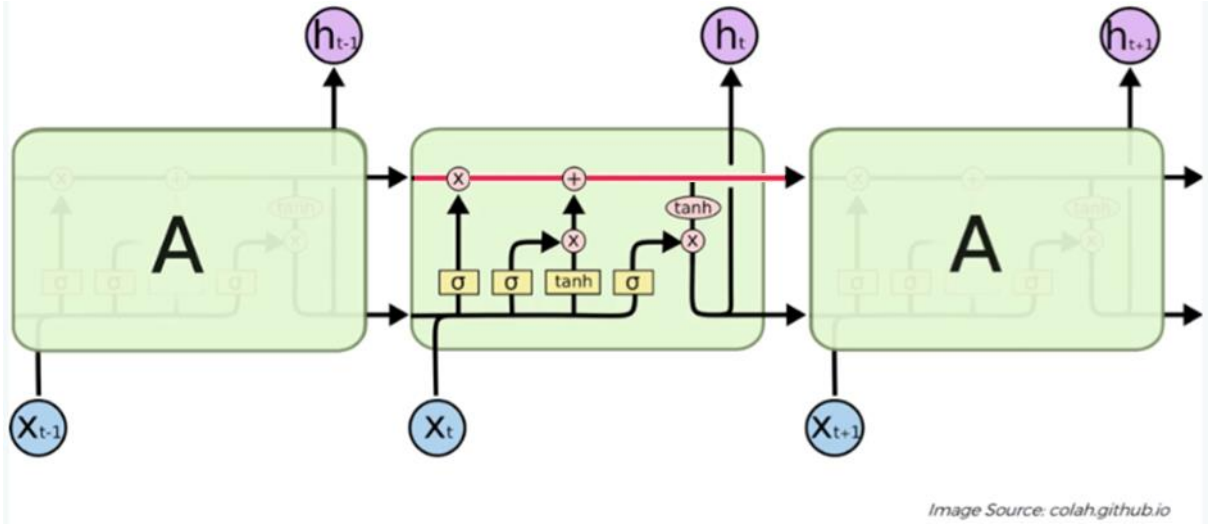


Figure 3 - LSTM layout, as cited by SuperDataScience[2]

2.3 Machine Learning Approaches to Price Prediction

A great deal of research has been done on market forecasting using artificial intelligence on the stock market and foreign exchange; however, it would seem a lot less research has been done into using this technology for predicting the price of cryptocurrencies, a fact which appears to be surprising, especially since this market is considered to be less efficient and therefore prone to exhibit more foreseeable behaviours in comparison to much older, regularized and legally empowered markets (though remaining hard to predict because of the granular noise and stochastic volatile nature of the market, which accounts for a continuing need for new approaches, especially on the machine learning side).

The papers that have been found on ML approaches to price prediction either use less efficient, older ML technologies on the Bitcoin (BTC) market, or using more modern algorithms on the stock exchange or other similar market; this literature discussion here will contain a mix of all the relevant studies on time-series prediction, regardless of the market it is applied to, as there is sufficient similarity between the BTC market and the others.

It is important to note the dynamic influence that technology development is having on the efficiency of a market: using a sliding-window MLP may have worked well in the past,

[2] https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-long-short-term-memory-lstm/ - accessed throughout (March) 2019

when few people were using the technique; however, as soon as there are considerable numbers of users of a technique such as this, the market changes and becomes more efficient, and so less profitable, in relation to that specific prediction solution. This effect not only accounts for the importance of discovering new technologies, in order to secure the advantage of using a currently-exclusive technology, but also raises a concern about interpreting the results of older works on this subject, which should be done with care.

As evidence to the soundness of this project's general approach—to use recurrent nets for financial price series prediction—(Dixon, 2017) has demonstrated that recurrent neural networks (RNNs) could help in high-frequency trading prediction, even though the type of network used (Elman net) is known not to be able to handle longer-term time dependencies well. Dixon's paper bases its work upon previous successes in predicting patterns in univariate financial time series using RNNs, and aims to solve the sequence classification problem of short-time market movements by using the limit order book and market orders within the Chicago Mercantile Exchange (CME). It argues that the RNN network has been able to efficiently capture the non-linear patterns of price change.

There are many approaches to predicting Bitcoin price movements. For example (Greaves and Au, 2015) studies the correlation between Bitcoin's transactional network topology and movements in price. Extracting transaction data from all the activity prior to 2013, the authors focus on using the layout of transactions to analyse the prediction capability of locality. They opt for a complex selection of input features computed for their statistical insight, while also making use of graph and set theory. This paper uses multiple sources for a more abundant dataset, and also performs non-trivial analyses over the Bitcoin network topology, yet results in a prediction accuracy hardly better than random. Another interesting idea was presented in (Groß et al., 2017) where the approach for predicting time-series was to use a "space-time" convolutional network in conjunction with a recurrent neural network. The authors were hoping to exploit the high performance of CNNs in analysing the spatial relationships in data. For this reason, they chose to translate the time-series data into a picture. Different moments in time are described by progressing coordinates within the picture; thus a sliding-window technique is being used to concentrate attention on the events in chronological order. The results of this paper are

stated to be better than the state of the art, although the most intriguing aspect is the ability of the space-time CNN to learn feature representations of the network at least similar in quality to those provided by a domain expert.

Another relevant paper was that of (Guo and Fantulin, 2018), which intended to compare various models to be used for Bitcoin price prediction. Here the authors have used hourly volatility data referring to the standard deviation and returns on the Bitcoin market. The dataset covered more than a year and was also linked with order book data. They extracted other statistical measurements as features to be used, and tested various mathematical models, with an evaluation based on root mean squared error (RMSE) and mean absolute error (MAE), though these may not be the most relevant indicators. Few of the models used machine learning, and the one that did, the Random Forest algorithm, could be considered unsuitable for time-series prediction. No trading strategy was implemented in this paper and part of the authors' conclusion was that using features from the order book did not help the prediction performance, which seems surprising.

Throughout the literature survey phase, it was noticed that a certain paper was quoted with significant frequency, and so was considered deserving of a deeper analysis: “Trading Bitcoin and Online Time Series Prediction” (Amjad and Shah, 2017). This study considers the poor performance exhibited by the traditional time series prediction tools such as the ARIMA mathematical model and also the lack of probabilistic interpretation presented by these methods. The paper intends to lay out a new framework for predicting and trading Bitcoin. The authors attribute the lesser efficiency of the Bitcoin market to stationarity and mixing of the market value time series, and argue that these are the very properties that should be exploited. Their work samples the Bitcoin price history every 10 seconds. While their dataset is large—requiring around eight months to train the models—they make use the order book only for calculating the Bitcoin price, and also do not use any of the newer algorithms which are known to be well-performing on time-series prediction. In their section of algorithm comparison they present the performance of the following algorithms (in order of increasing profit generated): ARIMA, Empirical Conditional (with an acceptable performance), and in the first place LDA, Logistic Regression and Random Forest algorithms, with extremely similar performances. Their ML solutions are without doubt

superior to the models they claim to be widely used (ARIMA), however it would be interesting to see how a modern neural network architecture, designed especially for the processing of temporal data, would perform using their setup.

2.4 Trend Prediction and the methods of Tsantekidis et al.

2.4.1 Novel approaches and characteristics

While most of the above-discussed work in the area of financial time series prediction is of interest, and some of the problems are tackled rather ingeniously, one questionable aspect is present in all of the approaches: the focus on one step ahead directional forecasting. This could mean that predictions may not be entirely practical due to their not accounting for sustained and longer term price movements.

With this in mind, one paper stood out as a promising foundation for this project. "Using Deep Learning to Detect Price Change Indications in Financial Markets" (Tsantekidis, et al., 2017) presents an interesting approach to the prediction problem, proposing a potential solution to the noise problem by use of smoothing. While this paper uses features extracted from the order book, instead of using the one step ahead prediction, the authors introduce two key parameters—*k* and *α*—that can tailor a forecast to a varied set of horizons and also smooth the inputs for a varying representation of past trends.

2.4.2 The smoothing procedure

The *k* variable acts as a lookahead: the algorithm takes, at each moment *t* in time, the price of the stock values for the following *k* timesteps in the future and averages it, then compares the resulting mean to the mean of prices of the previous *k* timesteps within the past, as follows,

$$difference = \frac{1}{k}\sum_{i=1}^{k} price_{t+i} - \frac{1}{k}\sum_{i=1}^{k} price_{t-i} \quad , \qquad (1)$$

in order to determine whether the price is likely to go up or down. Before considering the outcome, the difference obtained in the previous step is run through a threshold, α ,

$$label(midprice_n) = \begin{cases} (Up \quad | \quad difference > \alpha) \\ (Equal \quad | \quad \sqrt{difference^2} < \sqrt{\alpha^2}) \\ (Down \quad | \quad difference < -\alpha) \end{cases}, \qquad (2)$$

that checks whether the amount of movement is large enough to be taken into consideration; otherwise there is considered to be no movement.

2.4.3 Using LSTM units

Another notable aspect of this paper is that it also uses the highly efficient LSTM network that the rest of the studied literature has shown to be valuable. It is confirmed in the paper that thanks to the protection of the cell state provided by the gates within the LSTM design, the network is indeed able to efficiently correlate events distant in time, also solving the vanishing gradients problem. Regarding the architecture of the network, the authors come up with the experimental optimum range of 32 to 64 LSTM hidden units, claiming that more units would generate *overfitting* and that fewer would results in *underfitting* (resulting in a reduced accuracy); exactly 40 hidden units are chosen, though there is no justification within the paper for this, or for the authors' earlier statement about over- and underfitting.

2.4.4 Results obtained by Tsantekidis, et al.

Even though no trading strategy was attempted, it is reported that the algorithm performed encouragingly: 3 values of *k* were tested: 10, 20 and 30 with Cohen's Kappa values reported as 0.50, 0.43 and 0.41 respectively. Notably, however, the paper only presents results for these three values for *k*. This is a surprise: decreasing *k* from 30 to 10 improves performance, yet no values smaller than 10 are considered. The motives of the authors in not presenting results for smaller *k* values are unclear. While there is a chance a smaller *k* would perform better, it is possible that the optimum value might be discovered to be *k*=1. This last would be a problem, as for this value there is effectively no smoothing, and thus the paper's central argument—that smoothing helps with prediction—would be invalidated.

Chapter 3 - Design and Methodology

3.1 Data Representation

It is clear from the Literature Review section that many inputs and features can be fed into a prediction system. A number of choices could be taken here: 1) to gather tick data (every single transaction ever made) or use a fixed frequency sampler of the intended parameters; 2) to consider only the Bitcoin price itself, or track the transactions to discover the network topology; 3) to use limit order book data to gain enhanced insight into the state of the market at each timestep, or to use it for a simple calculation of volatility. Out of all these variants, the work here chooses to gather data from the limit order book, reconstructed from high frequency snapshots of the market. The choice of using order book data was made because of the more in-depth representation of the market it provides – by having the volumes associated with the trading positions at each step, a better understanding of the market dynamic is allowed – a fact the explored literature seemed to underline as an essential element in a better pattern recognition scheme, especially as opposed to the less informative overall price value for every tick of the market.

3.2 Data Acquisition

One of the important reasons for choosing cryptocurrency markets, as opposed to Foreign Exchange or the Stock Market, aside from the market itself being more volatile and inefficient (leading to increased predictability), was the accessibility of cryptocurrency data, which is far more easily acquired than that of the traditional markets which require large payments to various accredited brokers for receiving such a detailed dataset.

Initially, this project intended to use limit order book data by the tick (every single movement recorded on the market) from 10 days of Bitcoin activity. However, a compromise had to be made in terms of dataset size because of the increase in complexity that this scale of data acquisition would imply. Thus, the dataset used has the Order Book snapshot for 160,000 timesteps, over the 10 days to be considered – this approximates to one snapshot of the market every 10 seconds.

The data used in this project originate from the CoinBase Pro and Bitfinex platforms.

3.3 Feature Extraction & Preprocessing

Having chosen a similar data representation and temporality to that used in the foundation paper of Tsantekidis, et al., whose results seemed good, the intuitive decision was to try a similar approach here for feature extraction: 40 parameters have thus been selected for an in-depth representation of the market's structure at each moment in time (each timestep): first 10 of the lowest bid prices and first 10 of the highest ask prices on offer, together with their corresponding volumes per trade position.

The 40 parameters per timestep then undergo a normalization process (standardization) by subtracting the mean from each value and then dividing by the standard deviation:

$$x_{normalized} = \frac{x - \bar{x}}{stdDev(\bar{x})} \quad (3)$$

The values are thus scaled to an acceptable range for the neural network expected input (as was also done in (Tsantekidis, et al., 2017)).

3.4 Target Label Construction: *k* & *α*

The effective price at a specific moment in time can be derived from the dataset by calculating the arithmetic mean of the highest ask and lowest bid values. These price values are used for analysing the evolution of the market and computing the target labels.

There are 3 labels used here: *Up* (represented by the numerical value 2), *Down* (represented by 0) and *Equal* (represented by 1). The two parameters introduced in (Tsantekidis, et al., 2017), *k* and *α*, would decide how the labels are distributed among the three categories. In order to construct the labels, for a given timestep *t*, the average of the price values associated with the *previous k* timesteps with the equivalent average for the *following k* timesteps*,* using equation (1), presented in Chapter 2. If the absolute difference obtained is higher than the value of *α,* then we can say the fluctuation is notable and assign a label accordingly (Up or Down); otherwise, we consider the movement to be too little to take into consideration and assign (Equal), as described in equation (2).

k could be thought of as a "lookahead": its value decides the number of prices before and after the current step that have to be compared, while the *α* parameter is the threshold that must be surpassed for the movement to be considered as relevant.

3.4.1 Exploring α

The value of the threshold parameter *α* is unknown in the paper Tsantekidis, et al., and there is no indication as to the means by which a value should be chosen. Three ideas regarding the choice of *α* arose after considering the influences it might have on the rest of the system, along with the different aims a trading setup might demand:

1) *α* could be chosen in relation to the transaction costs of a trading platform, once we have set up a successful trading strategy, being the minimum value we would be aiming to gain from the investment, such that the maintenance costs of the transaction could be covered.
2) *α* could be chosen as the threshold value which gives a balanced three-way split between the up, down and equal categories that we use as target labels.
3) *α* could be optimized to give the largest profit.

3.4.2 Exploring *k*

As it has been discussed in Chapter 2, (Tsantekidis, et al., 2017) only consider three values for the smoothing parameter *k*, 10, 20 and 30, which display a decreasing performance in relation to Cohen's Kappa. The importance of exploring smaller values of *k* was emphasised in Chapter 2, due in part to the possibility of discovering the optimum value was $k=1$, which would invalidating the philosophy behind smoothing (implying an optimum lookahead value of $k > 1$). There is also a very high possibility that $k=10$ is not the optimum value for classification, but that some value $1 < k < 10$ might prove better. Either way, this project considers the presented arguments to be both interesting and challenging enough as to investigate the performance for lower values of the *k* parameter.

The higher the value of *k*, the more the focus shifts to the macro scale of movements and investigation would be expected to provide insight about long term movements, with a better chance of recognising higher amplitude patterns, extended over a longer period of time (and otherwise non-observable). On the other hand, it is also true that there will be a

value for *k* above which the prediction will be greatly degraded. Therefore, common sense implies the optimum value of *k* would be, for either trend classification or profit, somewhere between the two—higher than 1, performing better as *k* increases, peaking somewhere very likely below *k*=10, and then steadily going down again.

3.5 Training / Testing / Validation Data Split

This project will aim to follow the methodology of (Tsantekidis, et al., 2017) for ease of comparison, since a similar dataset will also be used (10 days of transaction data); therefore the split used here, for training/validation data relative to testing data, will also be 7:3.

3.6 Neural Network Architecture: LSTM Design Decisions

Designing any neural network implies analysing the input and expected results such that the proposed architecture performs the task most effectively. As stated in Chapter 2, different operations within different network layers have an impact on the program's learning and generalisation abilities. Although specialized architectures may put together lots of layers with different functions, creating a complex path for the processed data to go through, the purpose of this paper is to investigate the use of LSTM neural networks in cryptocurrency markets (Bitcoin) at a conceptual level; having an over-complicated layout could alter the clarity of our results. Therefore, the proposed network architecture contains only two layers: the LSTM itself with a chosen number of hidden units and a fully connected layer with *softmax* activation and 3 neurons for the 3 considered classes.

It is common knowledge in the field that in order to avoid *overfitting* on the training data provided (picking up particularities of the given dataset that will not generalize to other datasets which may be used for testing or in practice) it is best to restrict the number of parameters within the network in relation to the number of instances used for training—overfitting has the best chance of being avoided if the number of parameters is considerably smaller than the number of samples used (1/10th of the number is a common heuristic rule). This principle will guide the choice of the number of hidden units.

3.6.2 Hidden units

(Tsantekidis, et al., 2017) had a dataset of 5 stocks over 10 days; the duration is the same to the one used in this project, though here only Bitcoin data are being used. As described in the previous chapter they proposed a number of LSTM hidden units between 32 and 64 and claim that their experiments used 40 hidden units. The present project intends to experiment with a range of choices before settling on a number of hidden units appropriate for the cryptocurrency dataset size.

3.7 Performance Measures - Evaluation

3.7.1 – Measures of classification performance

Traditionally, when training machine learning algorithms, the standard accuracy metric, given simply by the proportion of correct predictions over total predictions, is most often used. However from a classification perspective, considering only accuracy could be rather misleading, especially while dealing with imbalanced data sets, as high accuracy values can be obtained by over-assigning to the majority class. This problem is even greater when the imbalanced data contain more than 2 label categories, as here.

This is the reason why in this project, a **confusion matrix** was regarded as the obvious choice for assessing the results. Also known by the name of *error matrix*, it is used in supervised machine learning as a tool of performance visualization in the form of a two-dimensional table which shows frequency distributions per each class.

Actual / *Predicted*	*UP*	*EQUAL*	*DOWN*
UP	**UP predicted as UP**	EQ predicted as UP	DOWN predicted as UP
EQUAL	UP predicted as EQ	**EQ predicted as EQ**	DOWN predicted as EQ
DOWN	UP predicted as DOWN	EQ predicted as DOWN	**DOWN predicted as DOWN**

The table's layout associates the ***predicted classes*** by *lines* and ***actual classes*** by *columns*. Adding the numbers from an entire row produces the total number of predicted instances of the class designated by the row, while similarly, adding up the numbers from a column gives the total number of actual instances (of the class) there were in the dataset.

Correctly predicted instances are located on the principal diagonal of this matrix, with the sum of the values on the principal diagonal equating to the traditional *accuracy* metric.

The matrix is then used here to compute the **Matthews Correlation Coefficient (MCC)** as a classification performance metric. The MCC has been chosen (over Cohen's Kappa used by the Tsantekidis et al. paper) mainly for its n-class variant:

$$M_{cc} = \frac{p_a n_a - u_a o_a}{\sqrt{(p_a + u_a)(p_a + o_a)(n_a + u_a)(n_a + o_a)}} \quad (4)$$

MCC formula for C-class problem, defined for class *a*, where:

- p_a = number of cases of class *a* correctly assigned to that class
- n_a = number of cases correctly predicted to be *not* class 'a'
- o_a = number of cases predicted to be class *a* which were *not* of this type (**false positives**)
- u_a = number of cases which *should* have been predicted to be class 'a' which were incorrectly assigned to one of the other classes (**false negatives**)

This formula gives an MCC value for each class separately. In order to provide a single-valued metric MCC value for comparison of the results from different experimental conditions (e.g. different values of the smoothing parameter *k*), the *arithmetic mean* of the 3 MCC values is computed for the 3 classes (Up, Down, Equal).

3.7.2 – Measures of profitability

Classification metrics (MCC, Cohen's Kappa, or even accuracy) do not relate directly to profit; maximum profit would not necessarily be where a classification metric peaks, as these metrics only take into consideration the direction of a movement of the price, not its amplitude. However the profit is highly dependent on the size of the movement. For this reason, profit-focused testing should also be performed.

Assuming that the methodology will perform well in relation to classification, we consider here a trading strategy based on the algorithm that we will be constructing. Assuming also we are concerned not only about profit but variance (reliability of the

algorithm within a practical trading approach) the testing of such a strategy should involve, as well as a standard measure of profit, a calculation of the Sharpe ratio, defined below:

$$Sharpe\ Ratio = \frac{R_p - R_f}{\sigma_p} \tag{5}$$

R_p = return of portfolio

R_f = risk-free rate

σ_p = standard deviation of the portfolio's excess return

Chapter 4 – Results

4.1 Initial Results

The results to be presented later in this chapter were based on an amended dataset, as the early work demonstrated discrepancies that were eventually traced to an anomaly in the initial dataset, such that part of the data was, in fact, missing. It appears that the server being used to record the market suffered a RAM malfunction, which locked the threads of execution for a period of time, resulting in partial data loss. This issue generated an anomaly within the dataset around one quarter into the observation period in place of the missing data. (see the fig. 4 below). This inconvenience has actually proven to be useful as it pointed

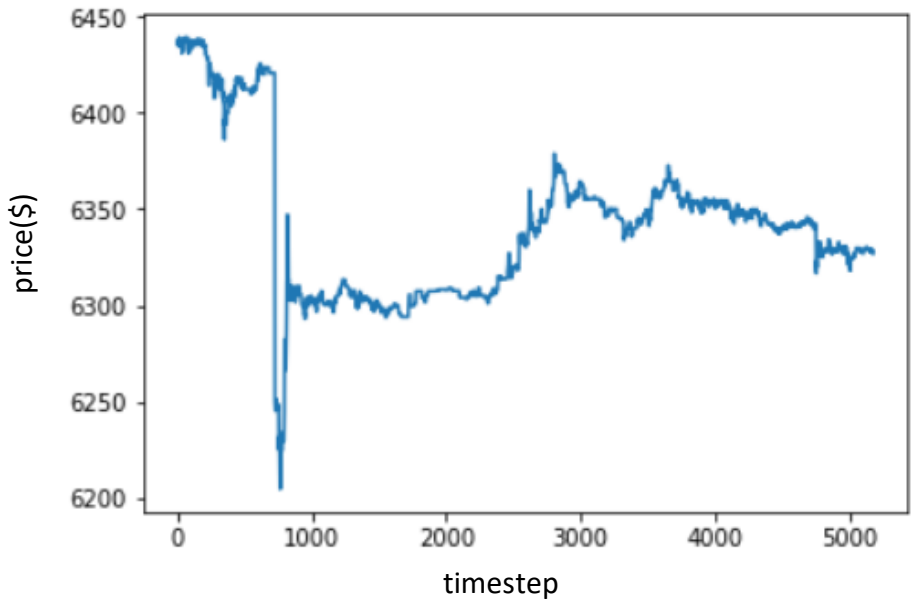


Figure 4 – First dataset's Bitcoin prices during the time period considered (timesteps of 10s used here and throughout this work)

out that instantaneous inconsistencies are not a very serious problem when basing predictions on trends (regardless of the movement amplitude); if only one of the many short-time-period differences in the dataset is excessively large this does not influence the overall reading to a considerable extent.

However, there was one positive outcome from this initial setback, which was that having sufficiently varied training data, in order for the algorithm to pick up all possibly relevant patterns in the market, was discovered to be more important, even, than the presence of the anomaly itself (fig. 4). This finding drew attention to the timeframe of 10

days being not as sufficient as initially believed, and further decisions, especially regarding the train/test/validation split, were influenced by this observation.

4.2 Selection of Second Dataset

As stated in the previous section, a major concern was to ensure sufficient variation in the training data, as well as making sure the data contained no erroneous entries. A second dataset was therefore obtained, that was not only checked to be free of anomalies, but was also much larger. The completely new data set, to be used for training, validation and testing for the rest of this project is shown in fig. 5 below.

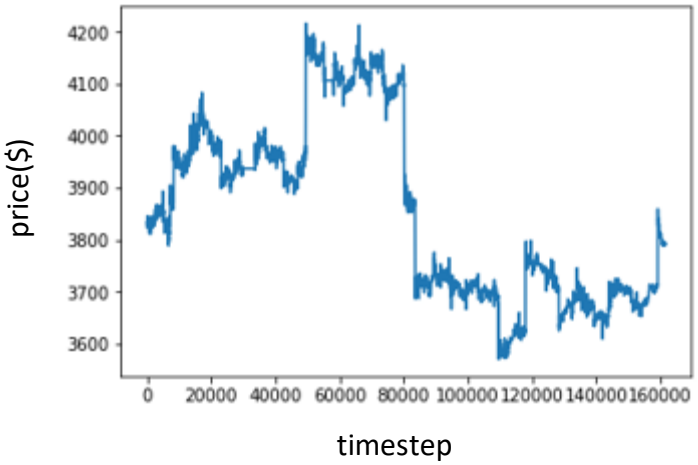


Figure 5 – Second dataset's Bitcoin prices during another 10 days' time period considered (thus, a different set of timesteps)

4.3 Train / Test / Validation data split

Although we had initially aimed for a 7:3 distribution among the train/test data, experiments showed that extending the training set with an extra 10% to cover more data greatly improved the classification accuracy, so the final split of the dataset was **80% training** (from which 20% was used as **validation**) and **20%** used as **testing** set, which the model never sees until evaluation.

4.4 Network hyperparameter selection

Hyperparameter selection was done in relation to the metrics supplied within the TensorFlow environment, loss (to drive the training) and accuracy (to monitor performance). The loss function used was “categorical crossentropy”.

One choice to be made was the number of training epochs; a value of 100 was chosen because it seemed the most substantial improvements in accuracy and loss function had already taken place by the 100th epoch.

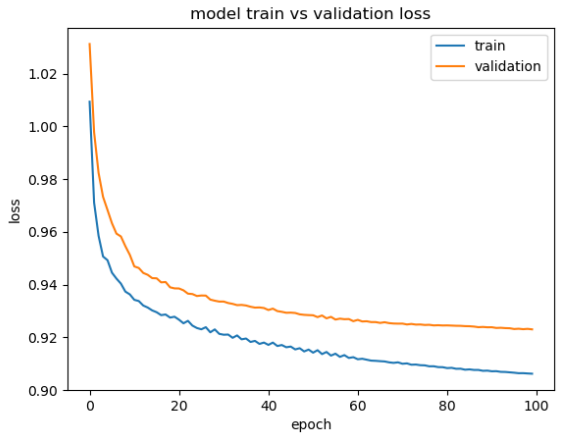


Figure 6 – loss evolution for one of the models

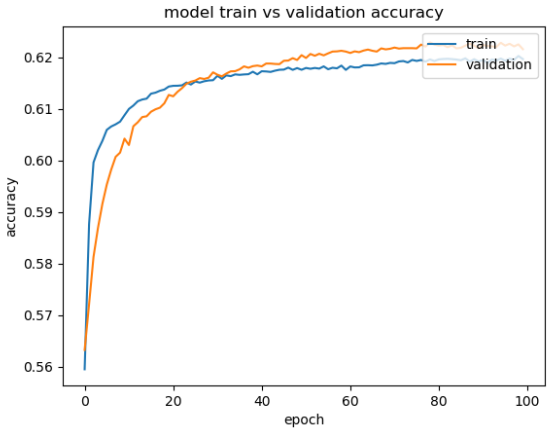


Figure 7 – accuracy evolution for one model

The number of hidden units in the LSTM had to be experimentally decided, taking into account the data differences between this project and (Tsantekidis et al., 2017): both pieces of work use limit order book data, but here the Bitcoin market is used, while in the cited paper various different stocks (from NASDAQ) are analysed. The number of markets (several stocks, one cryptocurrency) is not the main difference, which is the frequency of movements that occur on different markets: Bitcoin has a considerably lower number of transactions per hour than regular stocks. For these reasons, as stated in Methodology, we were expecting to need to choose fewer LSTM, units in order to avoid overfitting. The number of hidden units appropriate to our dataset size was found to be 15.

Mini-batching was used in these experiments, and another empirically established value was the batch size of 64, attempts to change it to either higher or lower powers of 2 resulting in greatly reduced classification performance.

4.5 Training label construction: *k* & *α*

4.5.1 Unexplored implications of the two parameters

For a fixed *k* lookahead, an *α* value of 0 would mean that *any* movement would be considered relevant since there would be no threshold. Therefore at the label construction stage such a setting would push as many readings as possible to the UP or DOWN (as opposed to EQUAL) label category; similarly, an increasing value of *α* would mean putting more and more instances into the EQUAL category since a higher value of *α* would have less chance of being met, such that over a certain limit, the value of the relevance threshold would become too high to ever be reached by movements on the market.

On the other hand, *k* acts to modify the prediction range. The higher *k* goes, the more timesteps are being averaged and the difference between past and present has increasing chances of being larger, therefore the movements' labels tend to move to the extremes (UP and DOWN categories). Conversely, when *k* goes down, the differences tend to be less noticeable for moments closer in time, so there will be a higher chance the movement is almost null, thus again moving more instances to the EQUAL label category.

As stated in the Methodology chapter, there is no explanation within (Tsantekidis et al., 2017*)* as to how *α* should be chosen. For example the paper could have used a single value of *α* for all the *k* ones considered, chosen to balance the middle value of k (in their case k=20 since they only tested 10, 20 and 30). This was a reasonable initial assumption. However, when we tried to select a single value of *α* chosen to accurately balance the labels for the middle value of *k*, it was observed that for the *k* values closer to the extremes, the labels became greatly unbalanced over the three categories, thus increasing the difficulty of prediction and making the use of accuracy as a performance measure unreliable. In order to make the classification problem reasonable, and also preserve the same label proportions

so that we could make a sound comparison across different values of *k*, we have chosen a value of α for each value of *k* that distributes the labels evenly into the three categories.

Empirically, it has been established that the following combinations of *k* and *α* values gave the most balanced distribution of labels:

k	*α*
1	0.00
2	0.00
3	0.00
5	0.05
10	0.20
20	0.50
30	0.75
45	1.00
60	1.30
120	2.00

Table 1 - *k* & *α* optimal pairs

4.5.2 Further investigation of the role of the *k* parameter

As discussed in the Methodology chapter, the reasoning of the Tsantekidis et al. paper was quite unintuitive in that they only showed results starting with $k = 10$, then jumped to 20, then 30—this is odd, as the accuracy was decreasing during this process. Intuition suggests that if your starting point is on a slope, you should try to find the peak, which would here imply investigating values $k < 10$. In addition the results of the paper left open the possibility that the best value of *k* might be $k=1$, which would have undermined the paper's central proposal that smoothing (using $k > 1$) benefitted prediction and/or profit. These several observations therefore prompt us to check the values of *k* below 10.

The figure below shows average Matthews Correlation Coefficient (MCC) values over 10 runs with different weight initialisations, together with standard deviations (as error

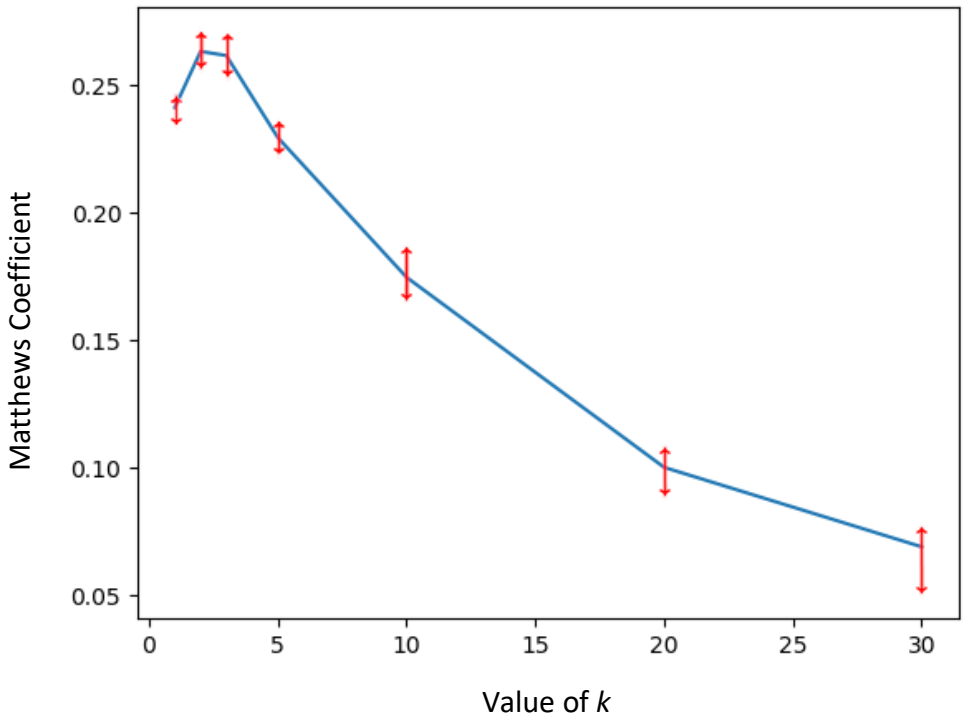


Figure 8 – Variation of MCC with lookahead parameter *k*

bars) as a function of the parameter *k*. The *α* value for each *k* chosen from Table 1 The choice of 10 runs for averaging was made by increasing the number of runs until the average MCC of each half of the set of runs was similar: 10 was found to be a sufficient number in this case, with both an acceptable margin of two decimal points in MCC, as well as a reasonable running time of about 5 hours for 10 runs of the algorithm with a single *k* & *α* selection (using a TensorFlow backend graphically accelerated on a machine with 4GB dedicated GPU—Nvidia GTX 950M).

It is plain from Figure 4, above, that 10 is not in fact the optimal value of *k*: while performance decreases as *k* increases to larger values, as was also found by (Tsantekidis et al., 2017) the optimal MCC value is for *k*=2, closely followed by *k*=3. This result is both reassuring—the optimal performance was *not* for *k*=1, thus validating the Tsantekidis et al. proposal of smoothing in order to better predict price trends—and interesting, in that the

best value differed substantially to the value of *k*=10 favoured by Tsantekidis et al. (though it is possible the optimal value could be dataset dependent). At any rate, figure 8 gives a good basis for hoping that a profitable trading strategy can now be developed.

4.6 Trading Strategy Implementation

Given that investigating and optimizing the methodology within the paper on which this project was based led to satisfactory results, and also that the project was well within its time frame, an attempt at implementing a trading strategy became possible.

Having established that price movements could be quite well predicted, we will be now trying to see how the prediction model would perform while linked with a trading strategy designed especially for it. The process of this section aims to gain an insight into how reliable and useful our approach could be in practice. We choose at this point to adopt a long-only trading strategy; clearly in future work shorting could also be considered (though this is not straightforward for cryptocurrencies). We also choose to ignore the transaction costs, which are low and, unlike conventional currencies, the same for any amount of Bitcoin traded.

4.6.1 Variable *k* lookahead, 3 steps trading

The initial approach proposed is derived from classic, straightforward, one step ahead trading: at every step, make a prediction for the following *k* steps; if that prediction is an upward movement, then buy, otherwise do nothing. If a position was bought, then regardless of the price value, once reaching the timestep for which the movement was previously predicted by the algorithm, we sell immediately. Since our top classification performance was for *k*=3, it seemed a fair decision, at least initially, to elaborate a trading tailored especially for this value, as follows (see fig. 9 on next page).

```
for each timestep t:

    predict trend at t+3 timestep (that means running the
algorithm for k=3)

    if the predicted trend is UP, then commit to a
position, otherwise do nothing
```

Figure 9 – 3 steps trading algorithm

This strategy implies that, at any moment in time, the maximum number of positions that can be held is 3. Since the purpose of this experiment is to gain a basic insight into the model's profitability, we will assume buying to mean simply purchasing an entire Bitcoin at every trading opportunity, for the market price. Since we could only hold 3 possible positions at one time, and since also the price of Bitcoin throughout our chosen period was around $3000(US dollars), it follows that the overall investment for this experiment will be somewhere below $10,000 (relevant for the scale of profit measurement).

4.6.2 Trading strategy results (profit)

When implementing the above-described trading strategy on top of the proposed prediction model for *k*=3, we get the following raw profit (fig. 10 & 11) over the testing period (20% of the dataset implying here that the testing period would be 2 days):

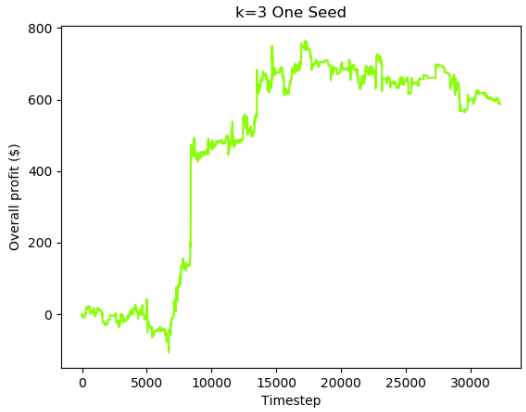


Figure 10 – profit for one seed (k=3)

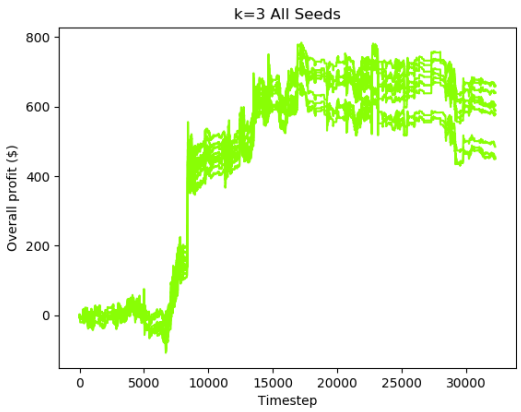


Figure 11 – profit for all seeds (k=3)

The top profit value for the seed illustrated in fig. 7 (left) is $764 (to be compared with a starting sum of about $10,000, hence around 8%), and the final value (when the testing period stops) for this seed is $587 (around 6%), which is an excellent result for two days of trading (though we note that transaction costs are not at the moment included). It thus seems that even with a rudimentary approach to trading, the prediction model proposed is capable of making profit. Of course it might have been that this one seed was 'lucky,' and for this reason the profit curves for all 10 have been plotted on right hand side of the figure above: it can be seen that the behaviours and final profits do not vary greatly, and that no run results in excessive variance or losses. This is another excellent and encouraging result.

Given the performance of this initial *k*=3 test, it was interesting to see how the trading strategy outlined in fig. 9 performs for other values of *k*, even though the trading method was originally designed for the *k*=3 value.

The same experiment was performed over all the 10 seeds for all the values of *k* (and their relevant *α*'s). The figure below (fig. 12) shows how one single seed behaves for different values of *k*:

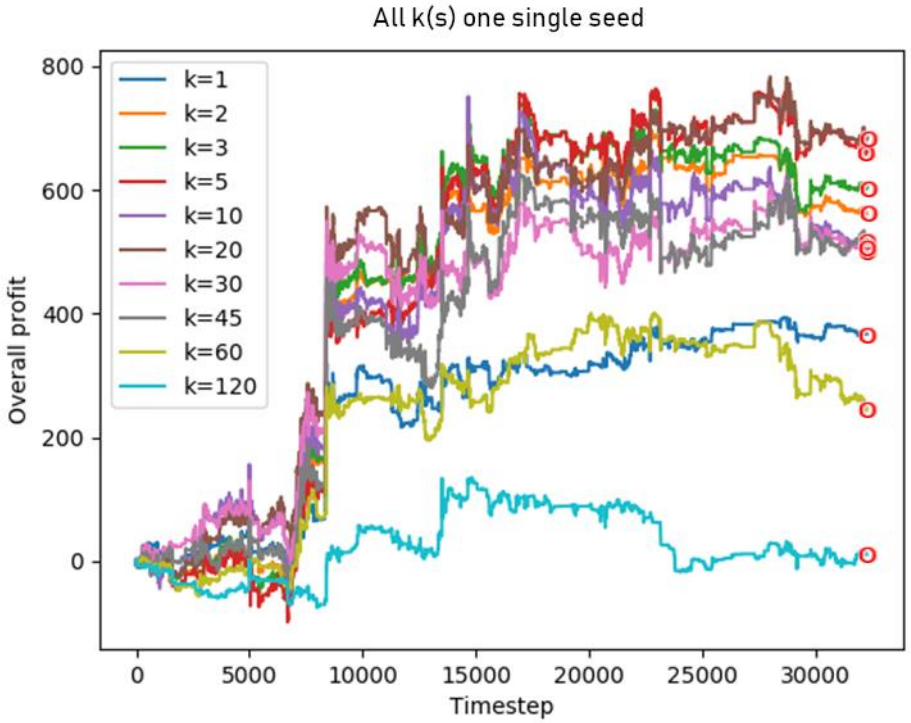


Figure 12 – profits for different *k* values for one single seed

It can be seen that profitability degrades quickly beyond *k*=30, this being a possible reason why results for these larger values were not shown in (Tsantekidis et al., 2017). But which value of *k* is best? It may not be *k*=3, the best value in relation to classification performance, since, as noted previously, the ability to classify price movements well does not translate automatically into high profits.

As the plot for each *k* had a slightly different look for each seed, our approach to finding the best *k* was to smooth the results by taking the final profit for each *k* (red circle, in the fig. 12 above), for each seed, and average it with the profits for the same *k* from the other seeds, and plot the average as a function of *k*.

The plot which resulted is the following:

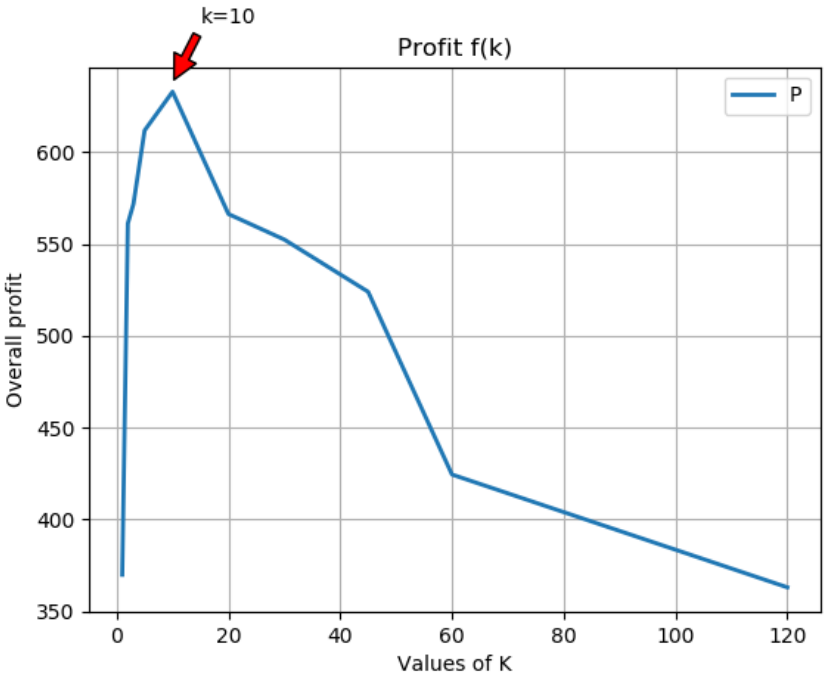


Figure 13 – final profit value for each *k* averaged over all the seeds

This plot shows that even when using a trading strategy designed for the optimum classification performance *k* value of *k*=3, a lookahead of higher value—*k*=10—that smooths away more of the noise is preferable, giving a highest final profit of around $700 on average. The results summarised in fig. 13 strengthen the finding that looking further into the future (and backward into the past) as opposed to doing simple, one-step prediction, is a methodology better able to handle noisy, irrelevant fluctuations, and provide useful predictions that would be profitable in practice. However fig. 13 also show in its right hand portion that attempting to smooth *too* much—equivalent here to trying to predict too far into the future—is not feasible. (Though notably profits for the very large lookahead of *k*=120 are only a little worse than for the 'classic' value of *k*=1, which achieves just above $350, underscoring the inadvisability of one-step prediction.)

The choice of 3 trading positions might be questioned in the cases for which *k* was not also 3, with the suspicion, for example, that a larger number of trading steps for larger *k* values might be superior. When this theory was put to the test, however, the outcome was

otherwise: the figures below (fig. 14 & fig. 15) show results for the combination *k*=10 with 10 trading positions opened (on the right hand side) compared to *k*=10 with 3 trading positions opened (on the left hand side). *k*=10 with 3 trading positions is clearly better.

(Since a maximum of 10 trading positions are allowed to be held in the alternative trading strategy for *k*=10, the investment would be approximatively 3 times larger than for the *k*=10 with 3 steps trading scenario, with profit multiplied likewise. For this reason the highest profit for the 10 steps trading strategy, on the right, should be divided by 3.)

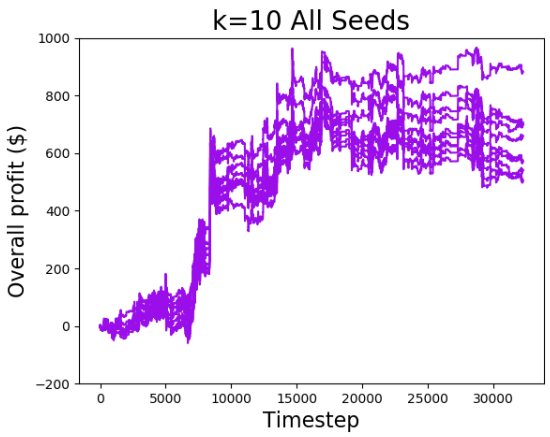


Figure 14 – profit for 3 steps trading strategy

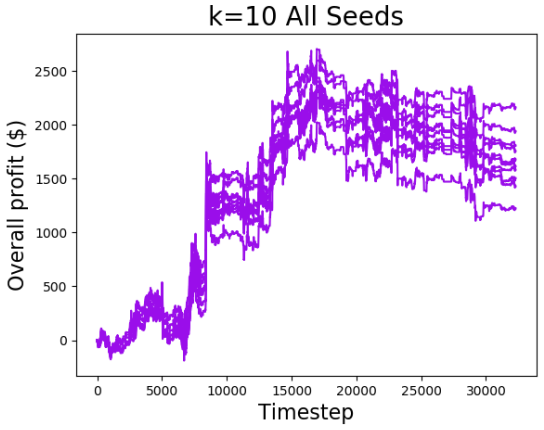


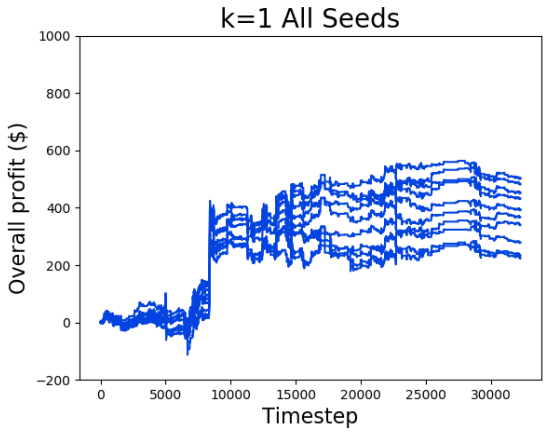
Figure 15 – profit for 10 steps trading strategy (different scale)

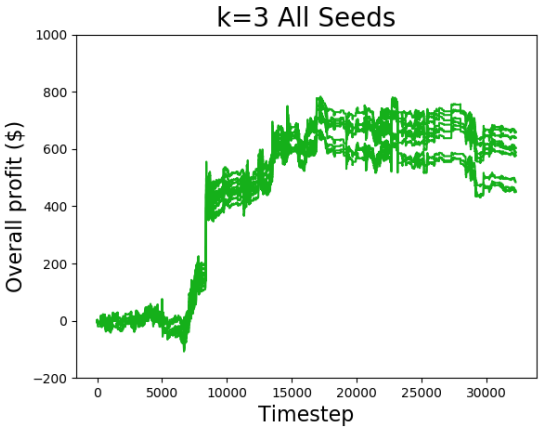
In this case, the 3 steps trading strategy has still managed to hold its ground. However why would exactly 3 be the optimum number of positions to act upon? Might there be another combination of *k* lookahead and trading positions that would give the highest profit of all? We plan to address this question later on, but for now, further investigation will be carried out on the 3 steps strategy, with the aim of better understanding the results.

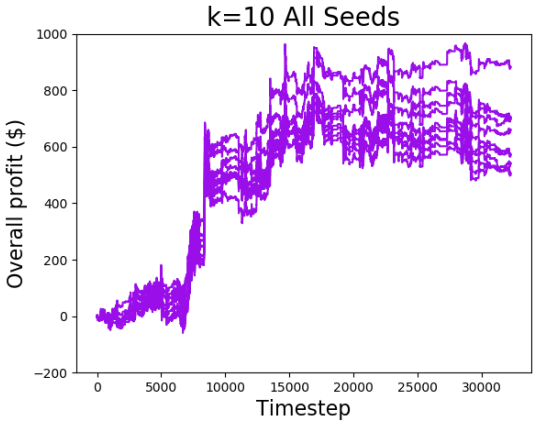
4.6.3 Trading strategy results (feasibility)

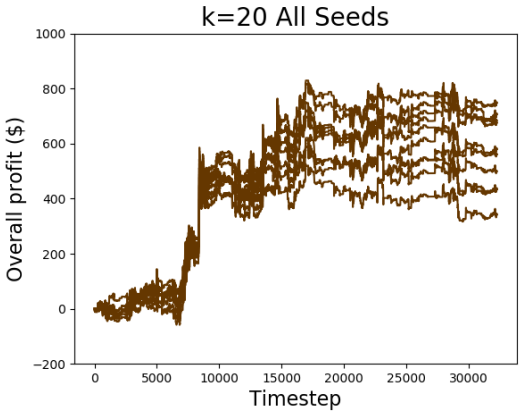
It has been shown that the highest profit is for *k*=10. However financial investment needs also to consider **risk**. The evaluation of the investment risk will be assessed using the Sharpe Ratio described in the Methodology. However another, graphical, indication of the reliability of the trading strategy would be to look at the variability in performance for the 10 different

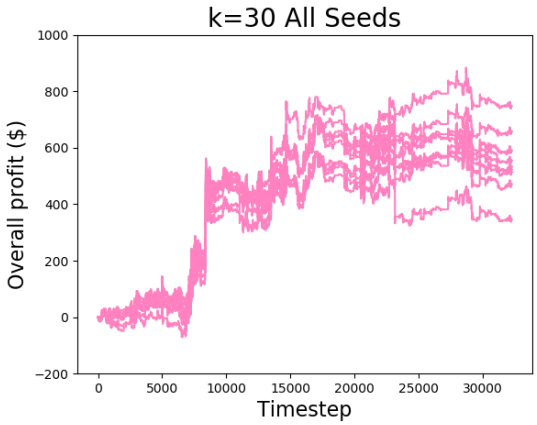
seeds for each *k*, which is done in the figures to follow. We might expect that higher reliability of prediction—a lessening of this variability—would correspond to an increasing MCC (better classification performance). But is this so?

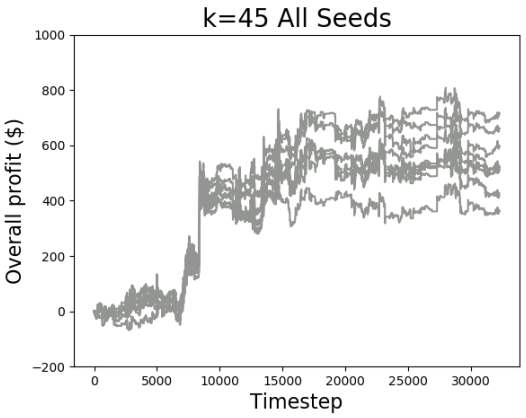












(see next page for remaining figures)

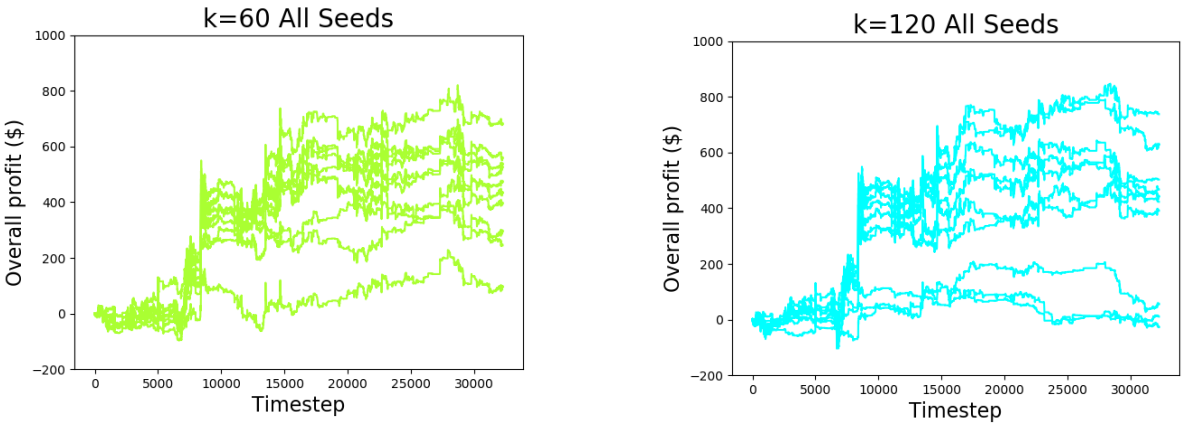


Figure 16 – profit for 3 steps trading strategy (for 8 values of *k* over all seeds)

In fact, as can be seen in the preceding figures, it is in fact true that the highest classification performance, for *k*=3, correlates to the 'tightest' spread of profits. This lack of variation (with seed) of final profit might be appealing to an investor concerned about the possibility of losing money, such that they might prefer a *k*=3 model over the on average more profitable *k*=10. In the end it would be down to the investor's appetite for risk.

4.6.4 Trading strategy results (Sharpe Ratio)

A more formal approach to risk is to calculate the Sharpe Ratio, described in section 3.7.2 and defined by formula (5). The *portfolio return* will be extracted from the above-mentioned simulations, specifically from the seed with the highest profit for *k*=10, by dividing the profit by the monetary input over the studied period,

$$R_p = \frac{profit}{investment} = \left(\frac{726\$}{9428\$}\right) = 7.7\%\,, \qquad (6)$$

where the investment for this period will be considered to be the maximum amount of funds to be consumed at a time. The R_f (risk-free return) has been chosen to be the return of a UK governmental bond taken from the Bloomberg website[3] of 0.70% per year, which we divide by 182 to account for the trading period of 2 days considered here, the tested period of our proposed trading strategy:

[3] https://www.bloomberg.com/markets/rates-bonds/government-bonds/uk - accessed in March 2019).

$$R_f = \frac{yearly\ yield\ (UK\ bond)}{182\ days} = 0.00003, \qquad \text{(for 2 days)} \tag{7}$$

Using the following notations

p_t – profit at timestep t (with T being the final timestep)

$x_t = p_t - p_{t-1}$

we define R_p more formally to be

$$R_p = \sum_{t=1}^{T} p_t - p_{t-1} \tag{8}$$

and the mean of x to be

$$\overline{x} = \frac{1}{T} \sum_{t=1}^{T} x_t \tag{9}$$

The standard deviation can be calculated by the following formula

$$\sigma_p = \sqrt{\frac{\sum_{t=1}^{T} (x_t - \overline{x})^2}{T-1}} \tag{10}$$

such that, finally, the Sharpe Ratio for the k=10, 3 steps trading strategy, over the testing period considered, can be calculated to be

$$Sharpe\ Ratio = \frac{R_p - R_f}{\sigma_p} = \frac{0.077 - 0.000038}{1.40} = 0.06 \qquad (11)$$

While any positive value of this coefficient demonstrates a trading strategy is not absurd, a value of 0.06 is considered to be rather small. However, there is one likely contributory explanation: the run-in period of the algorithm (see fig. 17), which takes roughly until the 7500th timestep of the testing period for all observed runs, would not be taken into consideration by a final, live, version of the algorithm, that was being used on real data.

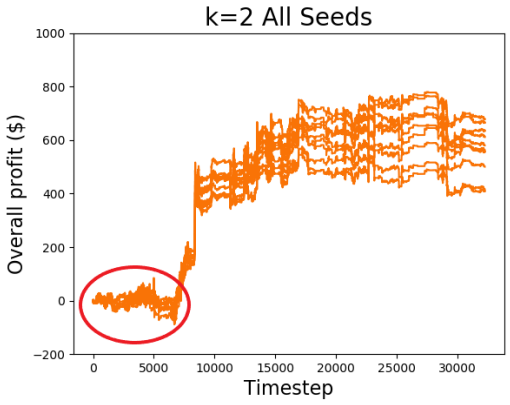


Figure 17 – run-in period not relevant

One should not be allowed to trade until the algorithm gets enough data to process the incoming price patterns, a principle which could be easily coded up in future.

Chapter 5 – Discussion and Conclusions

5.1 Discussion

The preliminary strategy explored here, which was intended to explore the potential of using a smoothing method for trading, has overall been very promising. Investigation validated the assertions of the paper on which the research of this project was based, that of (Tsantekidis et al., 2017), in fact doing so more persuasively than the paper did itself, being willing here to investigate a broader range of lookahead (smoothing) parameters *k*, and looking at results in terms of both classification performance and profit.

The trading strategy based on the prediction model was successful in being able to decide in all cases (for all the seeds and all *k* values) when exactly it should commit to a position, right before the largest variations of Bitcoin price when it had the biggest potential for profit. However as was seen at the end of the last chapter there was a high standard deviation of profit values over time, the reason for the low Sharpe ratio. While this could be to some degree alleviated by not trading until the LSTM had completed its 'run-in' period, to some degree the high variance may be an unavoidable a part of the methodology, in which case it would be down to investors to decide how much risk they would accept.

5.2 Future Work

It would be worthwhile to investigate more efficient and perhaps more complex approaches to trading strategies, including the possibility of adopting short positions. Transaction costs (while they are very small for Bitcoin) should ideally also be included. There is also the concern raised at the end of section 4.4.3 as to whether the multi-step trading strategy really does work best for 3 steps, as only two values for this parameter (3 and 10) have so far been tried . A good plan, therefore, for continuing this project would be to perform a grid search for the best combination of *k* lookahead and the number of positions to be held within the trading strategy.

On the hardware and software side, a possible solution to the server malfunction which generated a data anomaly affecting early results might be to change the system

architecture to one similar to the 'Redis' broker for efficient queue handling. Also, a plain SQL database has been used here; opting for a KDB related system could be beneficial.

5.3 Concluding Thoughts

As the current work has demonstrated, the use of neural networks, especially LSTMs, is appearing to be a very promising technique for market forecasting and it is therefore surprising that it has not already been used in a wider scale.

The objectives of this project—to implement the smoothing process proposed by (Tsantekidis et al., 2017), allowing longer-term predictions (longer than one step ahead prediction) but with a more rigorous approach to parameter selection—have been achieved and have generated results that exceeded our initial expectations. Furthermore, these techniques were applied on a considerably different medium – the one of cryptocurrencies as opposed to the traditional stock markets – fact which once again underlines the original contribution of the present work. Every project milestone was reached and time permitted a trading strategy as an extra accomplishment. While further work can certainly be done, in regard to both forecasting and trading, it could be argued this project has fulfilled all of the objectives set out at the beginning.

References:

1) Dixon, M. (2018). *Sequence classification of the limit order book using recurrent neural networks* - Journal of Computational Science 24 [p.277-286]
2) Guo, T. & Fantulin, N. (2018). *An experimental study of Bitcoin fluctuation using machine learning methods* - arXiv:1802.04065 [stat.ML]
3) Puljiz, M., Stjepan, B. et al. (2018). *Market Microstructure and Order Book Dynamics in Cryptocurrency Exchanges* - Croatian science foundation project ASYRMEA (5349).
4) Tsantekidis, A. (2017). *Using Deep Learning to Detect Price Change Indications in Financial Markets* - 25th European Signal Processing Conference (EUSIPCO) [p.2511-2515]
5) Cocco, L. (2017). *Using an artificial financial market for studying a cryptocurrency market* - Springer, J Econ Interact Coord [p.345–365]
6) Groß W. (2017). *Predicting Time Series with Space-Time Convolutional and Recurrent Neural Networks* - ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), (April 2017) [p.71-76]
7) Revuelta, V. (2018). *Design and implementation of a software system for the composition of a database and automated trading system on different cryptocurrency trading markets* - (ETH Zurich Master Thesis)
8) Greaves A. & Au B. (2015). *Using the Bitcoin Transaction Graph to Predict the Price of Bitcoin* – (snap.stanford.edu > No other data)
9) Härdle, W. (2018). *Time-varying Limit Order Book Networks* - IRTG 1792 Discussion Paper 2018-016
10) *Amjad, M. & Shah, D.* (2017). *Trading Bitcoin and Online Time Series Prediction* - Proceedings of the Time Series Workshop at NIPS 2016, PMLR 55:1-15, 2017.

Appendix:

Project Plan (November)

Dissertation Project Plan and Summary

Andrei Alexandru Maxim

Prediction of cryptocurrency price movements from order book data using LSTM networks

Supervisor**: Dr. Denise Gorse**

Background

There are a few papers that apply Deep Learning to financial data trying to judge by comparison the advantages given by state of the art Artificial Intelligence algorithms over the classical mathematical and statistical approach while trying to analyze market movements. The respective number of papers shrinks even less to the point that we could not find any reasonable paper (with novel enough Deep Learning techniques applied such as LSTM networks) when we are looking at cryptocurrency markets. I believe the overlapping of these two niches - Blockchain and AI – settles a promising developing environment for the paper I intend to write such that the content and approach may be original enough, without having to give up the advantage of having relevant references in other papers.

Aims

To learn about market forecasting and determine if Deep Learning techniques could be applied to gain insight on the price movements for cryptocurrencies.

Progress Update

1. General Reading: various articles on non-academic nor professional websites and blogs that describe possibilities of applying machine learning for time-series evaluation as well as a selection of 9 academic papers found on Google Scholar while searching for different combinations of relevant keywords (Bitcoin/Cryptocurrencies/LSTM/Machine Learning/Forex/Market Forecasting etc.)
2. Focused Reading: from the academic papers I have selected around four that I should inspire from while designing my project, and especially one - Avraam Tsantekidis et al. [1](2017) – that I could even use as a model for the initial experiment within my project as it provides impressing results on forex. Reading numerous studies within these papers, it has been proven that LSTM networks are

the most efficient models to be used when it comes to predicting financial time-series.

3. Data Acquisition: with the help of a Phd student met through my project supervisor, I have managed to get data with the same format as in the Avraam Tsantekidis et al. paper, but for Bitcoin instead of ForEx.

Objectives

1. Study the field and gain understanding on the subject
2. Learn about different market representations (Order-books, spreads, etc.)
3. Learn about different data layouts sampling methods (tick data, set timerate 1hr, 1 day etc.)
4. See relevant research for the best tools to process financial time-series
5. Code a neural network model with the proposed architecture and see how it performs
6. Extra work: see if we can implement a successful trading strategy using the constructed model.

Deliverables / Expected Outcomes

We expect to be able to replicate the results mentioned in the Avraam Tsantekidis et al. paper, for Bitcoin instead of ForEx, using LSTM networks with Keras and Tensorflow on the data that I have already gained, which should give a thorough response to the main aspect of my dissertation – how Deep Learning might perform in predicting cryptocurrency market movements.

However, from there we would also like to see if we can implement a successful trading strategy based on our model.

Project Timeline

Start of October – Covering Term 1 & Term 2 – Submission Deadline 29th April ~ 6 months

Supervisor meetings have been completed in a number of 4 so far and have agreed to regular meetings for progress report and discussions.

October – the preliminary research and initial work (scouting feasibility of data acquisition and techniques themselves) and planning the overall project milestones

November – I have gained the data needed for the initial experiment and I should be able to code a LSTM and see how it performs by the end of the month/

December / first half of January– Depending on my results I will then see whether I am to move forward and build an entire framework and strategy based on LSTM prediction

capability, or if proven not to be feasible, research on why that may be – differences between ForEx and Bitcoin that make them perform differently when fed to a neural network – and also try different algorithms to see if we get the same results.

Late January – I will begin project write-up based on notes collected throughout each step of the project so far.

As a sidenote, the plan is more well-defined for the first term than it is for the second term, and I am planning on completing this plan further in time for the intermarry report as I believe, since this is mainly a research project, my plans would vary considerably depending on the results I get at these stages of the project (as underlined in my plan for December).

Reference:

Using Deep Learning to Detect Price Change Indications in Financial Markets (Avraam Tsantekidis∗, Nikolaos Passalis∗, Anastasios Tefas∗ …) – [2017] 25th European Signal Processing Conference (EUSIPCO) (p.2511-2515)

Interim Report (February)

Intermarry Report

Andrei Alexandru Maxim

Prediction of cryptocurrency price movements from order book data using LSTM networks

Supervisor**: Dr. Denise Gorse**

Progress

As stated in the November plan, the first task that I've completed was to setup an experiment with the data I have been able to provide, to see whether I could get some good preliminary results with just a basic LSTM network. I have started by performing pre-processing for the data to prepare it to be fed into the network for the first basic experiment. It was then that my first concern came up. The data I have been given seemed to present an anomaly, more specifically, the servers collecting the information have been down during a significant period of the time at the middle of the studied period. That data was enough for me to get started on and code every bit of the experiment, however when it came down to perform optimizations of the parameters involved, I had asked for another set of data that had integrity – even though the anomaly itself did not interfere with the training of the algorithm, a bi-product of it did: considerably less data made the algorithm not able to generalize well.

Some other aspects that we had to take special caution with were the amount of inconsistencies and mistakes in the paper that we wished to use as a starting point. Many of the important technical details have been left uncovered, only briefly mentioned, so we had to figure out on our own what each of the key-parameters should be related to and how to choose them.

To be allowing for a bit more detail as to better report the challenges: there are two key values in this approach: a certain "k" that indicates the lookahead during training and an "alpha" that represents the threshold. The network is taking as features for each timestep present in the dataset, 40 values in total (10 first bid prices, 10 first ask prices, and their volumes within the order book – 10 more from each). The algorithm then interprets the data into three labels: going UP, going DOWN, doing nothing. The split between the three categories is conducted by the indications of the two aforementioned parameters k and alpha. When analyzing a timestep, it labels UP or DOWN if the the k-th element before is subtracted from the k-th timestep forward is greater or less than 0. That gives a more general overview on the price relating to time, with k getting bigger. Then the alpha parameter decides whether the difference obtained is high enough so that it is considered a movement on the market or it is too little to be taken into consideration. The essential

problem is that different combinations for K and alpha have a great effect on the distribution of labels into the three categories. It is known that the most efficient way of spreading the labels is the one that gives roughly the same number of elements within each. However, we have reasons to believe that in the Avraam Tsantekidis et al. paper the same value of alpha is used for different Ks, thus artificially complicating the work of the algorithm. Therefore, we expect that with an appropriate value of alpha for each K used, we should not observe the same decrease in efficiency as they say with K getting larger. This is actually this week's work and I am currently performing another experiment to see whether our assumption is true.

I have also researched different metrics for evaluating the algorithm, as the standard "accuracy" one has proved to be inappropriate for a relevant insight into the performance. Therefore, our current analysis constructs a Confusion Matrix that we use as base for implementing the Matthews Coefficient as a performance metric for our model. This coefficient has been chosen over other candidates like Cohen's Kappa due to its formula for n-classes problems that fits well for our purpose.

All in all, I have managed to get a satisfactory prediction performance with my model, between 0.27 and 0.30 Matthews.

The work I'm starting now is shifting the focus from having an efficient prediction to optimizing the system for profit. There are different approaches to the problem once one sets different goals with it, so many principles from these experiments must be slightly changed to implement a successful trading strategy. Having formed a solid insight on the problem and all its technicalities, I have a very good base for consistently describing all that is happening within the algorithm and the decisions I had to make. I have already started writing up the Final Report for my dissertation and I'm planning to continue working on it at the same time I am developing the last part of my project.

Remaining work

The plan remains for the following period to:

- Implement a trading strategy on top of the developed algorithm and see if it could be efficiently used for profit on a less efficient market such as Bitcoin.
- Write the effective report of my dissertation.

It would be interesting to see, should all my results point in the expected direction, why this technique has not been used for cryptocurrencies yet. Long-Short Term Memory Networks have proved to be efficient in a number of prediction problems including in the analysis of the stock market and foreign exchange.[2]

Technical details and libraries used

I have used python as the base language for this project – for ease of presentation during the meetings with my supervisor I have assembled the layout of Jupyter Notebooks for the possibility of running different cells individually and more freedom of idea exploration. The initial visual analysis and data extraction from the CSV files I've made with the Pandas and matplotlib python libraries, the pre-processing and data storage I've made using the NumPy library – which is considered to be the most efficient practice working with large arrays and optimized data processing. For standard data manipulation immediately before the neural network I have used the basic sci-kit learn pack, and for the deep learning itself I have used Keras with a TensorFlow (gpu accelerated) backend.

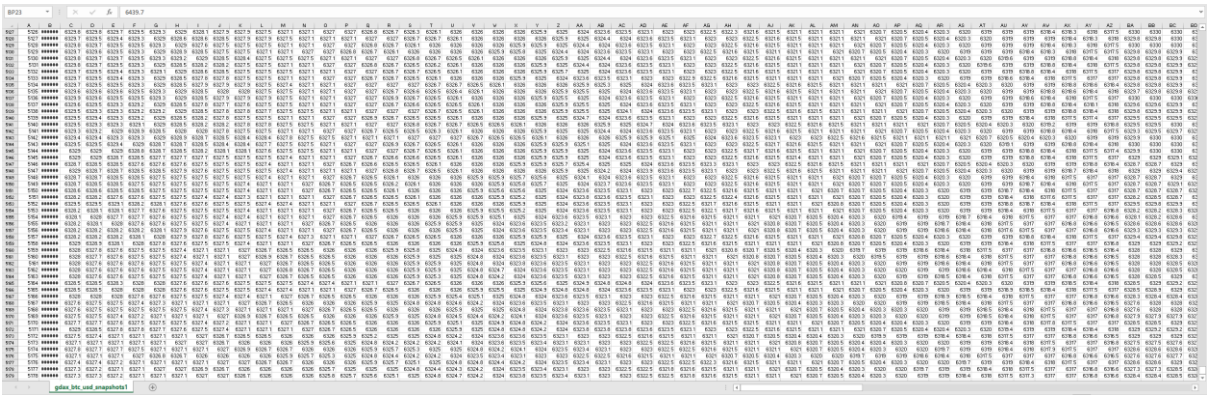
Reference:

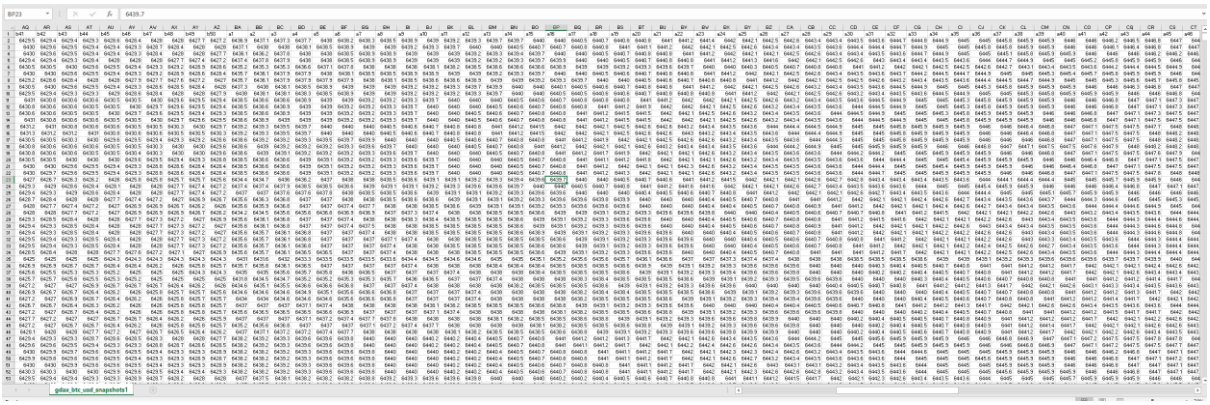
[1] *Using Deep Learning to Detect Price Change Indications in Financial Markets* (Avraam Tsantekidis∗, Nikolaos Passalis∗, Anastasios Tefas∗ …) – [2017] 25th European Signal Processing Conference (EUSIPCO) (p.2511-2515)

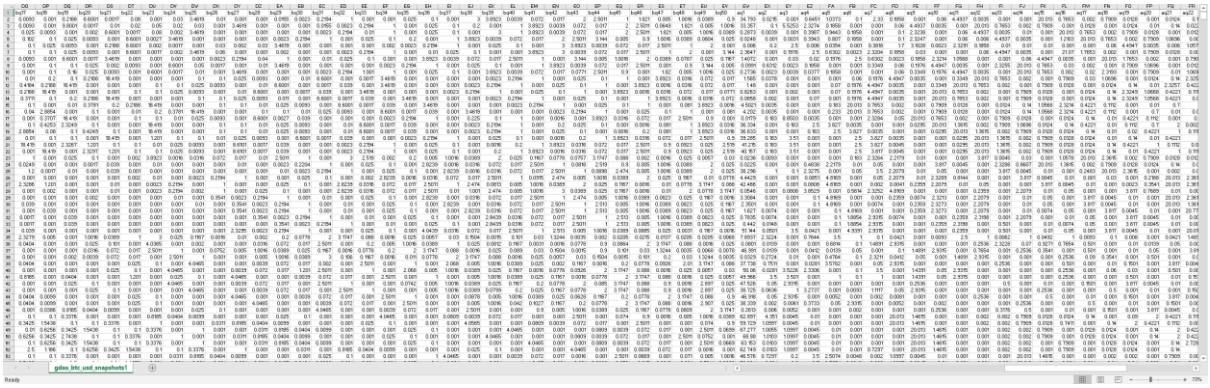
[2] *Sequence Classification of the Limit Order Book using Recurrent Neural Networks* (Matthew Dixon) – Stuart School of Business, Illinois Institute of Technology – Journal of Computational Science 24 (2018) (p.277-286)

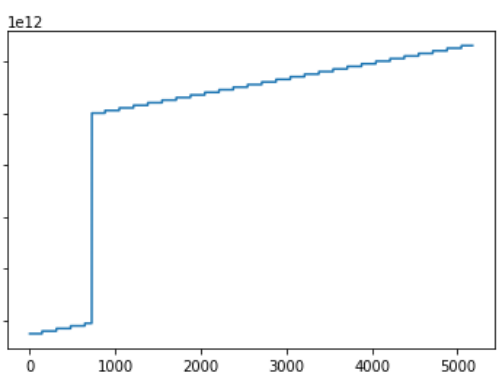
Data Samples

(Dataset 1 – 5000 timesteps / 200features/step)



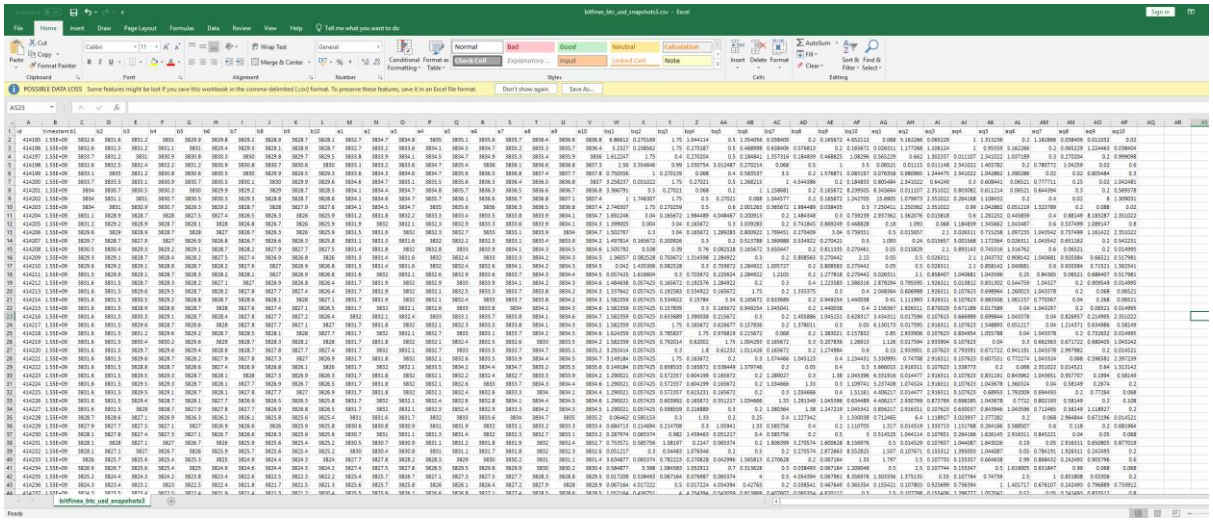


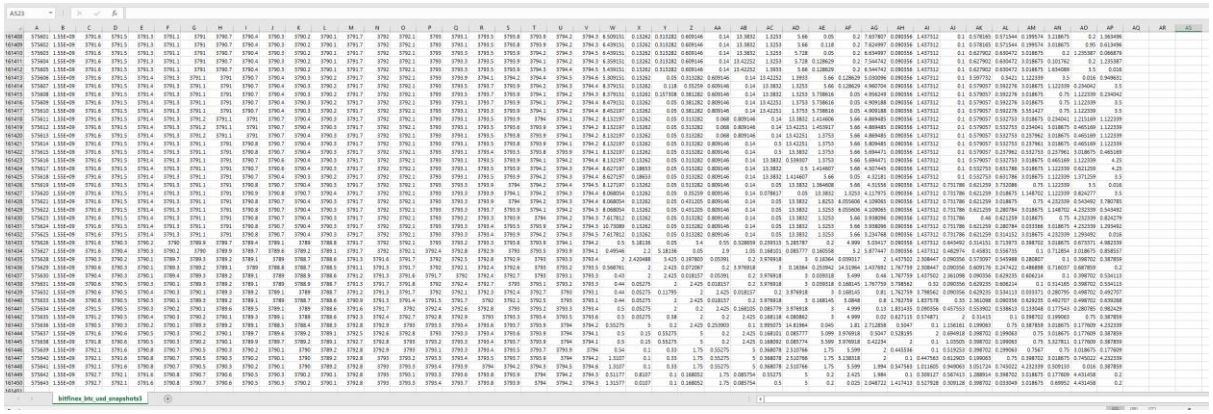




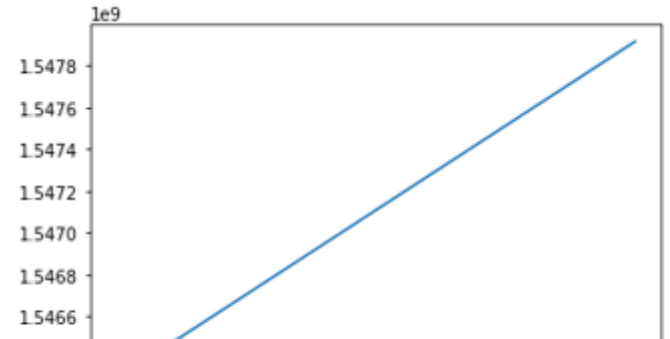
plot id/ timestep:

(Dataset 2 – 40 features / 16,000 timesteps)





plot id / timestep



MCC full results

```
Run number 1 k=1 alpha=0.00 Accuracy=0.64       Matthews Coefficient = 0.2419 (0.15, 0.36, 0.21)
Run number 2 k=1 alpha=0.00 Accuracy=0.64       Matthews Coefficient = 0.2407 (0.17, 0.36, 0.19)
Run number 3 k=1 alpha=0.00 Accuracy=0.63       Matthews Coefficient = 0.2413 (0.15, 0.36, 0.21)
Run number 4 k=1 alpha=0.00 Accuracy=0.63       Matthews Coefficient = 0.2434 (0.15, 0.36, 0.21)
Run number 5 k=1 alpha=0.00 Accuracy=0.63       Matthews Coefficient = 0.2362 (0.15, 0.37, 0.19)
Run number 6 k=1 alpha=0.00 Accuracy=0.63       Matthews Coefficient = 0.2424 (0.14, 0.37, 0.22)
Run number 7 k=1 alpha=0.00 Accuracy=0.63       Matthews Coefficient = 0.2449 (0.16, 0.36, 0.21)
Run number 8 k=1 alpha=0.00 Accuracy=0.64       Matthews Coefficient = 0.2398 (0.16, 0.36, 0.20)
Run number 9 k=1 alpha=0.00 Accuracy=0.64       Matthews Coefficient = 0.2382 (0.15, 0.36, 0.21)
Run number 10 k=1 alpha=0.00 Accuracy=0.63      Matthews Coefficient = 0.2450 (0.18, 0.37, 0.19)

Run number 1 k=2 alpha=0.00 Accuracy=0.55       Matthews Coefficient = 0.2683 (0.22, 0.37, 0.21)
Run number 2 k=2 alpha=0.00 Accuracy=0.55       Matthews Coefficient = 0.2621 (0.17, 0.37, 0.25)
Run number 3 k=2 alpha=0.00 Accuracy=0.55       Matthews Coefficient = 0.2588 (0.19, 0.36, 0.22)
Run number 4 k=2 alpha=0.00 Accuracy=0.55       Matthews Coefficient = 0.2701 (0.22, 0.37, 0.21)
Run number 5 k=2 alpha=0.00 Accuracy=0.55       Matthews Coefficient = 0.2579 (0.17, 0.36, 0.24)
Run number 6 k=2 alpha=0.00 Accuracy=0.55       Matthews Coefficient = 0.2675 (0.19, 0.37, 0.24)
Run number 7 k=2 alpha=0.00 Accuracy=0.56       Matthews Coefficient = 0.2600 (0.18, 0.37, 0.23)
Run number 8 k=2 alpha=0.00 Accuracy=0.55       Matthews Coefficient = 0.2633 (0.21, 0.37, 0.21)
Run number 9 k=2 alpha=0.00 Accuracy=0.55       Matthews Coefficient = 0.2599 (0.17, 0.37, 0.24)
Run number 10 k=2 alpha=0.00 Accuracy=0.56      Matthews Coefficient = 0.2657 (0.20, 0.37, 0.23)

Run number 1 k=3 alpha=0.00 Accuracy=0.52       Matthews Coefficient = 0.2604 (0.18, 0.36, 0.24)
Run number 2 k=3 alpha=0.00 Accuracy=0.52       Matthews Coefficient = 0.2646 (0.20, 0.37, 0.23)
Run number 3 k=3 alpha=0.00 Accuracy=0.52       Matthews Coefficient = 0.2646 (0.17, 0.37, 0.25)
Run number 4 k=3 alpha=0.00 Accuracy=0.51       Matthews Coefficient = 0.2563 (0.17, 0.36, 0.24)
Run number 5 k=3 alpha=0.00 Accuracy=0.52       Matthews Coefficient = 0.2589 (0.17, 0.36, 0.25)
Run number 6 k=3 alpha=0.00 Accuracy=0.52       Matthews Coefficient = 0.2651 (0.18, 0.38, 0.24)
Run number 7 k=3 alpha=0.00 Accuracy=0.52       Matthews Coefficient = 0.2626 (0.18, 0.37, 0.24)
Run number 8 k=3 alpha=0.00 Accuracy=0.51       Matthews Coefficient = 0.2546 (0.16, 0.36, 0.24)
Run number 9 k=3 alpha=0.00 Accuracy=0.52       Matthews Coefficient = 0.2700 (0.21, 0.38, 0.22)
Run number 10 k=3 alpha=0.00 Accuracy=0.51      Matthews Coefficient = 0.2604 (0.17, 0.37, 0.25)

Run number 1 k=5 alpha=0.05 Accuracy=0.49       Matthews Coefficient = 0.2265 (0.16, 0.29, 0.22)
Run number 2 k=5 alpha=0.05 Accuracy=0.49       Matthews Coefficient = 0.2261 (0.17, 0.30, 0.21)
Run number 3 k=5 alpha=0.05 Accuracy=0.49       Matthews Coefficient = 0.2314 (0.17, 0.31, 0.21)
Run number 4 k=5 alpha=0.05 Accuracy=0.49       Matthews Coefficient = 0.2293 (0.17, 0.30, 0.22)
Run number 5 k=5 alpha=0.05 Accuracy=0.49       Matthews Coefficient = 0.2297 (0.18, 0.31, 0.21)
Run number 6 k=5 alpha=0.05 Accuracy=0.49       Matthews Coefficient = 0.2338 (0.16, 0.31, 0.23)
Run number 7 k=5 alpha=0.05 Accuracy=0.49       Matthews Coefficient = 0.2300 (0.18, 0.30, 0.21)
Run number 8 k=5 alpha=0.05 Accuracy=0.49       Matthews Coefficient = 0.2243 (0.14, 0.30, 0.23)
Run number 9 k=5 alpha=0.05 Accuracy=0.49       Matthews Coefficient = 0.2354 (0.17, 0.31, 0.23)
Run number 10 k=5 alpha=0.05 Accuracy=0.49      Matthews Coefficient = 0.2294 (0.16, 0.31, 0.22)
```

```
Run number 1 k=10 alpha=0.20 Accuracy=0.44      Matthews Coefficient = 0.1717 (0.15, 0.20, 0.16)
Run number 2 k=10 alpha=0.20 Accuracy=0.45      Matthews Coefficient = 0.1785 (0.15, 0.21, 0.17)
Run number 3 k=10 alpha=0.20 Accuracy=0.45      Matthews Coefficient = 0.1816 (0.15, 0.22, 0.17)
Run number 4 k=10 alpha=0.20 Accuracy=0.45      Matthews Coefficient = 0.1728 (0.14, 0.20, 0.17)
Run number 5 k=10 alpha=0.20 Accuracy=0.44      Matthews Coefficient = 0.1674 (0.14, 0.19, 0.17)
Run number 6 k=10 alpha=0.20 Accuracy=0.44      Matthews Coefficient = 0.1699 (0.14, 0.20, 0.17)
Run number 7 k=10 alpha=0.20 Accuracy=0.45      Matthews Coefficient = 0.1752 (0.15, 0.20, 0.17)
Run number 8 k=10 alpha=0.20 Accuracy=0.45      Matthews Coefficient = 0.1763 (0.15, 0.21, 0.18)
Run number 9 k=10 alpha=0.20 Accuracy=0.44      Matthews Coefficient = 0.1686 (0.14, 0.19, 0.18)
Run number 10 k=10 alpha=0.20 Accuracy=0.46     Matthews Coefficient = 0.1862 (0.16, 0.22, 0.18)

Run number 1 k=20 alpha=0.50 Accuracy=0.39      Matthews Coefficient = 0.0956 (0.11, 0.06, 0.12)
Run number 2 k=20 alpha=0.50 Accuracy=0.39      Matthews Coefficient = 0.0970 (0.11, 0.06, 0.12)
Run number 3 k=20 alpha=0.50 Accuracy=0.39      Matthews Coefficient = 0.1017 (0.11, 0.09, 0.11)
Run number 4 k=20 alpha=0.50 Accuracy=0.40      Matthews Coefficient = 0.1075 (0.11, 0.09, 0.12)
Run number 5 k=20 alpha=0.50 Accuracy=0.39      Matthews Coefficient = 0.1020 (0.13, 0.07, 0.11)
Run number 6 k=20 alpha=0.50 Accuracy=0.39      Matthews Coefficient = 0.1039 (0.11, 0.08, 0.13)
Run number 7 k=20 alpha=0.50 Accuracy=0.39      Matthews Coefficient = 0.0904 (0.10, 0.07, 0.10)
Run number 8 k=20 alpha=0.50 Accuracy=0.40      Matthews Coefficient = 0.1015 (0.11, 0.08, 0.11)
Run number 9 k=20 alpha=0.50 Accuracy=0.40      Matthews Coefficient = 0.1061 (0.12, 0.07, 0.13)
Run number 10 k=20 alpha=0.50 Accuracy=0.40     Matthews Coefficient = 0.0978 (0.10, 0.08, 0.11)
```

```
Run number 1 k=30 alpha=0.75 Accuracy=0.37      Matthews Coefficient = 0.0754 (0.10, 0.03, 0.10)
Run number 2 k=30 alpha=0.75 Accuracy=0.36      Matthews Coefficient = 0.0520 (0.08, -0.00, 0.08)
Run number 3 k=30 alpha=0.75 Accuracy=0.38      Matthews Coefficient = 0.0711 (0.08, 0.03, 0.10)
Run number 4 k=30 alpha=0.75 Accuracy=0.38      Matthews Coefficient = 0.0716 (0.08, 0.03, 0.10)
Run number 5 k=30 alpha=0.75 Accuracy=0.37      Matthews Coefficient = 0.0669 (0.08, 0.02, 0.10)
Run number 6 k=30 alpha=0.75 Accuracy=0.38      Matthews Coefficient = 0.0763 (0.09, 0.04, 0.10)
Run number 7 k=30 alpha=0.75 Accuracy=0.37      Matthews Coefficient = 0.0656 (0.09, 0.02, 0.09)
Run number 8 k=30 alpha=0.75 Accuracy=0.37      Matthews Coefficient = 0.0700 (0.08, 0.03, 0.09)
Run number 9 k=30 alpha=0.75 Accuracy=0.37      Matthews Coefficient = 0.0706 (0.09, 0.02, 0.10)
Run number 10 k=30 alpha=0.75 Accuracy=0.38     Matthews Coefficient = 0.0726 (0.10, 0.03, 0.09)

Run number 1 k=45 alpha=1.00 Accuracy=0.36      Matthews Coefficient = 0.0424 (0.05, 0.01, 0.07)
Run number 2 k=45 alpha=1.00 Accuracy=0.37      Matthews Coefficient = 0.0564 (0.07, 0.02, 0.08)
Run number 3 k=45 alpha=1.00 Accuracy=0.36      Matthews Coefficient = 0.0424 (0.06, 0.00, 0.06)
Run number 4 k=45 alpha=1.00 Accuracy=0.37      Matthews Coefficient = 0.0526 (0.06, 0.02, 0.07)
Run number 5 k=45 alpha=1.00 Accuracy=0.37      Matthews Coefficient = 0.0550 (0.07, 0.02, 0.07)
Run number 6 k=45 alpha=1.00 Accuracy=0.36      Matthews Coefficient = 0.0451 (0.07, 0.00, 0.06)
Run number 7 k=45 alpha=1.00 Accuracy=0.38      Matthews Coefficient = 0.0612 (0.08, 0.04, 0.07)
Run number 8 k=45 alpha=1.00 Accuracy=0.36      Matthews Coefficient = 0.0476 (0.06, 0.01, 0.07)
Run number 9 k=45 alpha=1.00 Accuracy=0.37      Matthews Coefficient = 0.0513 (0.06, 0.03, 0.07)
Run number 10 k=45 alpha=1.00 Accuracy=0.37     Matthews Coefficient = 0.0512 (0.06, 0.02, 0.07)


Run number 1 k=60 alpha=1.30 Accuracy=0.35      Matthews Coefficient = 0.0216 (0.05, -0.00, 0.02)
Run number 2 k=60 alpha=1.30 Accuracy=0.37      Matthews Coefficient = 0.0472 (0.06, 0.02, 0.06)
Run number 3 k=60 alpha=1.30 Accuracy=0.36      Matthews Coefficient = 0.0342 (0.05, 0.00, 0.05)
Run number 4 k=60 alpha=1.30 Accuracy=0.36      Matthews Coefficient = 0.0295 (0.04, 0.00, 0.04)
Run number 5 k=60 alpha=1.30 Accuracy=0.37      Matthews Coefficient = 0.0447 (0.04, 0.03, 0.06)
Run number 6 k=60 alpha=1.30 Accuracy=0.36      Matthews Coefficient = 0.0345 (0.05, -0.00, 0.06)
Run number 7 k=60 alpha=1.30 Accuracy=0.36      Matthews Coefficient = 0.0326 (0.03, 0.01, 0.06)
Run number 8 k=60 alpha=1.30 Accuracy=0.37      Matthews Coefficient = 0.0410 (0.07, 0.00, 0.05)
Run number 9 k=60 alpha=1.30 Accuracy=0.36      Matthews Coefficient = 0.0264 (0.05, -0.01, 0.03)
Run number 10 k=60 alpha=1.30 Accuracy=0.37     Matthews Coefficient = 0.0411 (0.05, 0.01, 0.06)

Run number 1 k=120 alpha=2.00 Accuracy=0.37     Matthews Coefficient = 0.0062 (0.06, -0.01, -0.03)
Run number 2 k=120 alpha=2.00 Accuracy=0.36     Matthews Coefficient = 0.0418 (0.07, 0.00, 0.06)
Run number 3 k=120 alpha=2.00 Accuracy=0.34     Matthews Coefficient = 0.0133 (0.04, -0.01, 0.01)
Run number 4 k=120 alpha=2.00 Accuracy=0.35     Matthews Coefficient = 0.0185 (0.05, -0.03, 0.04)
Run number 5 k=120 alpha=2.00 Accuracy=0.36     Matthews Coefficient = 0.0370 (0.05, -0.00, 0.06)
Run number 6 k=120 alpha=2.00 Accuracy=0.38     Matthews Coefficient = 0.0460 (0.06, 0.02, 0.06)
Run number 7 k=120 alpha=2.00 Accuracy=0.35     Matthews Coefficient = 0.0225 (0.05, -0.02, 0.05)
Run number 8 k=120 alpha=2.00 Accuracy=0.37     Matthews Coefficient = 0.0217 (0.06, -0.01, 0.02)
Run number 9 k=120 alpha=2.00 Accuracy=0.36     Matthews Coefficient = 0.0003 (0.03, -0.02, -0.01)
Run number 10 k=120 alpha=2.00 Accuracy=0.37    Matthews Coefficient = 0.0377 (0.06, 0.01, 0.04)

Run number 1 k=240 alpha=2.70 Accuracy=0.37     Matthews Coefficient = 0.0538 (0.06, 0.02, 0.08)
```

Code entry:

Plot Matthews Coefficient:

PlotMatthewsCoefPerformance.py

```
from matplotlib import pyplot as plt
from collections import defaultdict
import numpy as np

file = open('input(Real).txt', 'r')
raw = file.readlines()
file.close()

#d = defaultdict(list)

#d = {1:[0.,1., 0.], 2:[0.,1., 0.], 3:[0.,1., 0.], 5:[0.,1., 0.], 10:[0.,1., 0.], 20:[0.,1., 0.], 30:[0.,1., 0.]}
'''
k:(accumulator, minVal, maxVal)
d[k][0] = acc
d[k][1] = min
d[k][2] = max                                    ACTUALLY I DON"T NEED DICTS... :(
'''

dict={1:0, 2:1, 3:2, 5:3, 10:4, 20:5, 30:6}
dictInverse={0:1, 1:2, 2:3, 3:5, 4:10, 5:20, 6:30}

d = [[0.,1., 0.], [0.,1., 0.], [0.,1., 0.], [0.,1., 0.], [0.,1., 0.], [0.,1., 0.], [0.,1., 0.]] # 7 terms up to k=30


for line in range(len(raw)):  # Go through each line in input.txt (from accuracy.txt)
    if(raw[line]!="\n"):
        k=int(raw[line].split('\t')[0].split()[3][2:]) # Select k value
        index = dict[k]  # lookup k index in d
        #print(k)
        current = raw[line].split('\t')[1]
        temp = float(current.split()[3])  # select MC
        d[index][0] = d[index][0] + temp    # accumulate on k

        # NOW ADD ERROR MARGINS -> Retain MIN value and MAX
        if(d[index][1]>temp):
            d[index][1] = temp
        if(d[index][2]<temp):
            d[index][2] = temp
```

```

xAxis = []
yAxis = []

for each in range(len(d)):
    d[each][0] = d[each][0]/10
    print(d[each])

    #aah = np.asarray([d[each][0]-d[each][1],d[each][2]-d[each][0]])
    #print(aah.shape)
    #plt.errorbar(dictInverse[each], d[each][0], yerr=[aah], ecolor='blue', barsabove='true')#, capsize=2)
    plt.plot(dictInverse[each],d[each][2],dictInverse[each],d[each][1])
    plt.plot(dictInverse[each],d[each][2],'r2')
    plt.plot(dictInverse[each],d[each][1],'r1')
    xAxis.append(dictInverse[each])
    yAxis.append(d[each][0])


print(d)

plt.plot(xAxis, yAxis,'#75bbfd')
plt.ylabel("Matthew's Coefficient")
plt.xlabel("ValueS of 'k'")
plt.savefig("Try2.png")
plt.show()

```

Feature extraction:

ExtractFeatures.py

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
#import tensorflow-gpu as tf
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout,  Flatten
from keras.utils import to_categorical
from keras.models import load_model
from keras import backend as K

from sklearn.model_selection import train_test_split

trades10 = pd.read_csv('./Datasets/gdax_btc_usd_snapshots1.csv')
#trades10 = pd.read_csv('./Datasets/bitfinex_btc_usd_snapshots3.csv')

# DATA EXTRACTION
# ____________________________________________________________

times = np.asarray(trades10["timestamp"])
rawBidPrice = []
rawAskPrice = []
rawBidDepth = []
rawAskDepth = []
no_entries = trades10["id"].count()

for j in range(no_entries):
    for i in range(1,11):
        rawBidPrice.append(trades10["b%d"%i][j])
        rawAskPrice.append(trades10["a%d"%i][j])
        rawBidDepth.append(trades10["bq%d"%i][j])
        rawAskDepth.append(trades10["aq%d"%i][j])

#print(len(rawBidPrice))
bidPrice4step = np.asarray(rawBidPrice).reshape(no_entries, 10)
askPrice4step = np.asarray(rawAskPrice).reshape(no_entries, 10)
bidDepth4step = np.asarray(rawBidDepth).reshape(no_entries, 10)
askDepth4step = np.asarray(rawAskDepth).reshape(no_entries, 10)

#np.set_printoptions(threshold=1000#np.nan)
#print(bidPrice4step)
print(bidPrice4step.shape)
print(askPrice4step.shape)
print(bidDepth4step.shape)
print(askDepth4step.shape)


# Mid price is for timestep t:
midPrice = []
for t in range(no_entries):
    midPrice.append((trades10["a1"][t]+trades10["b1"][t])/2)
midPrice  = np.asarray(midPrice)

print(midPrice)

# NORMALIZATION
#_____________________________________________________________

X_prices = np.concatenate((bidPrice4step, askPrice4step), axis=1)
X_volumes = np.concatenate((bidDepth4step, askDepth4step), axis=1)

prices_mean = np.mean(X_prices)
volumes_mean = np.mean(X_volumes)
print(prices_mean, volumes_mean)

prices_std = np.std(X_prices)
volumes_std = np.std(X_volumes)
print(prices_std, volumes_std)

# Normalizing
X_prices = (X_prices-prices_mean)/prices_std
X_volumes = (X_volumes-volumes_mean)/volumes_std

X = np.concatenate((X_prices, X_volumes), axis = 1 )

X = X.reshape(X.shape[0], 1, X.shape[1] )

np.save("X_dataset1", X)
```

Code for model training:

TheCode (training).py

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
#import tensorflow-gpu as tf
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout,  Flatten
from keras.utils import to_categorical
from keras.models import load_model
from keras import backend as K

from sklearn.model_selection import train_test_split

# pd.set_option('display.mpl_style', 'default') # Make the graphs a bit prettier
# plt.rcParams['figure.figsize'] = (15, 5)

#trades10 = pd.read_csv('./Datasets/gdax_btc_usd_snapshots1.csv')
trades10 = pd.read_csv('./Datasets/bitfinex_btc_usd_snapshots3.csv')


#print(trades10[:5])
trades10["timestamp"].plot() # PLOT TIMESTAMPS
# trades10["timestamp"]

trades10["b1"].plot()  # PLOT PRICES OF THE DATASET

# DATA EXTRACTION
# ____________________________________________________________

# NORMALIZATION
#____________________________________________________________

# FROM FILE - EASY WAY OUT

X = np.load("X_dataset2.npy")

no_entries = trades10["id"].count()

# Mid price is for timestep t:
midPrice = []
for t in range(no_entries):
    midPrice.append((trades10["a1"][t]+trades10["b1"][t])/2)
midPrice  = np.asarray(midPrice)

print(midPrice)

```

```
# LABEL CREATION, ALPHA AND K SET
#__________________________________________________________

labels = []
alpha = 1.
k = 45
for i in range(no_entries):
    if i<(no_entries-k) and i>k:
        kPrev=0
        kFollow=0
        for stepK in range(1,k+1):
            kPrev=kPrev+midPrice[i-stepK]
            kFollow=kFollow+midPrice[i+stepK]
        kPrev=kPrev/k
        kFollow=kFollow/k

        if kFollow>(kPrev+alpha):
            labels.append(2)
        elif kFollow<(kPrev-alpha):
            labels.append(0)
        else :
            labels.append(1)
    else:
        labels.append(1)


#print(labels)
print("Number of 0s: ", labels.count(0))
print("Number of 1s: ", labels.count(1))
print("Number of 2s: ", labels.count(2))


labels = np.asarray(labels)
labels = to_categorical(labels)


# NORMALIZATION
#__________________________________________________________



#TRAIN TEST VALIDATION SPLIT
#__________________________________________________________

X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, shuffle = False)

_, X_val, _, y_val = train_test_split(X_train, y_train, test_size=0.2, shuffle = False)


number_of_runs = 10

for run in range(10,number_of_runs+1):

    # BUILDING THE NETWORK
    #_____________________________________________________

    model = Sequential()

    model.add(LSTM(15, activation='tanh', dropout=0.0, recurrent_dropout=0.0, input_shape=(1 ,X_train.shape[2
    #model.add(Flatten())
    model.add(Dense(3, activation='softmax'))
    #model.add(Flatten())
    model.compile(loss='categorical_crossentropy', optimizer = "adam", metrics=["accuracy"])

    print(model.summary())


    # TRAINING
    #_____________________________________________________

    history = model.fit(X_train, y_train, epochs=100, batch_size=64, shuffle=False, validation_data=(X_val, y

    # TESTING
    #_____________________________________________________

    evaluation = model.evaluate(x=X_test, y=y_test, batch_size=64, verbose=1, sample_weight=None, steps=None)

    print(evaluation)

    # plt.plot(history.history["acc"])
    # plt.plot(history.history["val_acc"])
    # plt.plot(history.history["loss"])
    # plt.plot(history.history["val_loss"])


    # Building the CONFUSION MATRIX
    #_____________________________________________________

    predictions = model.predict(X_test, batch_size=64, verbose=0, steps=None)
    predictions = np.argmax(predictions, axis=1)
    grounds = np.argmax(y_test, axis=1)

    print(grounds)
    print(predictions)

    confusion_matrix = np.zeros((3, 3))

    for i in range(len(predictions)):
        x = grounds[i]
        y = predictions[i]
        confusion_matrix[x][y] += 1

    print(confusion_matrix)
```

```
    # MATTHEWS COEFFICIENT
    #_____________________________________________________________

    values = []

    for l in range(3):
        p = confusion_matrix[l][l] # number of correctly assigned to l class
        n = 0
        u = 0
        o = 0
        for i in range(3):
            for j in range(3):
                if i != l and j != l:
                    n = n + confusion_matrix[i][j]
                if i != l and j == l:
                    u = u + confusion_matrix[i][j]
                if i ==l and j != l:
                    o = o + confusion_matrix[i][j]
        mcc = (p*n - u*o)/((p+u)*(p+o)*(n+u)*(n+o))**(1/2)
        values.append(mcc)

    print(values)

    final_coefficient = (values[0] + values[1] + values[2])/3
    a = values[0]
    b = values[1]
    c = values[2]

    print(final_coefficient)


    plt.clf()

    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model train vs validation accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper right')
    plt.savefig('./Output/Run%d K%d alpha%.2f accuracy.png' %(run, k, alpha))
    plt.clf()

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model train vs validation loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper right')
    plt.savefig('./Output/Dataset1 Run%d K%d alpha%.2f loss.png' %(run, k, alpha))
    plt.clf()
```

```
    maif = open('accuracy.txt', 'a')
    maif.write('Run number %d k=%d alpha=%.2f Accuracy=%.2f\t Matthews Coefficient = %.4f (%.2f, %.2f, %.2f) \n' % (run, k, alpha, evaluation[1], final_coefficient, a, b ,c))
    maif.close()



    model.save('./Output/run%dk%dalpha%.2f.h5' % (run, k, alpha))

    K.clear_session()




# THIS IS THE END, my only friend, the end;
```

Code for trading strategy implementation (k-steps):

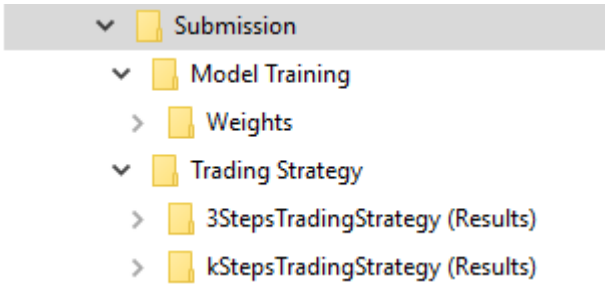
TheCode (trading-ksteps).py

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
#import tensorflow-gpu as tf
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout,  Flatten
from keras.utils import to_categorical
from keras.models import load_model
from keras import backend as K
import os

from sklearn.model_selection import train_test_split

# pd.set_option('display.mpl_style', 'default') # Make the graphs a bit prettier
# plt.rcParams['figure.figsize'] = (15, 5)

#trades10 = pd.read_csv('./Datasets/gdax_btc_usd_snapshots1.csv')
trades10 = pd.read_csv('./Datasets/bitfinex_btc_usd_snapshots3.csv')


#print(trades10[:5])
trades10["timestamp"].plot() # PLOT TIMESTAMPS
# trades10["timestamp"]

trades10["b1"].plot()  # PLOT PRICES OF THE DATASET

# DATA EXTRACTION
# ____________________________________________________________

no_entries = trades10["id"].count()

X = np.load("x_dataset2.npy")


# Mid price is for timestep t:
midPrice = []
for t in range(no_entries):
    midPrice.append((trades10["a1"][t]+trades10["b1"][t])/2)

#midPrice = midPrice[-30000:]
midPrice  = np.asarray(midPrice)
'''
print(midPrice)

plt.plot(midPrice)
plt.ylabel("Price")
plt.xlabel("Timestep")
plt.savefig("Price per last 30000 timesteps.png")
plt.show()
'''

# LABEL CREATION, ALPHA AND K SET
#____________________________________________________________

#params = [(1,0.),(2,0.),(3,0.),(5,0.05),(10,0.2),(20,0.5),(30,0.75)]

#for pair in params:

labels = []
alpha = 0.
k = 3

os.makedirs('./kStepsTradingStrategy/k%dalpha%.2f' % (k, alpha))

for i in range(no_entries):
    if i<(no_entries-k) and i>k:
        kPrev=0
        kFollow=0
        for stepK in range(1,k+1):
            kPrev=kPrev+midPrice[i-stepK]
            kFollow=kFollow+midPrice[i+stepK]
        kPrev=kPrev/k
        kFollow=kFollow/k

        if kFollow>(kPrev+alpha):
            labels.append(2)
        elif kFollow<(kPrev-alpha):
            labels.append(0)
        else :
            labels.append(1)
    else:
        labels.append(1)


#print(labels)
print("Number of 0s: ", labels.count(0))
print("Number of 1s: ", labels.count(1))
print("Number of 2s: ", labels.count(2))


labels = np.asarray(labels)
labels = to_categorical(labels)


#TRAIN TEST VALIDATION SPLIT
#____________________________________________________________

X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, shuffle = False)

_, X_val, _, y_val = train_test_split(X_train, y_train, test_size=0.2, shuffle = False)

#X_test, X_train, y_test, y_train = train_test_split(X, labels, test_size=0.6, shuffle = False)

#_, X_val, _, y_val = train_test_split(X_train, y_train, test_size=0.2, shuffle = False)


for run in range(1,11):

    model = load_model('./../Training/Output/Weights/k%dalpha%.2f/run%dk%dalpha%.2f.h5' % (k, alpha, run, k,

    evaluation = model.evaluate(x=X_test, y=y_test, batch_size=64, verbose=1, sample_weight=None, steps=None)

    print(evaluation)

    # plt.plot(history.history["acc"])
    # plt.plot(history.history["val_acc"])
    # plt.plot(history.history["loss"])
    # plt.plot(history.history["val_loss"])


    # Building the CONFUSION MATRIX
    #____________________________________________________________

    predictions = model.predict(X_test, batch_size=64, verbose=0, steps=None)
    predictions = np.argmax(predictions, axis=1)
    grounds = np.argmax(y_test, axis=1)
```

TheCode (trading-ksteps).py

```
            kPrev=kPrev+midPrice[i+stepK]
            kFollow=kFollow+midPrice[i+stepK]
        kPrev=kPrev/k
        kFollow=kFollow/k

        if kFollow>(kPrev+alpha):
            labels.append(2)
        elif kFollow<(kPrev-alpha):
            labels.append(0)
        else :
            labels.append(1)
    else:
        labels.append(1)


#print(labels)
print("Number of 0s: ", labels.count(0))
print("Number of 1s: ", labels.count(1))
print("Number of 2s: ", labels.count(2))


labels = np.asarray(labels)
labels = to_categorical(labels)


#TRAIN TEST VALIDATION SPLIT
#_____________________________________________________________

X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, shuffle = False)

_, X_val, _, y_val = train_test_split(X_train, y_train, test_size=0.2, shuffle = False)

#X_test, X_train, y_test, y_train = train_test_split(X, labels, test_size=0.6, shuffle = False)

#_, X_val, _, y_val = train_test_split(X_train, y_train, test_size=0.2, shuffle = False)


for run in range(1,11):

    model = load_model('./../Training/Output/Weights/k%dalpha%.2f/run%dk%dalpha%.2f.h5' % (k, alpha, run, k,

    evaluation = model.evaluate(x=X_test, y=y_test, batch_size=64, verbose=1, sample_weight=None, steps=None)

    print(evaluation)

    # plt.plot(history.history["acc"])
    # plt.plot(history.history["val_acc"])
    # plt.plot(history.history["loss"])
    # plt.plot(history.history["val_loss"])


    # Building the CONFUSION MATRIX
    #_____________________________________________________

    predictions = model.predict(X_test, batch_size=64, verbose=0, steps=None)
    predictions = np.argmax(predictions, axis=1)
    grounds = np.argmax(y_test, axis=1)

    print(grounds)
    print(predictions)
    print(midPrice)

    activeBuy1 = 0
    mp1 = 0
    activeBuy2 = 0
    mp2 = 0
    activeBuy3 = 0
    mp3 = 0
    totalProfit = 0

    activebuy = np.zeros(k)
    mp = np.zeros(k)


    maif = open('./kStepsTradingStrategy/k%dalpha%.2f/k%dalpha%.2frun%d.txt'%(k, alpha, k, alpha, run), 'w')


    for i in range(len(grounds)): # buy every step and sell in 3 steps from then
        profitquant = 0
        for rest in range(0,k):
            if i+k > (len(grounds)-3):
                break
            if i%k==rest:
                if activebuy[rest]:
                    #sell and see profit:
                    profitquant = midPrice[i] -mp[rest]
                    totalProfit = totalProfit +profitquant
                    mp[rest]=0
                    activebuy[rest] = 0
                if predictions[i-k]==2:
                    mp[rest] = midPrice[i]
                    activebuy[rest] = 1


        maif.write('profitquant = %.2f i-3 was %.2f now i is %.2f \t totalProfit is now: %.2f \n' % (profitqu


    ...

    plt.clf()

    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model train vs validation accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper right')
    plt.savefig('./Output/Run%d K%d alpha%.2f accuracy.png' %(run, k, alpha))
    plt.clf()

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model train vs validation loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper right')
    plt.savefig('./Output/Run%d K%d alpha%.2f loss.png' %(run, k, alpha))
    plt.clf()

        ...

    maif.close()



    K.clear_session()




# THIS IS THE END, my only friend, the end;
```

Manual for the submitted code:



(folder hierarchy)

In Model Training – the code for extracting features from the dataset can be found, together with the code for training the model, .txt files of the MCC coefficients and the code for plotting it, the two datasets saved as *npy* arrays + all the weights for the models I've trained. (one can also find *a jupyter notebook* version of training code in this folder)

In Trading Strategy – one can find code for plotting most of the graphs used in this project report, as well as the code for generating the results using the trained models from the precious point, as well as the results themselves in two sub-folders for the corresponding two trading strategies attempted.