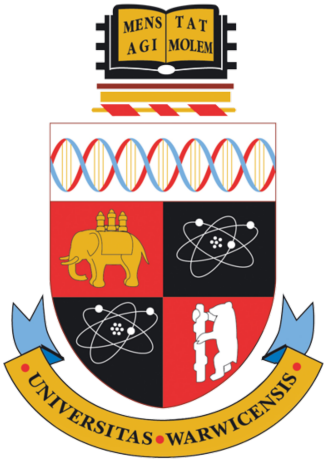

Predicting Bitcoin Prices - Can Machine Learning Algorithms Help?

Student: Kevin Südmersen (ID: 1791791)
Supervisor: Piotr Jelonek
Date: September 12, 2018
University of Warwick

Abstract

Predicting financial asset prices is difficult, because asset prices have a sizeable unpredictable component (Elliott and Timmermann, 2013) and because ongoing competition in the market makes it impossible to generate consistent profits with a strategy which has previously been successful (Lo, 2005). Economists currently do not have forecasting models which work well on non-stationary data with non-linear patterns, such as financial time series data. So, we tested whether three flexible machine learning algorithms[1] can help predict the prices of a highly non-stationary and non-linear financial time series, i.e. the Bitcoin closing price five minutes ahead, using eleven technical indicators. We found that RT, SVR and ANN outperformed a naïve benchmark in 6/10, 6/10 and 0/10[2] rolling windows respectively and that all algorithms performed worse on average. We suspect that RT, SVR and ANN under-performed on average, because we did not find the optimal combination of hyper-parameters and we recommend future researchers of this topic (i) to explore the the hyper-parameter space more thoroughly, (ii) to treat every arbitrarily set parameter as an additional hyper-parameter, (iii) to consider block-chain data as additional independent variables, (iv) to implement a trading strategy based on the predictions of the forecasting models, (v) to predict volatility instead of prices or (vi) to pursue a more passive investment strategy.

[1] Regression Trees (RT), Support Vector Regression (SVR), Artificial Neural Networks (ANN)
[2] The notation x/10 refers to "x out of ten"

1 Acknowledgements

First, I would like to thank my parents for always supporting my career choices and for paying this course. Second, I would like to thank my family, friends and girlfriend for bearing with me during the last year, I know this was not always easy. And finally, I would like to thank my thesis supervisor Piotr Jelonek, who gave me inspiration and constructive feedback, and my Econometrics tutor Terry Cheng who taught Econometrics in a very enthusiastic way.

Contents

2 Introduction

In this paper, we will investigate whether three flexible[3] machine learning (ML) algorithms, i.e. Regression Trees (RT), Support Vector Regression (SVR) and Artificial Neural Networks (ANN), can help predict Bitcoin closing prices five minutes ahead more accurately than a simple naïve benchmark. While the prediction accuracy[4] is our primary interest, we will also evaluate the practicability of these algorithms.

Bitcoin is an electronic peer-to-peer, decentralized network intended for online payments without the need for a third party (Nakamoto, 2008). Investigating the predictability of Bitcoin prices is of great interest for Bitcoin traders, since being able to predict Bitcoin prices comes with financial benefits and a deeper understanding of market efficiency. As figure 1 shows, Bitcoin's price series started to take off in the beginning of 2017 and reached its current all time high of $\$19,666$[5] on 17/12/2017[6]. After that, Bitcoin has been on a long-term down trend, but between 01/01/2017 and 12/06/2018, the absolute, average intra-day fluctuations were $496.74 which demonstrates that Bitcoin day traders can make a substantial profit, if the entry and exit is well timed.

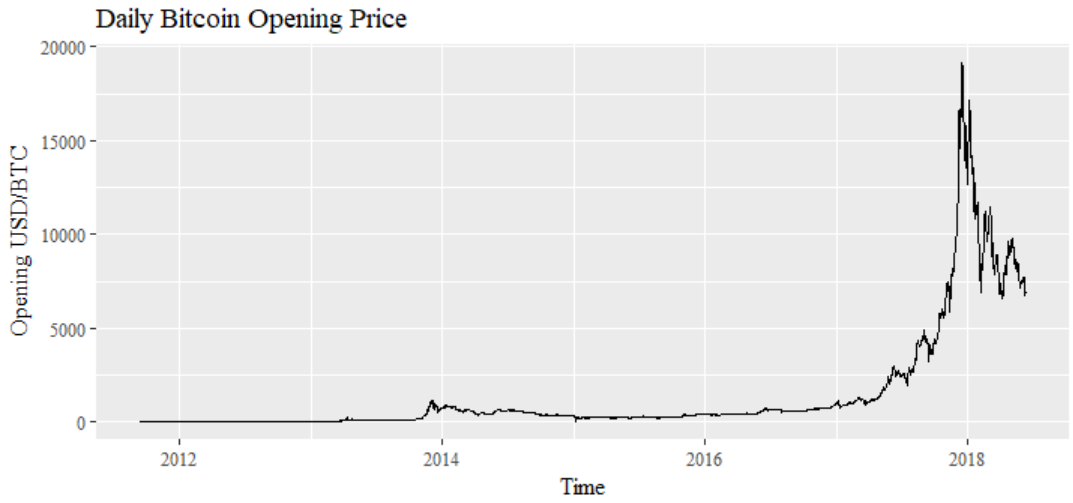

Figure 1: Daily Bitcoin Opening Price from 13/09/2011 - 12/06/2018 (quandl.com, 2018)

Traditionally, economists used autoregressive integrated moving average (ARIMA) time series forecasting models (Yeh *et al.*, 2011) and linear regression models (Elliott and Timmermann, 2013) to forecast future returns, but these methods were not very successful. ARIMA models are not applicable to non-stationary data and ARIMA as well as regression models are restricted to be linear in parameters. Since many prices of financial assets, like stocks and Bitcoin, do not move in a stationary and linear fashion (Wen *et al.*, 2010), there is clearly a need for models being able to capture non-linear patterns in non-stationary data.

[3]Flexible in the sense that there are no linearity restrictions in the model's parameters

[4]Sometimes, we will refer to this as prediction performance

[5]"$" refers to US Dollars

[6]We will use the notation day/month/year

There are two main approaches to asset price prediction, i.e. fundamental and technical analysis. Fundamental analysts estimate the intrinsic value of an asset considering its micro and macro environment, while technical analysts study historic price and volume data, try to identify trends and estimate future prices based on existing trends. Technical analysts assume that (i) anything affecting the value of an asset is already discounted in its price, that (ii) prices move in trends and that (iii) history repeats itself (Murphy, 1999). Clearly, a technical analysis has the main advantage that one only needs to gather price and volume data of the asset in question. This makes technical analysis far less time consuming than fundamental analysis and hence, this paper will follow the approach of technical analysis to train[7] machine learning algorithms.

Predicting asset prices is difficult, since asset price movements have a sizeable unpredictable component (Elliott and Timmermann, 2013). Additionally, if one successful forecasting model is discovered, it could sooner or later be copied by the whole market, causing asset prices to move in a way which eliminates the model's forecasting ability (Lo, 2005). Also, according to traditional economic theories, it should be impossible to create excess returns by investing in financial assets. The Random Walk Hypothesis (RWH) states that asset prices follow a random walk and that it is impossible to outperform market averages (Malkiel and McCue, 1985). The closely related Efficient Market Hypothesis (EMH) states that financial assets always trade at their fair value (Malkiel and Fama, 1970), meaning that it is impossible to buy undervalued stocks and to sell overvalued stocks. Yet, there are traders, e.g. Warren Buffet or George Soros, who have proven that markets are definitely not always efficient (Clarke *et al.*, 2001), and economists do not have a good explanation for that. All economists can say is that asset prices are the result of supply and demand.

3 Literature review

While there is a lot of *in-sample* evidence in favour of asset return predictability (Campbell, 2000), classical asset pricing models have performed very badly in *out-of-sample* tests[8] (Bossaerts and Hillion, 1999; Goyal and Welch, 2003; Welch and Goyal, 2007). High in-sample results should be viewed with scepticism, because one can easily increase the in-sample fit, i.e. increase the R-squared or reduce the in-sample mean squared error (MSE)[9], by adding additional variables to the model. Though, including too many and possibly irrelevant variables is likely to lead to *over-fitting* (James *et al.*, 2013) and if the model is over-fitted, it generalizes very badly to unseen out-of-sample test data, which is the problem of real interest (Campbell, 2008).

Moreover, Welch and Goyal (2007) show that conventional predictive regression models fail to consistently outperform a simple historical average forecast. They claim that inherent "model uncertainty and parameter instability render conventional predictive regression models unreliable". To improve the performance of conventional predictive regression models,

[7]Training ML algorithms refers to the process of estimating the algorithm's parameters

[8]In-sample data refers to the proportion of the dataset which is used for estimating the parameters of the model and out-of-sample data is used for evaluating the prediction performance of the model

[9]R-squared and the MSE both evaluate the goodness of fit of a model. The higher (lower) the R-squared (MSE), the better the fit

a number of adjustments, e.g. economically motivated model restrictions and regime shifts (Elliott and Timmermann, 2013), have been implemented, but it is still not clear to us, how these adjustments should be able to capture the inherent non-linearity and non-stationarity of asset price movements.

While ML algorithms can produce outstanding prediction performance, when properly trained (Ballings *et al.*, 2015; Patel *et al.*, 2015; Jang and Lee, 2018), researchers have little theoretical guidance on how to achieve good prediction performance and *must* make some arbitrary choices at some point. While it makes intuitive sense to choose flexible ML algorithms for non-stationary data with non-linear patterns, it is not guaranteed at all whether RT, SVR and ANN are the most suitable algorithms for predicting Bitcoin prices.

Choosing the improper algorithm may cause prediction performances to deteriorate. E.g., Ładyżyński *et al.* (2013) and Greaves and Au (2015) reported prediction performances which are hardly better than simple benchmarks. Despite different results in the literature, most authors were often using the same algorithms, such as Random Forests (RFs), Support Vector Machines (SVM) and ANN. These differences in prediction performance might be due to the improper setting of hyper-parameters[10] and due to using different datasets.

At the moment, there is also no theory that can tell one how to properly set the model's hyper-parameters, so typically researchers need to try many possible combinations of hyper-parameters and pick the combination which yield the best results. Thus, it could be that some researchers either had expert knowledge about the proper hyper-parameter setting, or that some had more powerful computers which allowed them to test more combinations of hyper-parameters.

Due to possibly long computation times of ML algorithms, it is very difficult to find the optimal hyper-parameters. Cherkassky and Ma (2004) propose an analytical approach to finding the best parameters, but it is understood that their approach relies on fitting an auxiliary model before fitting the actual model and their auxilary model also depends on its own hyper-parameters. Bergstra and Bengio (2012) propose to randomly select the hyper-parameters and found that this improves the performance of ML algorithms. This seems promising, but one still needs to manually specify a range from which values are randomly selected and then, it is obviously not guaranteed that the range was specified correctly.

Furthermore, Snoek *et al.* (2012) used a Bayesian approach to find the optimal hyper-parameters and also found promising results. It is understood that the authors created a model which maximizes the probability to yield an improved performance of the ML algorithm, when the next combination of hyper-parameters is tested. However, the Bayesian approach is computationally very expensive and its performance also depends on the choice of its own, independent hyper-parameters.

Yeh *et al.* (2011) propose an "automatic" way of selecting hyper-parameters. Their algorithm yields outstanding results, but also relies on some manual pre-specifications. The authors first manually set two out of three hyper-parameters to some fixed values and then let their algorithm find the optimal value of the third hyper-parameter from a range which they pre-specified. In addition to some manual pre-specification of hyper-parameters, their algorithm is also computationally very expensive.

Although many authors reported extremely high prediction performance, we noticed that

[10] A hyper-parameter is a parameter of a ML algorithm which needs to be manually specified

ML algorithms were rarely applied by economists and more often by computer scientists and mathematicians. Maybe this is because highly flexible ML algorithms are not as interpretable as classical regression models, because ML algorithms try to fit rather complicated functions with many parameters (James *et al.*, 2013), and the technicalities of these algorithms are not a traditional economic discipline.

Concluding, we can see that asset price prediction is very difficult and that there is a wide gap in prediction results. ML algorithms are worth exploring, since they are not restricted to be linear in parameters unlike classical regression models and since they can be applied in non-stationary data unlike traditional ARIMA models. However, there has been very little ground-breaking theoretical research about ML algorithms and hence, ML algorithms might perform very badly, if they are applied on unsuitable problems and if their hyper-parameters are improperly chosen. The optimal choice of algorithms and hyper-parameters is very challenging, since some ML algorithms are computationally expensive, and therefore, only few combinations of hyper-parameters can actually be tested. On top of that, a forecasting model is only successful, if it has not been widely adopted by the market yet, so it is unlikely that a model which has been very successful in the past will remain successful in the future.

4 Methodology

Forecasting models can either predict the class of the dependent variable, such as "up" and "down", or they can predict its numeric value. Predicting categories is referred to as *classification* and predicting numeric values is known as *regression*. This paper will use regression algorithms, because predicting numeric values might be more informative for investors than predicting price direction, as an investor might need to know whether his profits can cover certain fixed costs or transaction costs.

As mentioned above each of these algorithms have a number of hyper-parameters to *tune*[11], and we will perform *grid search* to find the optimal combination of hyper-parameters. I.e., we will specify a range of values for each hyper-parameter and then construct a grid with all possible combinations of hyper-parameter values. When performing grid search, we will take the heuristics recommended by Nielsen (2015) and by Hsu *et al.* (2003) into account. These authors recommend to size down the training and test data, to consider algorithms with as few hyper-parameters as possible and to vary hyper-parameters exponentially. After having found a range of parameters which gives good results, one can sub-divide this range into smaller intervals to find even better parameter values.

Base Algorithm: When training, tuning and evaluating each ML algorithm with its corresponding hyper-parameters, we will proceed as follows. We will sequentially divide the whole dataset into several overlapping rolling windows. Then, we will divide each window into a training and test set, where the training set is subdivided into a training subset and validation set[12]. We found that different training set sizes can have a substantial effect on prediction performance, so we treat the training set size as a separate hyper-parameter to tune. This seemed reasonable to us, because regimes of the Bitcoin price could have

[11]Tuning hyper-parameters refers to the process of finding the optimal values of these parameters

[12]When we refer to the training set, we mean the complete training set, i.e. the training subset *and* the validation set

different durations and a professional Bitcoin trader does not know a priori whether the current Bitcoin price is at the beginning, the middle or at the end of its current regime.

Thus, assuming that we are currently at time t, we fit the model with all pre-specified hyper-parameter combinations on the training subsets of all training sets ending at time t and starting at $t-200, t-400, ..., t-1000$, and evaluate the model's performance on the validation set. Then, we chose the hyper-parameter combination together with the optimal training set size, which yielded the best performance on the validation set. With the optimal hyper-parameter combination and training set size, we fit the model on the *whole* training set and evaluate its performance on the test set. After that, we move back in time by x units if x was the optimal size of the current training set. We will stop this process after 10 iterations, so with this approach, it may be the case that some of the oldest observations of the sampling period are not used.

The validation and test set size will only contain 10 observations, because we wanted to minimize the risk that the validation and test sets were already part of a new regime. Because we want to make predictions of y_{t+5}, the Bitcoin price 5 minutes ahead, we will train each model with the predictor values at time t and the Bitcoin prices at time $t+5$ and we will evaluate each model's performance by calculating the MSE. Assuming that each model estimates the function $f(\cdot)$ from the data, we will calculate the MSE on the validation and test sets as follows:

$$MSE = \frac{1}{10}\sum_{t=1}^{10}\left(f(\mathbf{x}_t) - y_{t+5}\right)^2,$$

where $\mathbf{x}_t = [x_{1,t}, x_{2,t}, ..., x_{p,t}]^T$ is a p-dimensional vector of an observation at time t. Figure 2 gives an overview of the procedure outlined above.

4.1 Regression Trees

RT divide the predictor space[13] of the training data into regions based on logical rules and only stop the splitting process when certain stopping criteria are reached. The prediction for each new observation $\mathbf{x}_t = [x_{1,t}, x_{2,t}, ..., x_{p,t}]^T$ is the average of the Bitcoin prices in the *terminal node*[14], in which $\mathbf{x}_t$ falls.

The splitting algorithm is called *binary recursive splitting* (BRS). BRS divides the predictor space $X_1, ..., X_p$ into J distinct and non-overlapping regions $R_1, R_2, ..., R_J$ and chooses the predictor X_j along with the splitting point s, such that splitting the predictor space into the two regions $R_1(j,s) = \{X|X_j < s\}$ and $R_2(j,s) = \{X|X_j \geq s\}$[15] yields the largest reduction in the residual sum of squares (RSS)[16] (James *et al.*, 2013). Thus, at each split,

[13] Assuming that we have p predictors, i.e. independent variables, the predictor space is a multi-dimensional coordinate system with p independent variables such as $X_1, X_2, ..., X_p$

[14] A terminal node is a sub-region which does not contain any further splits. See figure 3 for an example

[15] This notation refers to the region of the predictor space, where X_j takes a value greater than or equal to s

[16] RSS is a metric for evaluating a model's goodness of fit. The lower the RSS, the better the fit

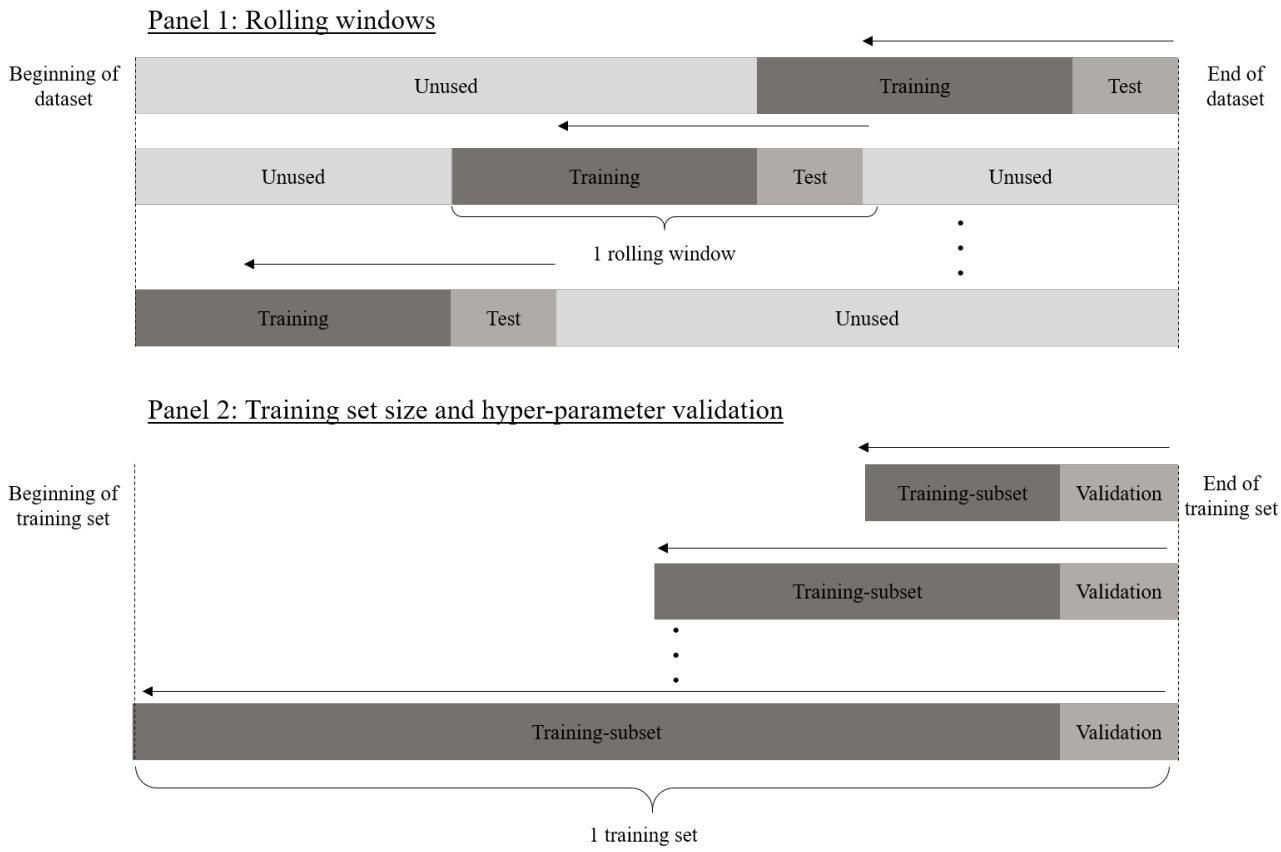

Figure 2: Panel 1: The rolling window method starting from the most recent observations and ending at one of the oldest observations. Panel 2: Validating different parameter combinations and training set sizes of different length. Note that panel 2 *zooms-in* on the training set of panel 1 and that each window has an overlap equal to the number of observations in the test set

the goal is to find a predictor X_j and a cutting point s such that

$$\sum_{t:\mathbf{x}_t \in R_1(j,s)} (y_{t+5} - \hat{y}_{R_1})^2 + \sum_{t:\mathbf{x}_t \in R_2(j,s)} (y_{t+5} - \hat{y}_{R_2})^2 \tag{1}$$

yields the lowest possible value, i.e. the lowest possible RSS. $\hat{y}_{R_1}$ is the mean of Bitcoin prices at time $t+5$, whose corresponding $\mathbf{x}_t$ falls into region $R_1(j,s)$ and $\hat{y}_{R_2}$ is the mean of Bitcoin prices at time $t+5$ whose corresponding $\mathbf{x}_t$ falls into region $R_2(j,s)$[17].

Next, this process is repeated *within* each of the sub-regions that were just created, i.e. within $R_1(j,s)$ and $R_2(j,s)$, and the process only stops until each terminal node, has some pre-specified number of observations left in it. This pre-specified number is another hyper-parameter to tune and we will refer to this hyper-parameter as *minbucket*.

[17]Remember that we are training the models with predictor values at time t and Bitcoin prices at time $t+5$

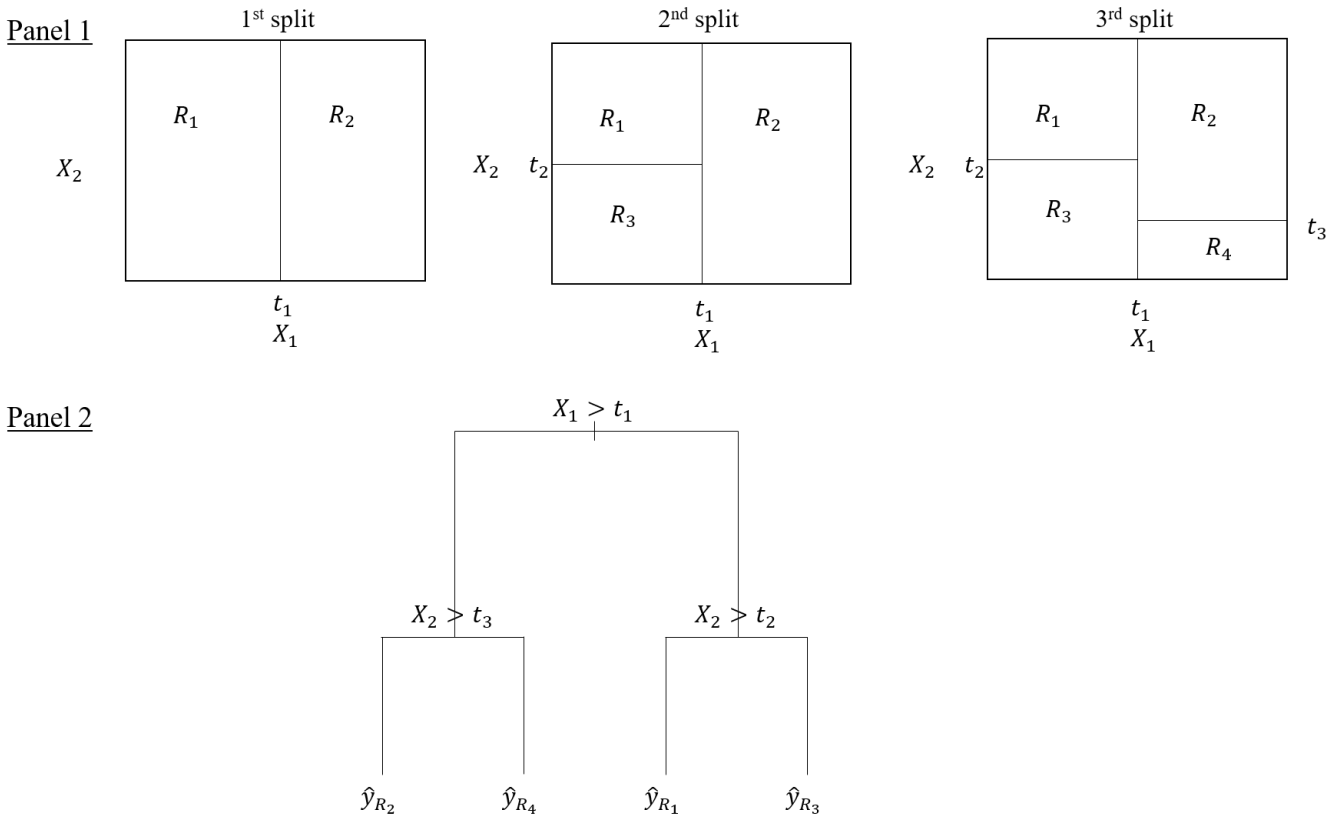

Figure 3: Panel 1: The first split with cutting point t_1 divides the predictor space X_1, X_2 into two regions, R_1, R_2. The second split with cutting point t_2 divides region R_1 into two sub-regions and the third split with cutting point t_3 divides region R_2 into 2 sub-regions. The terminal nodes of the tree are R_1, R_2, R_3, R_4 (based on (James *et al.*, 2013)). Panel 2: Visualization of panel 1 as a tree. $\hat{y}_{R_1}, \hat{y}_{R_2}, \hat{y}_{R_3}, \hat{y}_{R_4}$ are the average Bitcoin prices at time $t+5$ of each terminal node, i.e. predictions of each terminal node.

Technically, it is possible to create a tree with terminal nodes of only one observation left in it. This model would perfectly fit the training data, but generalize very poorly to the validation and test data, i.e. it would terribly over-fit (James *et al.*, 2013). To prevent over-fitting, one can add a penalty term to the objective function (OF) of RT, which penalizes the model for having many terminal nodes, i.e. for constructing very complex trees. Then, the OF of RT becomes:

$$\text{minimize} \sum_{m=1}^{|T|} \sum_{t:\mathbf{x}_t \in R_m} (y_{t+5} - \hat{y}_{R_m})^2 + \alpha|T|, \tag{2}$$

where $|T|$ is the number of terminal nodes in the tree, R_m is the subset of the predictor space corresponding to the m-th terminal node, $\hat{y}_{R_m}$ is the mean of the Bitcoin prices in terminal node m, and α is the hyper-parameter which controls the number of terminal nodes in the tree, also called *cost complexity* parameter (CP) (James *et al.*, 2013).

So, RT will fit a function $f(\cdot)$ of the following form to predict Bitcoin closing prices five minutes ahead:

$$\hat{y}_{t+5} = f(\mathbf{x}_t) = \sum_{m=1}^{|T|} \hat{y}_{R_m} \mathbb{1}(\mathbf{x}_t \in R_m),$$

where

$$\mathbb{1}(\mathbf{x}_t \in R_m) = \begin{cases} 1, & \text{if } \mathbf{x}_t \in R_m \\ 0, & \text{otherwise} \end{cases}$$

Summarizing, during training, a tree with splitting rules and terminal nodes is being constructed. During validation and testing, these splitting rules and the values in the terminal nodes will remain constant. So, to predict $\hat{y}_{t+5}$, one simply needs to retrieve the value of the terminal node in which $\mathbf{x}_t$ falls.

4.2 Support Vector Regression

The SVR algorithm enlarges the predictor space using *kernels*[18] and then performs linear regression in the enlarged predictor space (Smola and Schölkopf, 2004), which we will refer to as *feature space* in this section. The goal is to find a function which has at most ϵ deviation from the target values y_{t+5} for all $t = 1, ..., n$[19] and is as flat as possible (Smola and Schölkopf, 2004). We will first explain this process for linear functions, as it is easy to extend this problem to the non-linear case.

Because it has yielded empirically better results (Yeh *et al.*, 2011; Wen *et al.*, 2010), we will first scale the data into the range $[0, 1]$. Every observation $\mathbf{x}_t = [x_{1,t}, x_{2,t}, ..., x_{p,t}]^T$ of predictor X_j is scaled as follows:

$$\mathbf{x}_t^s = \frac{\mathbf{x}_t - \min(X_j)}{\max(X_j) - \min(X_j)}, \tag{3}$$

and similarly, every training observation of y_{t+5} is scaled as follows[20]:

$$y_{t+5}^s = \frac{y_{t+5} - \min(Y_{t+5})}{\max(Y_{t+5}) - \min(Y_{t+5})}, \tag{4}$$

where Y_{t+5} is the complete time series of Bitcoin closing prices at $t + 5$.

It can be shown that the linear SVR function can be fully represented with the inner products $\langle \cdot\ ,\ \cdot \rangle$:

$$f(\mathbf{x}_t^s) = \langle \mathbf{w}, \mathbf{x}_t^s \rangle + b \text{ with } \mathbf{w} \in \mathbf{X}, b \in \mathbb{R}, \tag{5}$$

[18] In short, a kernel function is a function that quantifies the similarity between two training observations. How enlarging the predictor space works is discussed further below

[19] n refers to the number of observations in the training set

[20] Note that y_{t+5} only needs to be scaled while training the model, whereas the observations of each predictor need to be scaled while training, validating and testing the model

where $\mathbf{w} = [w_1, w_2, ..., w_p]^T$ is a weight vector and where $\mathbf{X}$ denotes the p-dimensional predictor space (Smola and Schölkopf, 2004).

Finding the flattest possible $f(\mathbf{x}_t^s)$ is equivalent to finding the minimum value of $\mathbf{w}$ which can be achieved by minimizing its squared norm, i.e. $||\mathbf{w}||^2 = \langle \mathbf{w}, \mathbf{w} \rangle$. As it is sometimes infeasible to find a function with at most ϵ deviation from all target values y_{t+5}^s, the slack parameters ξ and ξ^* are introduced, for training observations where $|f(\mathbf{x}_t^s) - y_{t+5}^s| \geq \epsilon$. ξ is used for observations above $f(\mathbf{x}_t^s)$ and ξ^* is used for observations below $f(\mathbf{x}_t^s)$ (Smola and Schölkopf, 2004). ξ and ξ^* are defined by the ϵ - insensitive loss function:

$$|\xi|_\epsilon = \begin{cases} 0, & \text{if } |\xi| \leq \epsilon \\ |\xi| - \epsilon, & \text{otherwise} \end{cases} \tag{6}$$

Now, one can formulate this as a convex optimization problem:

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{t=1}^{n}(\xi_t + \xi_t^*) \\ \text{subject to} \quad & y_{t+5}^s - \langle \mathbf{w}, \mathbf{x}_t^s \rangle - b \leq \epsilon + \xi_t, \\ & \langle \mathbf{w}, \mathbf{x}_t^s \rangle + b - y_{t+5}^s \leq \epsilon + \xi_t^*, \\ & \xi_t, \xi_t^* \geq 0, \text{ for } t = 1, ..., n, \end{aligned} \tag{7}$$

where the regularization term C controls the trade-off between the flatness of $f(\mathbf{x}_t^s)$ and the maximum number observations which deviate by more than ϵ, i.e. $|f(\mathbf{x}_t^s) - y_{t+5}^s| \geq \epsilon$ (Smola and Schölkopf, 2004). C and ϵ are both hyper-parameters which we will tune using the "Base Algorithm" described earlier.

The optimization problem in (7) can be solved by rearranging the constraints and setting up a Lagrangian introducing the Lagrange multipliers η_t, η_t^*, α_t, α_t^*:

$$\begin{aligned} L = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{t=1}^{n}(\xi_t + \xi_t^*) - \sum_{t=1}^{n}(\eta_t\xi_t + \eta_t^*\xi_t^*) \\ - \sum_{t=1}^{n}(\alpha_t(\epsilon + \xi_t - y_{t+5}^s + \langle \mathbf{w}, \mathbf{x}_t^s \rangle + b)), \\ - \sum_{t=1}^{n}(\alpha_t^*(\epsilon + \xi_t^* + y_{t+5}^s - \langle \mathbf{w}, \mathbf{x}_t^s \rangle - b)) \end{aligned} \tag{8}$$

where $\alpha_t^{(*)}, \eta_t^{(*)} \geq 0$[21] (Smola and Schölkopf, 2004).

It can be shown that (8) has a saddle point with respect to (w.r.t.) the primal variables, $\mathbf{w}, b, \xi_t, \xi_t^*$ and w.r.t. the dual variables, $\alpha_t^{(*)}, \eta_t^{(*)}$ (Smola and Schölkopf, 2004). Therefore, the partial derivatives of L w.r.t. the primal variables shown below have to vanish for optimality:

$$\frac{\partial L}{\partial b} = \sum_{t=1}^{n}(\alpha_t^* - \alpha_t) = 0 \tag{9}$$

[21]For notational ease, $\alpha_t^{(*)}$ refers to α_t and α_t^*, $\eta_t^{(*)}$ refers to η_t and η_t^*, and $\xi_t^{(*)}$ refers to ξ_t and ξ_t^*

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{t=1}^{n}(\alpha_t^* - \alpha_t)\mathbf{x}_t^s = 0 \tag{10}$$

$$\frac{\partial L}{\partial \xi_t^{(*)}} = C - \alpha_t^{(*)} - \eta_t^{(*)} = 0 \tag{11}$$

Substituting (9), (10) and (11) back into (8) yields the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \begin{cases} -\frac{1}{2}\sum_{t,j=1}^{n}(\alpha_t - \alpha_t^*)(\alpha_j - \alpha_j^*)\langle \mathbf{x}_t^s, \mathbf{x}_j^s \rangle \\ -\epsilon\sum_{t=1}^{n}(\alpha_t - \alpha_t^*) + \sum_{t=1}^{n} y_{t+5}^s(\alpha_t - \alpha_t^*) \end{cases} \\ \text{subject to} \quad & \sum_{t=1}^{n}(\alpha_t - \alpha_t^*) = 0, \\ & \alpha_t, \alpha_t^* \in [0, C], \end{aligned} \tag{12}$$

where the Lagrange Multipliers $\eta_t^{(*)}$ were eliminated, because equation (11) could be restated as $\eta_t^{(*)} = C - \alpha_t^{(*)}$ (Smola and Schölkopf, 2004).

From equation (10), it follows that:

$$\mathbf{w} = \sum_{t=1}^{n}(\alpha_t^* - \alpha_t)\mathbf{x}_t^s, \tag{13}$$

which can be substituted back into (5) giving us the linear SVR function:

$$f(\mathbf{x}_t^s) = \sum_{t=1}^{n}(\alpha_t - \alpha_t^*)\langle \mathbf{x}_t^s, \mathbf{x}_{t'}^s \rangle + b, \tag{14}$$

where $\mathbf{x}_{t'}^s$ refers to all observations other than $\mathbf{x}_t^s$, and where

$$b = y_k + \epsilon - \sum_{t=1}^{n}(\alpha_t - \alpha_t^*)\langle \mathbf{x}_t^s, \mathbf{x}_k^s \rangle$$

is obtained from any α_k^* with $0 < \alpha_k^* < C$ (Yeh *et al.*, 2011). (14) is the so called *Support Vector Expansion*, which shows that $\mathbf{w}$ can be formulated as a linear combination of the training observation $\mathbf{x}_t^s$ and that it is not necessary to compute $\mathbf{w}$ explicitly (Smola and Schölkopf, 2004).

To estimate the parameters $(\alpha_t - \alpha_t^*)$ and b, we need the dot products $\langle \mathbf{x}_t^s, \mathbf{x}_{t'}^s \rangle$ between all pairs of training observations, that is between $n(n-1)/2$ pairs. However, it turns out that only for training observations where $|f(\mathbf{x}_t^s) - y_{t+5}^s| \geq \epsilon$, the Lagrange multipliers $(\alpha_t - \alpha_t^*)$ are non-zero (Smola and Schölkopf, 2004). These training observations are called *support vectors*. So, when we want to evaluate $f(\mathbf{x}_t^s)$ from (14) for a new observation of the validation or test set, we would only need to calculate:

$$f(\mathbf{x}_t^s) = \sum_{i \in S}(\alpha_t - \alpha_t^*)\langle \mathbf{x}_t^s, \mathbf{x}_{t'}^s \rangle + b, \tag{15}$$

where S are the indices of the support vectors (James *et al.*, 2013).

To estimate a non-linear $f(\mathbf{x}_t^s)$, we first map the p-dimensional predictor space $\mathbf{X} = \{X_1, ..., X_p\}$ into some higher, say $p + d$ dimensional feature space $\mathbf{F} = \{F_1, ..., F_{p+d}\}$ by a map Φ, i.e. $\Phi : \mathbf{X} \rightarrow \mathbf{F}$ (Smola and Schölkopf, 2004). It is understood that $\mathbf{X} \in \mathbb{R}^p$ and $\mathbf{F} \in \mathbb{R}^{p+d}$ and that mapping the predictor space into higher dimensional feature space (i.e. *enlarging* the predictor space) works as follows: E.g. one could enlarge $\mathbf{X} = \{X_1, ..., X_p\}$ by adding quadratic terms of each X_j in which case the feature space would become $\mathbf{F} = \{X_1, X_1^2, ..., X_p, X_p^2\}$, one could also add interaction terms, in which case $\mathbf{F} = \{X_1, X_1^2, X_1 \times X_2, ..., X_p, X_p^2, X_{p-1} \times X_p\}$, etc. It is easy to see that one could endlessly enlarge the predictor space to fit ever more complicated functions, but, this approach is computationally very expensive due to additional parameters to be estimated for each additional F_j (James *et al.*, 2013).

Kernels enlarge the predictor space in a computationally efficient way. It can be shown that a kernel function is defined as the inner product between observation t and all other observations mapped into the feature space, i.e. $K(\mathbf{x}_t^s, \mathbf{x}_{t'}^s) = \langle \Phi(\mathbf{x}_t^s), \Phi(\mathbf{x}_{t'}^s) \rangle$ (Smola and Schölkopf, 2004). The most widely used kernel function is the radial basis function (RBF) (Yeh *et al.*, 2011), which we will use, because it has only one hyper-parameter. The RBF is defined as follows:

$$K(\mathbf{x}_t^s, \mathbf{x}_{t'}^s) = \exp(-\gamma ||\mathbf{x}_t^s - \mathbf{x}_{t'}^s||^2), \tag{16}$$

where γ is the non-negative width parameter of the RBF kernel, which we will tune using the "Base Algorithm". It is understood that the RBF also has to be calculated between all $n(n-1)/2$ pairs of training observations.

After the mapping, one can perform the same regression algorithm as above, i.e. perform linear SVR in the higher dimensional feature space $\mathbf{F}$. It can be shown that the steps from equation (5) to (15) remain exactly the same, with the only difference that the inner product $\langle \mathbf{x}_t^s, \mathbf{x}_j^s \rangle$ in (12) is replaced by the RBF kernel $K(\mathbf{x}_t^s, \mathbf{x}_j^s)$ leading to the following result (Smola and Schölkopf, 2004):

$$f(\mathbf{x}_t^s) = \sum_{i \in S} (\alpha_t - \alpha_t^*) K(\mathbf{x}_t^s, \mathbf{x}_{t'}^s) + b, \tag{17}$$

where

$$b = y_k + \epsilon - \sum_{t=1}^{n} (\alpha_t - \alpha_t^*) K(\mathbf{x}_t^s, \mathbf{x}_k^s)$$

is obtained from any α_k^* with $0 < \alpha_k^* < C$ (Yeh *et al.*, 2011).

After having estimated $f(\mathbf{x}_t^s)$ from (17), we will first make the prediction of the scaled values, i.e. $f(\mathbf{x}_t^s) = \hat{y}_{t+5}^s$, and then scale them back to make the actual predictions for the Bitcoin prices in the validation and test set, i.e.

$$\hat{y}_{t+5} = \hat{y}_{t+5}^s \times (\max(Y_{t+5}) - \min(Y_{t+5})) + \min(Y_{t+5}), \tag{18}$$

4.3 Artificial Neural Networks

ANNs consist of an input layer, at least one hidden layer and one output layer. Each layer consists of neurons with activation values interconnected by weights. The number of input neurons will be equal to the number of predictors and the number of output neurons will be one for regressions and equal to the number of unique categories for classifications. The number of hidden layers and the number of neurons in each hidden layer are hyper-parameters (Nielsen, 2015).

In the training phase, the weights and biases are iteratively adjusted so that the difference between the network's output values and target values converges to zero. The weights and biases in the network are adjusted with the *back-propagation* algorithm and the gradient descent method, which we will discuss in depth below, in order to approximate the minimum of a certain loss function (Nielsen, 2015). Figure 4 gives an overview of a simple network with two hidden layers.

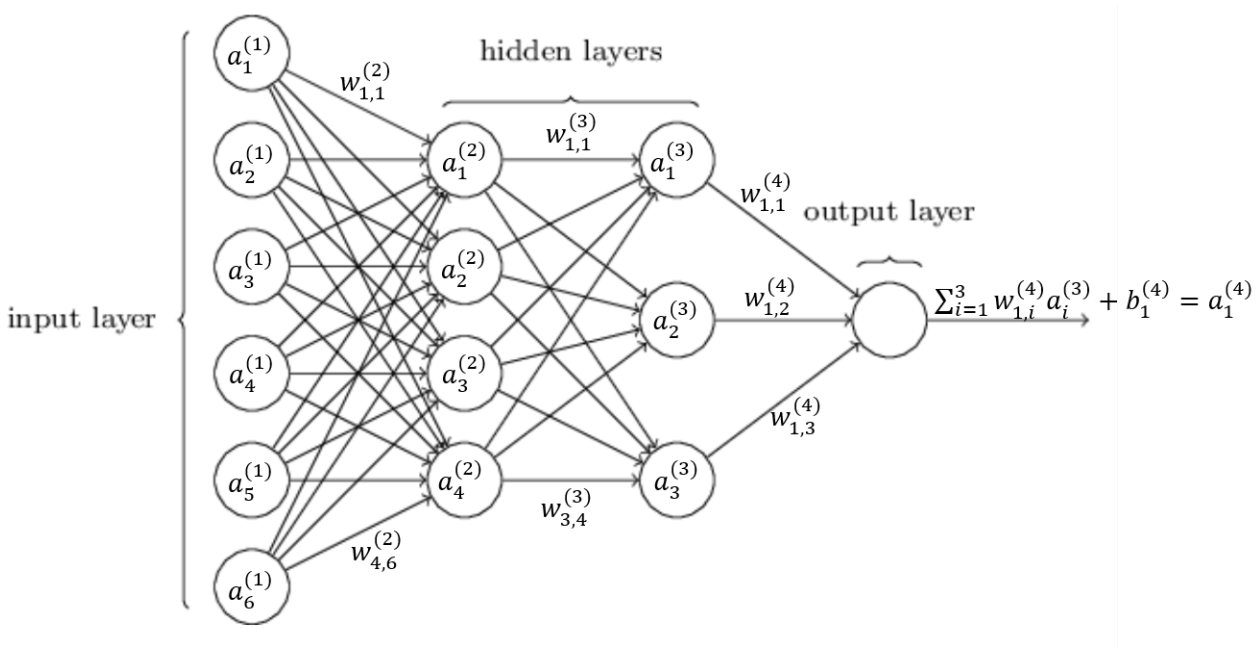

Figure 4: An ANN for a regression problem (Nielsen, 2015), edited for illustrative purposes

4.3.1 Forward propagation

The flow of training observation $\mathbf{x}_t = [x_{1,t}, x_{2,t}, ..., x_{p,t}]^T$ for $t = 1, ..., n$ through the network is as follows. First, all predictors are scaled as described by (3) and all training observations of y_{t+5} are scaled according to (4). After the scaled $\mathbf{x}_t^s$ is fed into the input layer of the network, the activation value of the j-th neuron in the l-th layer, i.e. $a_j^l(\mathbf{x}_t^s)$, is related to

the neurons in the $(l-1)$-th layer in the following way:

$$a_j^l(\mathbf{x}_t^s) = f\left(\sum_{k=1}^{K} w_{jk}^l a_k^{l-1} + b_j^l;\ \mathbf{x}_t^s\right), \tag{19}$$

where the notation $a_j^l(\mathbf{x}_t^s)$ is not a product, but merely denotes that the value of a_j^l is dependent on $\mathbf{x}_t^s$. (19) shows that each neuron is the result of the *weighted input* $z_j^l(\mathbf{x}_t^s) = \sum_{k=1}^{K} w_{jk}^l a_k^{l-1} + b_j^l$, plugged into some non-linear activation function $f(\cdot)$. The sum is over all K neurons in the $(l-1)$-th layer[22], w_{jk}^l represents the weight connecting the j-th neuron in layer l with the k-th neuron in layer $l-1$, and b_j^l represents the bias term of the j-th neuron in layer l. We can also express the collection of all J neurons in layer l in vectorized form as follows:

$$\mathbf{a}^l(\mathbf{x}_t^s) = f(\mathbf{w}^l\mathbf{a}^{l-1} + \mathbf{b}^l;\ \mathbf{x}_t^s), \tag{20}$$

where

$$\mathbf{a}^l = \begin{bmatrix} a_1^l \\ a_2^l \\ \vdots \\ a_J^l \end{bmatrix}, \mathbf{w}^l = \begin{bmatrix} w_{1,1}^l & w_{1,2}^l & \dots & w_{1,K}^l \\ w_{2,1}^l & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ w_{J,1}^l & \dots & \dots & w_{J,K}^l \end{bmatrix}, \mathbf{a}^{l-1} = \begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \\ \vdots \\ a_K^{l-1} \end{bmatrix}, \mathbf{b}^l = \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_J^l \end{bmatrix},$$

and where J represents all neurons in layer l and K all neurons in layer $l-1$.

After training observation $\mathbf{x}_t^s$ has been *forward propagated* from layer to layer through the network[23], the activation value of the one and only output neuron $a^L(\mathbf{x}_t^s)$ is calculated as follows:

$$a^L(\mathbf{x}_t^s) = g\left(\sum_{k=1}^{K} w_{jk}^L a_k^{L-1} + b_j^L;\ \mathbf{x}_t^s\right), \tag{21}$$

where L is the output layer, and the activation function $g(.)$ is a linear function, i.e. $g(x) = x$. Note that in regression, the activation function in the output layer is different from the one in the other layers.

4.3.2 Cost function

After computing $a^L(\mathbf{x}_t^s)$, a certain cost such as the squared error, can be evaluated:

$$C(\mathbf{x}_t^s) = \frac{1}{2}\left(a^L(\mathbf{x}_t^s) - y_{t+5}^s\right)^2, \tag{22}$$

where y_{t+5}^s is the actual, scaled Bitcoin price at $t+5$. The procedure from (19) to (22) is repeated for each training observation, so ANNs are trying to minimize a re-scaled version of the MSE:

$$C(\mathbf{x}_1^s, ..., \mathbf{x}_n^s) = \frac{1}{2n}\sum_{t=1}^{n}\left(a^L(\mathbf{x}_t^s) - y_{t+5}^s\right)^2, \tag{23}$$

[22]Note that the number of layers per layer might vary

[23]I.e. After training observation $\mathbf{x}_t^s$ has been recursively plugged into (19) until the final layer is reached

4.3.3 Gradient descent

Every time $C(\mathbf{x}_t^s)$ is calculated, ANNs are trying to figure out how to adjust the weights and biases in the network to yield the largest reduction in $C(\mathbf{x}_1^s, ..., \mathbf{x}_n^s)$. This is done by calculating the gradient for each training observation, averaging it over all training observations, and then, applying the gradient descent update rule (Nielsen, 2015). The gradient for $C(\mathbf{x}_t^s)$ is defined as follows:

$$\nabla C(\mathbf{x}_t^s) = \left[\frac{\partial C}{\partial \mathbf{w}}, \frac{\partial C}{\partial \mathbf{b}}\right]^T, \tag{24}$$

where $\mathbf{w}$ denotes the collection of all weights and $\mathbf{b}$ denotes the collection of all biases in the entire network[24].

$\nabla C(\mathbf{x}_t^s)$ is then averaged as follows:

$$\nabla \bar{C}(\mathbf{x}_1^s, ..., \mathbf{x}_n^s) = \frac{1}{n}\sum_{t=1}^{n} \nabla C(\mathbf{x}_t^s), \tag{25}$$

where $\nabla \bar{C}(\mathbf{x}_1^s, ..., \mathbf{x}_n^s)$ now includes the *averaged desired changes* of all weights and biases in the network to achieve the most significant decrease in (23). In other words, now we know how to adjust the weights and biases in the network to decrease the value of the cost function.

Suppose $\nabla \bar{C}(\mathbf{x}_1^s, ..., \mathbf{x}_n^s)$ has the following components:

$$\nabla \bar{C}(\mathbf{x}_1^s, ..., \mathbf{x}_n^s) = \left[\frac{\partial C}{\partial \bar{\mathbf{w}}}, \frac{\partial C}{\partial \bar{\mathbf{b}}}\right]^T, \tag{26}$$

where $\partial C/\partial \bar{\mathbf{w}}$ and $\partial C/\partial \bar{\mathbf{b}}$ represent the collection of averaged desired changes of the weights and biases in the entire network. Based on the components of the averaged gradient, one can apply the gradient descent update rules to nudge the weights and biases as follows:

$$\mathbf{w}^{new} \leftarrow \mathbf{w}^{old} - \eta \frac{\partial C}{\partial \bar{\mathbf{w}}}, \tag{27}$$

and:

$$\mathbf{b}^{new} \leftarrow \mathbf{b}^{old} - \eta \frac{\partial C}{\partial \bar{\mathbf{b}}}, \tag{28}$$

where η is the *learning rate*, another hyper-parameter, proportional to the step size of the gradient descent. If η is too low, it might take too long to reach the minimum and if η is too large, the gradient descent step might overshoot the minimum of (23).

The gradient descent update rules (27) and (28) can be applied after all training examples have been forward and back-propagated through the network, i.e. after one *epoch* has passed. However, if the network has hundreds of thousands or even millions of weights and biases, it might be hard for the computer to store all their gradients into memory and in addition to that, the network is likely to learn slowly. So, in practice chunks or *batches* of training examples are fed into the network and after each batch has been forward and back-propagated

[24]Note that the weights and biases have been initialized with random values

through the network, the gradient descent update rules are applied, i.e. one gradient descent step has been taken. This technique is called *stochastic gradient descent*.

In figure 5, each gradient descent step is illustrated by one black arrow and the star represents the initial value of (23). For a good approximation of the minimum of (23), the number of epochs should be chosen sufficiently large to reach the minimum of (23) and sufficiently low to prevent over-fitting. The number of epochs and the batch size are also hyper-parameters.

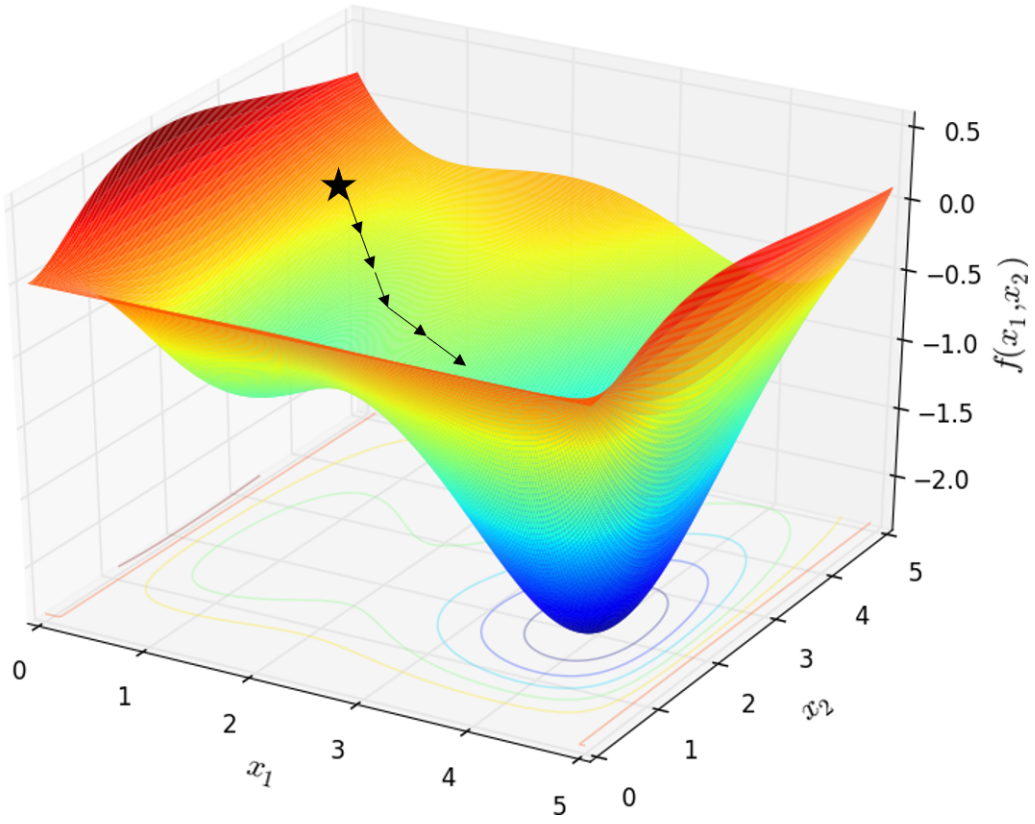

Figure 5: An example of the gradient descent. The star denotes some starting point and the arrows shall illustrate the iterative approximation of the functions' minimum. The length of each arrow is comparable to the learning rate (xpertup.com, 2018)

4.3.4 Back-propagation

The algorithm for computing all partial derivatives in each gradient $\nabla C(\mathbf{x}_t^s)$ is called the back-propagation algorithm which basically calculates all partial derivatives $\partial C/\partial \mathbf{w}$ and $\partial C/\partial \mathbf{b}$ using the chain rule. Computing all partial derivatives of the network can be achieved by computing the *error* at the output layer δ^L and then propagating it back trough the network by recursively computing the vectorized errors at the previous layers up to the second layer, i.e. $\boldsymbol{\delta}^{L-1}, \boldsymbol{\delta}^{L-1}, ..., \boldsymbol{\delta}^2$.

It can be shown that δ^L is calculated as follows:

$$\delta^L = \frac{\partial C}{\partial a^L(\mathbf{x}_t^s)}\ f'(z^L), \tag{29}$$

where $\frac{\partial C}{\partial a^L(\mathbf{x}_t^s)}$ measures how fast the value of the cost function changes w.r.t. a change in a^L, and where $f'(z^L)$ measures how fast the value of a^L changes w.r.t. changes in the weighted input z^L (Nielsen, 2015). From equation 22 we know that $\frac{\partial C}{\partial a^L(\mathbf{x}_t^s)} = \big(a^L(\mathbf{x}_t^s) - y_{t+5}^s\big)$, which is very easy to calculate once $\mathbf{x}_t^s$ has been forward propagated through the network. Also, $f'(z^L)$ should not cause too much computational overhead if $f(\cdot)$ is known[25]. So, we can rewrite equation 29 as follows:

$$\delta^L = \big(a^L(\mathbf{x}_t^s) - y_{t+5}^s\big)\ \ f'(z^L) \tag{30}$$

Second, since we need to propagate δ^L backwards through the network, we need an expression that relates δ^L to the error in the previous layer, i.e. $\boldsymbol{\delta}^{L-1}$. It can be shown that the following difference equation does exactly this and holds for any two connecting layers in the network:

$$\boldsymbol{\delta}^{L-1} = (\mathbf{w}^L)^T \delta^L \odot f'(\mathbf{z}^{L-1}), \tag{31}$$

where $\odot$ denotes the Hadamard product (Nielsen, 2015).

In the first back-propagation step, δ^L from equation 30 is plugged into equation 31, which yields the following result:

$$\boldsymbol{\delta}^{L-1} = (\mathbf{w}^L)^T \left[(a^L(\mathbf{x}_t^s) - y_{t+5}^s)\ f'(z^L)\right] \odot f'(\mathbf{z}^{L-1}) \tag{32}$$

Then, we can just iterate equation 31 backward which gives the following result:

$$\boldsymbol{\delta}^{L-2} = (\mathbf{w}^{L-1})^T \boldsymbol{\delta}^{L-1} \odot f'(\mathbf{z}^{L-2}), \tag{33}$$

and then, we can plug $\boldsymbol{\delta}^{L-1}$ from equation 32 into equation 33 and keep repeating this recursive substitution all the way back to layer 2.

Intuitively, by multiplying the transpose weight matrix $(\mathbf{w}^{l+1})^T$ with $\boldsymbol{\delta}^{l+1}$, we are propagating the error of the previous layer backward through the network which gives us some error measurement at the l-th layer. Then, by multiplying the result of that by the Hadamard product $\odot f'(\mathbf{z}^l)$, we are squashing the error through the activation function at layer l giving us the error of the weighted input to layer l (Nielsen, 2015).

The only thing we still need to explain about the back-propagation algorithm is how to relate δ^L and $\boldsymbol{\delta}^{L-1}$ to the partial derivatives $\frac{\partial C}{\partial \mathbf{w}^l}$ and $\frac{\partial C}{\partial \mathbf{b}^l}$ for any layer l in the network. It can be shown that

$$\frac{\partial C}{\partial \mathbf{b}^l} = \boldsymbol{\delta}^l, \tag{34}$$

[25]Actually, $f'(z^L)$ is very trivial in our case since we will be using the ReLU activation function as we will describe further below. We are just not using it here, because we want to show that (29) holds for any activation function $f(\cdot)$, meaning that we could actually *design* an activation function with desirable properties

and that

$$\frac{\partial C}{\partial \mathbf{w}^l} = \mathbf{a}^{l-1}\boldsymbol{\delta}^l, \tag{35}$$

where it is understood that $\mathbf{b}^l$ represents all biases at layer l and $\mathbf{w}^l$ denotes the weight matrix between layer l and $l-1$. This is great news, because $\boldsymbol{\delta}^l$ and $\mathbf{a}^{l-1}$ are all values we have already computed at this point. Now, that we have shown how to calculate actual values for all partial derivatives in the entire network, the discussion of the back-propagation algorithm is concluded.

4.3.5 Regularization

To prevent over-fitting, we will use the L1 (Lasso) regularization term, because this will shrink some weights exactly to zero and cause the weights of the network to concentrate in a small number of high-importance connections[26] (Nielsen, 2015). The MSE plus the L1 regularization term becomes:

$$C(\mathbf{x}_1^s, ..., \mathbf{x}_n^s) = \frac{1}{n}\sum_{t=1}^{n}\left(a^L(\mathbf{x}_t^s) - y_{t+5}^s\right)^2 + \frac{\lambda}{n}\sum_{w}|w|, \tag{36}$$

where sum $\sum_w$ is taken over all the weights in the network and where λ is another hyper-parameter (Nielsen, 2015). Note that when computing the gradient, the partial derivative w.r.t. a weight, i.e. $\partial|w|/\partial w$ is defined to be zero, if $|w| = 0$.

4.3.6 Activation function

Finally, we will describe how to choose the activation function $f(\cdot)$. Choosing the appropriate function is difficult, because day traders need to find a good balance between computational speed and prediction accuracy. We will use the *Rectified Linear Unit* (ReLU) function, because it can be computed approximately six times faster than other activation functions, such as the sigmoid or tanh function (Pan and Srikumar, 2016) and it has empirically shown accurate results (Nair and Hinton, 2010; Krizhevsky *et al.*, 2012; Glorot *et al.*, 2011; Jarrett *et al.*, 2009). The ReLU is defined as follows:

$$f(x) = \max(0, x), \tag{37}$$

and its derivative is defined as such:

$$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{38}$$

Unlike the sigmoid or tanh function, the first derivative of the ReLU does not slowly converge to zero for large input values, i.e. the ReLU does not *saturate*. Saturation slows

[26] Note that we also considered the drop-out regularization method which randomly disables a certain fraction of neurons in the network for each epoch. This method has shown very good results, but it approximately doubles the training time (Duyck *et al.*, 2014), and that is obviously not practical for day traders

down computation time, because the weights and biases in ANN are adjusted according to rules (27) and (28). So, when the value of the first derivative of the activation function is small, $\mathbf{w}^{new}$ and $\mathbf{b}^{new}$ only change very little compared to $\mathbf{w}^{old}$ and $\mathbf{b}^{old}$, and therefore many iterations would be needed to achieve a significant reduction in $C(\mathbf{x}_1, ..., \mathbf{x}_n)$.

On the other hand, if the weighted input $z_j^l(\mathbf{x}_t)$ is negative, the derivative $f'(z_j^l(\mathbf{x}_t))$ is zero and therefore, neuron j in layer l will stop learning entirely, i.e. it will always be zero (Nielsen, 2015). This problem is referred to the *vanishing gradient* problem which shouldn't occur often, if η is set sufficiently low.

After the training phase, we will scale $a^L(\mathbf{x}_t^s)$ back to make the actual predictions for the Bitcoin prices in the validation and test set:

$$\hat{y}_{t+5} = a^L(\mathbf{x}_t^s) \times (\max(Y_{t+5}) - \min(Y_{t+5})) + \min(Y_{t+5})$$

4.4 Benchmark model

We will use a simple naïve forecast to judge whether the aforementioned highly flexible ML algorithms can actually help predicting Bitcoin prices. The forecast for $\hat{y}_{t+5}$ for all observations in the test sets is equal to y_t, i.e. $\hat{y}_{t+5} = y_t$.

4.5 Methodology overview

Each algorithm is trying to learn a different object and has its own advantages and disadvantages. RT are trying to learn splits, SVR is trying to learn a slope coefficient of the linear regression in the feature space and ANN are trying to learn weights and biases. RT have the advantage that they are easily interpretable (James *et al.*, 2013) and relatively fast to compute, but RT are not *forward-looking*, i.e. they produce splits which yield the largest reduction in RSS only considering the current split, but not all possible future splits (Mount, 2017). Compared to ANN, the major advantage of SVR is that due to the formulation of its optimization problem, SVR will find the global and unique optimum (Tay and Cao, 2001), and therefore does not have the risk of getting stuck in a local minimum or not converging to a solution. On the other hand, ANN are able to model any function up to a pre-specified level of error (Nielsen, 2015), but like SVR, ANN also suffer from long execution times.

5 Data / Feature Engineering

We will use transaction level data from the "bitstampUSD" exchange (bitcoincharts.com, 2018). We computed aggregated open, high, low and close (OHLC) and volume data for every minute and we will use $10,000$ observations to begin with. The first observation was on 24/07/2018 at 11:04 hours and the last observation was on 31/07/2018 12:05 hours. Below, we will show the formulas of each technical indicator in the predictor space and provide some intuition why each technical indicator is worth measuring.

5.1 Momentum (M)

$$M_t(h) = \begin{cases} NA, & \text{if } t < h \\ C_t - C_{t-h}, & \text{otherwise} \end{cases}, \tag{39}$$

where C_t is the closing price of the current minute t, C_{t-h} is the closing price h minutes ago and NA stands for "not available". Momentum measures the velocity of price changes (Murphy, 1999), so if momentum is increasing and is above (below) zero, prices are rising (falling) at an increasing (decreasing) rate. If momentum is decreasing and is above (below) zero, prices are rising (falling) at a decreasing (increasing) rate. For Momentum and all other indicators, we will set $h = 15$, unless otherwise specified.

5.2 Moving averages (MAs) cross-overs

In this section, we will introduce three variables which measure the difference between three types of MAs of different lengths, because according to Murphy (1999), MAs of different lengths crossing each other, i.e. when the difference of between two MAs is zero, are generating trend reversal signals. The first variable will measure the difference between two simple MAs (SMAs) of different length:

$$DSMA_t(h, j) = SMA_t(h) - SMA_t(j), \tag{40}$$

where

$$SMA_t(h) = \begin{cases} NA, & \text{if } t < h \\ \dfrac{\sum_{i=0}^{h-1} C_{t-i}}{h}, & \text{otherwise} \end{cases}$$

The second variable will measure the difference between two weighted MAs (WMAs):

$$DWMA_t(h, j) = WMA_t(h) - WMA_t(j), \tag{41}$$

where

$$WMA_t(h) = \begin{cases} NA, & \text{if } t < h \\ \dfrac{\sum_{i=0}^{h-1}(h-i)C_{t-i}}{h(h-1)/2}, & \text{otherwise} \end{cases}$$

The third variable will measure the difference between two exponentially smoothed MAs (EMAs):

$$DEMA_t(h, j) = EMA_t(h) - EMA_t(j), \tag{42}$$

where

$$EMA_t(h) = \begin{cases} NA, & \text{if } t < h \\ SMA_t(h), & \text{if } t = h \\ \dfrac{2}{h+1} C_t + (1 - \dfrac{2}{h+1}) EMA_{t-1}(h), & \text{if } t > h \end{cases},$$

where we will set $j = 30$ in all indicators, unless otherwise specified. Note that the EMA is calculated recursively.

WMAs and EMAs place more importance on recent values than SMAs, while the weight factor of the WMA decreases linearly and the weight factor of the EMA decreases exponentially. Unlike SMAs and WMAs, EMAs do not drop off any past values and therefore also account for any sharp price changes in the past. The convergence and divergence of MAs of different lengths may be an early trend reversal signal and when the shorter MA crosses above (below) the longer MA, an up-trend (down-trend) is assumed to be confirmed (Murphy, 1999). Since MAs are an average of many prices, i.e. since they *lag* price action[27], they might generate more reliable trend reversal signals, than e.g. Momentum.

5.3 Commodity Channel Index (CCI)

$$CCI_t(h) = \begin{cases} NA, & \text{if } t < h \\ \dfrac{TP_t - SMA_t(h)}{0.015 AD_t(h)}, & \text{otherwise} \end{cases}, \tag{43}$$

where

$$TP_t = \frac{H_t + L_t + C_t}{3},$$

$$SMA_t(h) = \begin{cases} NA, & \text{if } t < h \\ \dfrac{\sum_{i=0}^{h-1} (TP_{t-i})}{h}, & \text{otherwise} \end{cases},$$

$$AD_t(h) = \begin{cases} NA, & \text{if } t < h \\ \dfrac{\sum_{i=0}^{h-1} |\, TP_{t-i} - SMA_t(h)|}{h}, & \text{otherwise} \end{cases}$$

H_t and L_t represent the high and low price of every minute respectively, so TP_t represents a "typical" price in period t (Murphy, 1999). In this case $SMA_t(h)$ is an h-period MA of TP_t and $AD_t(h)$ measures the average distance of TP_t from $SMA_t(h)$. By including the constant 0.015, "most" $CCI_t(h)$ values will fall in the range $[-100, 100]$ (Murphy, 1999), so any values approaching or exceeding this range indicate that a trend reversal could happen soon. $CCI_t(h)$ may help spotting new trends in their early stages, since it can help to identify whether some TP_t is just within its usually occurring fluctuations, or whether TP_t is significantly different from its past h values.

5.4 Relative Strength Index (RSI)

$$RSI_t(h) = \begin{cases} NA, & \text{if } t < h \\ 100 - \dfrac{100}{1 + RS_t(h)}, & \text{otherwise} \end{cases}, \tag{44}$$

[27] Since MAs are an average of many past prices, prices change much faster than MAs, hence MAs "lag" price action

where $RS_t(h)$ represents the ratio of two MAs, i.e.:

$$RS_t(h) = \begin{cases} NA, & \text{if } t < h \\ \dfrac{(1/h)\sum_{i=0}^{h-1} UP_{t-i}}{-(1/h)\sum_{i=0}^{h-1} DO_{t-i}}, & \text{otherwise} \end{cases},$$

where

$$UP_t = \begin{cases} NA, & \text{if } t < h \\ C_t - C_{t-1}, & \text{if } t \geq h,\ C_t - C_{t-1} \geq 0 \\ 0, & \text{if } t \geq h,\ C_t - C_{t-1} < 0 \end{cases},$$

$$DO_t = \begin{cases} NA, & \text{if } t < h \\ C_t - C_{t-1}, & \text{if } t \geq h,\ C_t - C_{t-1} < 0 \\ 0, & \text{if } t \geq h,\ C_t - C_{t-1} \geq 0 \end{cases}$$

The $RSI_t(h)$ is bound between $[0, 100]$ and $RSI_t(h)$ takes on higher values in up-trends and lower values in down-trends (Murphy, 1999). Usually, when the $RSI_t(h)$ is above 70 (below 30), the market is considered to be overbought (oversold) and a down-trend (up-trend) might be near. If $C_t - C_{t-1} > 0$ for h periods, $RS_t(h)$ is not defined, in which case, we will set $RSI_t(h) = 100$.

5.5 %K and %D cross-over

In this section we will introduce a variable measuring the difference of the %K and %D lines:

$$DKD_t(h) = \%K_t(h) - \%D_t, \tag{45}$$

where

$$\%K_t(h) = \begin{cases} NA, & \text{if } t < h \\ 100 \times \dfrac{C_t - LL_{t-(h-1)}}{HH_{t-(h-1)} - LL_{t-(h-1)}}, & \text{otherwise} \end{cases},$$

and

$$\%D_t = \begin{cases} NA, & \text{if } h + 3 < t \\ \dfrac{\sum_{i=0}^{3-1} \%K(h)_{t-i}}{3}, & \text{otherwise} \end{cases}$$

$LL_{t-(h-1)}$ represents the lowest low and $HH_{t-(h-1)}$ represents the highest high of the past h trading periods. $\%K_t(h)$ is based on the observation that as prices increase, closing prices tend to be closer to the upper boundary of the h-period price range, and closer to the lower boundary, if prices decrease (Murphy, 1999). The major trend reversal signal to notice is when %K crosses its own 3-period MA, which is called $\%D_t$. The interpretation is the same as the interpretation of MA cross-over signals.

5.6 Larry Williams %R

$$\%R_t(h) = \begin{cases} NA, & \text{if } t < h \\ 100 \times \dfrac{HH_{t-(h-1)} - C_t}{HH_{t-(h-1)} - LL_{t-(h-1)}}, & \text{otherwise} \end{cases} \tag{46}$$

$\%R$ is based on the same observation which inspired the creation of $\%K$, with the only difference that $\%R$ shows the relationship between C_t and $HH_{t-(h-1)}$ in relation to the maximum price range of the last h periods (Murphy, 1999).

5.7 Force Index (FI)

$$FI_t(h) = \begin{cases} NA, & \text{if } t = 1 \\ (C_t - C_{t-1})V_t, & \text{if } t = 2 \\ \dfrac{2}{h+1}(C_t - C_{t-1})V_t + (1 - \dfrac{2}{h+1})(C_{t-1} - C_{t-2})V_{t-1}, & \text{if } t > 2 \end{cases} \tag{47}$$

According to its inventor Alexander Edler, the FI measures the extent of the price change by $C_t - C_{t-1}$ and the commitment of the buyers or sellers by V_t which represents the trading volume at time t (Ładyżyński *et al.*, 2013). So, positive (negative) price changes combined with heavy volume may indicate that there are many committed buyers (sellers) in the market.

5.8 Vortex Indicator (VI)

The VI, invented by Botes and Siepman (2010), consists of an upper and a lower boundary which generate trend reversal signals, if they cross-over, i.e. if the difference of the two is zero. Hence, we will introduce the following variable:

$$DVI_t(h) = {VI_t}^+(h) - {VI_t}^-(h), \tag{48}$$

where

$${VI_t}^+(h) = \begin{cases} NA, & \text{if } t < h \\ \dfrac{\sum_{i=0}^{h-1} |H_{t-i} - L_{t-1-i}|}{\sum_{i=0}^{h-1} \max\{(H_{t-i} - L_{t-i}), |H_{t-i} - C_{t-1-i}|, |L_{t-i} - C_{t-1-i}|\}}, & \text{otherwise} \end{cases},$$

and

$${VI_t}^-(h) = \begin{cases} NA, & \text{if } t < h \\ \dfrac{\sum_{i=0}^{h-1} |L_{t-i} - H_{t-1-i}|}{\sum_{i=0}^{h-1} \max\{(H_{t-i} - L_{t-i}), |H_{t-i} - C_{t-1-i}|, |L_{t-i} - C_{t-1-i}|\}}, & \text{otherwise} \end{cases}$$

Notice that the only thing which differentiates ${VI_t}^+(h)$ from ${VI_t}^-(h)$ is the switch in the numerator and that the denominator is an h-period sum of the true range (Żbikowski, 2015). When $DVI_t(h) > 0$, the market is trending up and when $DVI_t(h) < 0$, the market is trending down. When $DVI_t(h) = 0$, a trend reversal is in effect and when the absolute value, $|DVI_t(h)|$, increases (decreases), i.e. if both VIs diverge (converge), the current trend strengthens (weakens).

5.9 On Balance Volume (OBV)

$$OBV_t = \begin{cases} V_t, & \text{if } t = 1 \\ OBV_{t-1} + V_t, & \text{if } t > 1,\ C_t > C_{t-1} \\ OBV_{t-1} - V_t, & \text{if } t > 1,\ C_t < C_{t-1} \\ OBV_{t-1}, & \text{if } t > 1,\ C_t = C_{t-1} \end{cases} \tag{49}$$

OBV, invented by Granville (1964), is based on the theory that changes in volume precede changes in price. A rising (falling) OBV represents positive (negative) volume pressure which could eventually lead to higher (lower) prices.

5.10 Summary Statistics

Table 1 shows the summary statistics of each variable and figure 6 shows the Bitcoin closing prices during the sampling period.

Table 1: Summary Statistics

Statistic	N	Mean	St. Dev.	Min	Pctl(25)	Pctl(75)	Max
Close	10,000	8,164.3	**111.2**	7,798.5	8,130.0	8,220.0	8,475.0
Volume	10,000	5.9	14.8	0.001	0.3	5.3	347.8
M	10,000	0.1	24.6	−189.2	−11.4	11.6	229.9
DSMA	10,000	0.05	10.2	−66.9	−4.7	4.9	88.8
DWMA	10,000	0.03	7.1	−49.5	−3.2	3.3	60.1
DEMA	10,000	0.04	7.4	−40.0	−3.1	3.1	58.8
CCI	10,000	0.2	110.5	−410.5	−83.9	84.4	500.0
RSI	10,000	50.0	11.1	12.1	43.0	57.1	88.3
DKD	10,000	−0.000	0.1	−0.6	−0.1	0.1	0.6
R	10,000	0.5	0.3	0.0	0.2	0.8	1.0
FI	10,000	−1.4	303.5	−12,136.0	−3.3	4.2	6,274.8
DVI	10,000	0.01	0.4	−1.1	−0.2	0.3	2.1
OBV	10,000	5,032.7	562.8	3,044.9	4,697.7	5,473.4	5,816.7

Notes: All variables except for Close and Volume are part of the predictor space. Their names are the same as in sections 5.1 to 5.9, except for %R which has become "R". The first $h-1$, $j-1$, or $h+3$ values of some indicators are NA, so for the computation of the variables, we added 500 observations to the beginning of the data frame. After we calculated all technical indicators, we discarded the x most recent observations and the oldest $500-x$ observations (if the prediction horizon was x), so that we have exactly $10,000$ non-NA observations to begin with

As table 1 shows, Bitcoin's is "a highly speculative asset" (Yellen, 2017). This becomes evident when looking at Bitcoin's standard deviation. In the sampling period, Bitcoin's standard deviation was \$111.2 (1.36% of sample mean), while the standard deviation of

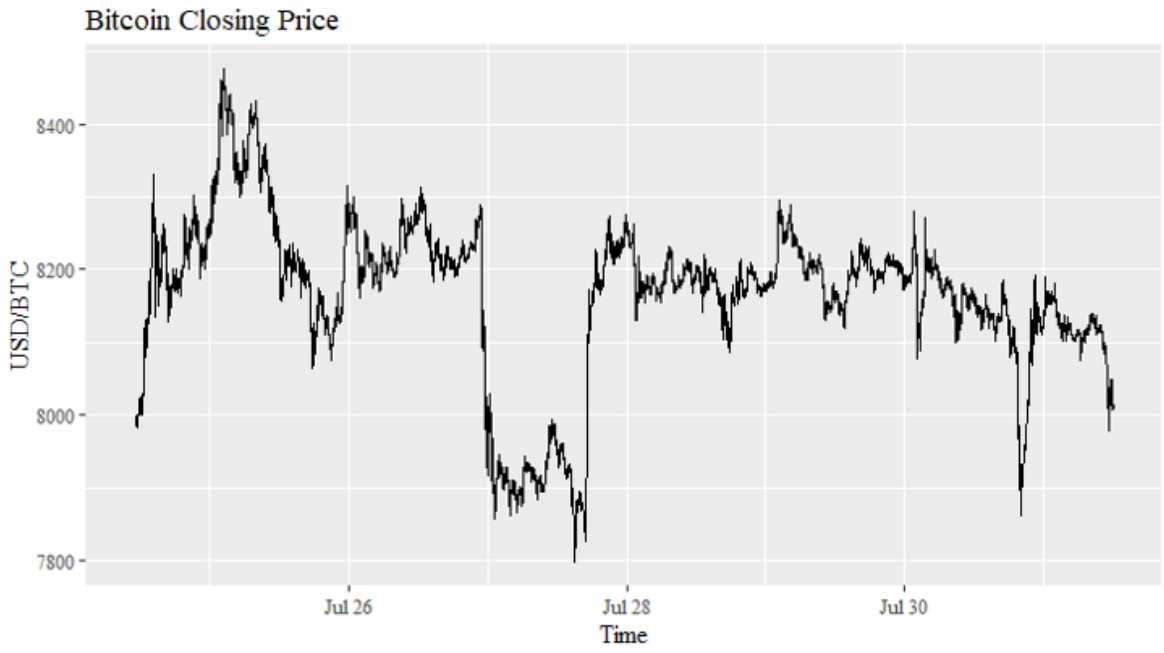

Figure 6: Bitcoin closing prices of every minute between 24/07/2018 11:04 hours and 31/07/2018 12:05 hours. Source: bitcoincharts.com (2018)

the S&P 500 during approximately the same period was 14.798 points (0.52% of sample mean)(S&P, 2018)[28].

6 Results

6.1 Experimental Settings

For RT, we set the sequence of the CP to $2^0, 2^{-1}, ..., 2^{-30}$ and to set the sequence for *minbucket*, we first constructed the sequence $1.2^1, 1.2^2, ...1.2^{35}$, then we rounded these values to integers and then, we eliminated the duplicates. This left us with 31 values of both CP and *minbucket*, and hence $31^2 = 961$ different combinations to test on each training set size. We tested five different training set sizes including $200, 400, ..., 1,000$ observations, so we had to test $961 \times 5 = 4,805$ different hyper-parameter combinations in each rolling window.

For SVM, we set the sequence of ϵ to $2^0, 2^{-1}, ..., 2^{-20}$, we set the sequence of γ and the sequence of the cost parameter C to $2^{10}, 2^9, ..., 2^{-10}$. This left us with 21 values of ϵ, γ and C and therefore $21^3 \times 5 = 46,305$ different combinations to test on each rolling window.

For ANN, we chose 5 hidden layers with 10 neurons each. Unfortunately, it was not computationally feasible to test multiple different values of the hyper-parameter values of ANN[29], so we set the number of epochs to 1000, the learning rate (LR) to 0.0001 and the

[28]For the S&P 500, we could only find daily data for the sampling period

[29]Because e.g. R just closed itself overnight, because an error related to memory space occurred, or because one of the central processing units (CPUs) died while parallel computing on all but one CPU

L1 regularization term to 0.001.

6.2 Presentation of Results

Tables 2,3,4 display the results for each individual model. The "Window" column represents the window index measured from the end of the dataset[30], the "MSE" column contains the model's test MSE, the following two or three columns contain the optimal hyper-parameter values, the "MSE naive" column contains the test MSE of the naïve benchmark forecast and the "MSE ratio" column contains the ratios of the model's MSE to the benchmark's MSE. Therefore, the MSE ratio shows the degree by which the model performed better or worse than the benchmark. Ratios above (below) 1 indicate that the model did worse (better) than the benchmark. Table 5 shows the computation time and average test MSE of each model as well as the average test MSE of the naïve benchmark.

Table 2: Results of Regression Trees (dependent variable: $Close_{t+5}$)

Window	MSE	Training	CP	Minbucket	MSE Naive	MSE ratio
1	4,905.259	200	9.313e-10	2	186.422	26.313
2	59.585	200	9.313e-10	5	87.994	0.677
3	1.626	800	5.000e-01	1	94.757	0.017
4	66.470	400	9.313e-10	8	62.039	1.071
5	314.567	200	7.812e-03	1	12.480	25.206
6	29.937	400	9.313e-10	79	9.627	3.110
7	160.035	800	9.313e-10	164	182.905	0.875
8	72.788	200	9.313e-10	7	394.200	0.185
9	150.945	1,000	9.313e-10	284	156.547	0.964
10	8.770	600	9.313e-10	6	141.187	0.062

Notes: MSE refers to the model's test MSE and Training refers to the optimal training set size. CP is the cost complexity parameter equivalent to to alpha in equation (2). Any split that does not decrease the overall lack of fit by a factor of CP is not attempted. Minbucket is the minimum number of observations in any terminal node. The MSE ratio is the model's MSE divided by the benchmark's MSE

Tables 2, 3 and 4 show that RT, SVR and ANN beat the benchmark in 6/10, 6/10 and 0/10[31] windows respectively, and that both RT and SVR have by far the most severe underperformance in windows 1 and 5. Table 5 shows that the models perform a lot worse on average and that RT is by far the most accurate and fastest algorithm.

[30]So, the 1st window contains the most recent observations and the last window contains the oldest observations

[31]$x/10$ refers to "x out of ten"

Table 3: Results of Support Vector Regression (dependent variable: $Close_{t+5}$)

Window	MSE	Training	Cost	Epsilon	Gamma	MSE Naive	MSE ratio
1	8,634.612	400	2.560e+02	1.221e-04	3.125e-02	186.422	46.318
2	1.741	600	2.500e-01	5.000e-01	1.024e+03	87.994	0.020
3	338.798	200	1.024e+03	3.052e-05	3.125e-02	94.757	3.575
4	38.455	400	5.120e+02	2.500e-01	7.812e-03	62.039	0.620
5	806.301	600	4.000e+00	1.562e-02	3.125e-02	12.480	64.609
6	37.943	200	6.400e+01	3.125e-02	1.953e-03	9.627	3.941
7	111.152	600	6.250e-02	1.250e-01	5.000e-01	182.905	0.608
8	19.643	800	1.250e-01	5.000e-01	2.500e-01	394.200	0.050
9	78.953	600	1.000e+00	3.906e-03	1.600e+01	156.547	0.504
10	138.942	1,000	5.000e-01	5.000e-01	4.000e+00	141.187	0.984

Notes: MSE refers to the model's test MSE and Training refers to the optimal training set size. Cost controls the trade-off between the flatness of $f(\mathbf{x}_t)$ and the maximum number of observations which deviate by more than Epsilon (see equation 7). Epsilon controls the size of the ϵ-insensitive tube (see equation 6), and Gamma is the width parameter of the RBF Kernel (see equation 16). The MSE ratio is the model's MSE divided by the benchmark's MSE

Table 4: Results of Artificial Neural Networks (dependent variable: $Close_{t+5}$)

Window	MSE	Training	Epochs	LR	Cost	MSE Naive	MSE ratio
1	7,807.791	200	1,000	1e-03	1e-03	186.422	41.882
2	7,798.745	600	1,000	1e-03	1e-03	87.994	88.628
3	7,800.053	1,000	1,000	1e-03	1e-03	94.757	82.316
4	7,798.845	1,000	1,000	1e-03	1e-03	62.039	125.708
5	7,798.688	200	1,000	1e-03	1e-03	12.480	624.910
6	7,799.743	1,000	1,000	1e-03	1e-03	9.627	810.205
7	7,799.253	400	1,000	1e-03	1e-03	182.905	42.641
8	7,798.709	400	1,000	1e-03	1e-03	394.200	19.784
9	7,801.514	400	1,000	1e-03	1e-03	156.547	49.835
10	7,815.329	200	1,000	1e-03	1e-03	141.187	55.354

Notes: MSE refers to the model's test MSE and Training refers to the optimal training set size. Epochs is the number of gradient descent steps taken towards the minimum of the cost function. LR is the learning rate (see equation 27 and 28), and Cost is the L1 regularization parameter (see equation 31). The results of each run of ANN may differ, because the weight initialization is random. The MSE ratio is the model's MSE divided by the benchmark's MSE

Table 5: Average Results (dependent variable: $Close_{t+5}$)

Model	Comp. Time Model	Avg. MSE Model	Avg. MSE Naive	Avg. MSE ratio
RT	5.7 Minutes	576.998	132.816	4.344
SVR	6.98 Hours	1,020.654	132.816	7.685
ANN	7.38 Minutes	7,801.867	132.816	58.742

Notes: Results of RT, SVR and ANN averaged over all rolling windows. "Comp". refers to "computation" and "Avg." refers to "average". The MSE ratio is the model's average MSE divided by the benchmark's average MSE

6.3 Discussion of Results

From a technical point of view, the most probable reason for under-performance of RT, SVR and ANN is that we did not pre-specify the model's hyper-parameters correctly. Especially, in the case of ANN, finding the optimal combination of hyper-parameters is very difficult, since ANN's hyper-parameter space is extremely vast. Assuming that the number of hidden layers is fixed to h, there are already $h + 3$ different hyper-parameters to tune[32], and that quickly becomes computationally infeasible. In particular, if the number of epochs and the learning rate are both too low, the ANN algorithm will never converge to the minimum of the cost function, and if the number of epochs and learning rate are both too high, the algorithm will constantly overshoot the minimum. Additionally, it could be the case that the ANN algorithm did not converge to the global minimum of the cost function, but merely to some local minimum.

We also suspected that the under-performance was due to a price jump or fall which the model might not have been able to capture. From an economic point of view, potential jumps in prices could have been caused by news shocks related to Bitcoin, or possibly by market manipulation.

During the sampling periods[33], a couple of news shocks actually occurred. E.g., the US Securities and Exchange Commission (SEC) dampened hopes that Bicoin related Exchange Traded Funds (ETFs) might be traded soon which would have led to more transparency and better protection against fraud (Rooney, 2018*b*). At the same time, there were news about a hacker stealing $2 million worth of Bitcoins (Russom and Flanigan, 2018) and the bank UBS saying that Bitcoin is still too "unstable to become mainstream money" (Rooney, 2018*a*).

In terms of market manipulation, it could have been the case that a group of investors illegally arranged to buy or sell huge amounts of Bitcoin at the same time and thereby affected Bitcoin's price. Since Bitcoin transactions are being made anonymously, it is not possible to verify this theory directly, but there is some evidence that there has indeed been some suspicious market activity in the past. Griffin and Shams (2018) found that at least half of the price hike in 2017 could have been due to price manipulation using Tether[34]. It

[32]I.e. the number of neurons in each hidden layer, the number of epochs, the learning rate, and the regularization parameter

[33]I.e. approximately between 27.07.2018 and 02.08.2018, the actually used time frame for evaluating the models

[34]Tether is another crypto-currency pegged to the US Dollar

actually turned out that Tether purchases were timed following market downturns which resulted in sizeable increases in the Bitcoin price.

Besides that, since RT, SVR and ANN are highly flexible ML algorithms, we suspected that they might only be able to capture non-linear patterns in the data. In that case, they would be performing very badly while the market is trading sideways, i.e. during periods when the Bitcoin time series is rather flat.

To verify the above hypotheses, we took a closer look at the sampling periods of RT and SVR[35]. As figure 7 shows, it is not possible to verify the hypotheses about price jumps being caused by news shocks or market manipulation, because even if there is a dramatic price change, as indicated by the solid red arrows, the ML algorithms are sometimes performing better than the naïve benchmark. We also can't confirm the hypothesis that the ML algorithms are performing worse while the market is trading sideways, because during the sampling period, the market was never really trading sideways, and during some periods with relatively low fluctuations, the ML algorithms were sometimes even performing better, as indicated by the dashed red arrows.

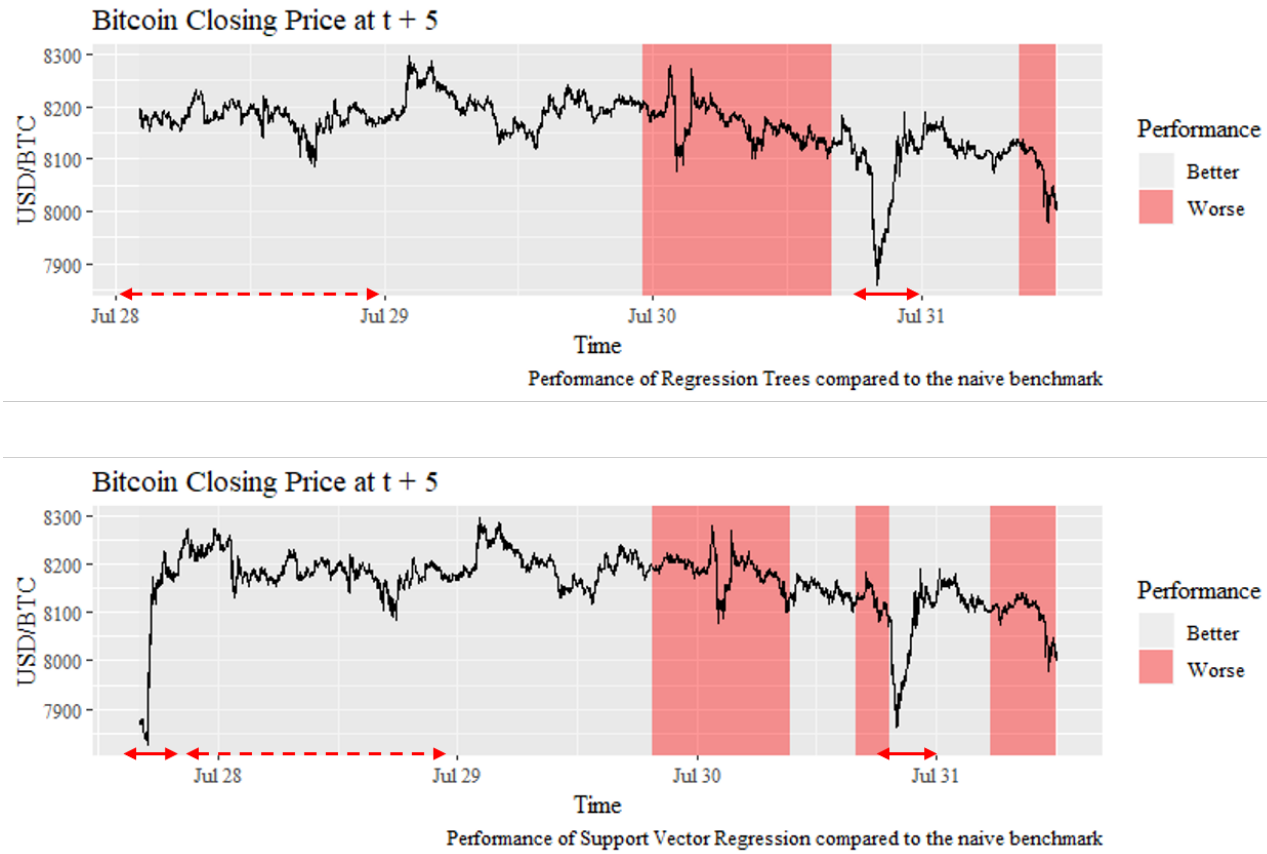

Figure 7: Performance of RT (top chart) and SVR (bottom chart) relative to the naïve benchmark

[35]Note that due to the consistent under-performance of ANN, we only further investigated the under-performance of RT and SVR

6.4 Robustness Checks

We varied the prediction horizon to 3 minutes ahead and 10 minutes ahead. When predicting y_{t+3}, SVR performed best and when predicting y_{t+10}, RT performed best again, while the performance of ANN did not change by a considerable amount. While the average MSE of each model increased as the prediction horizon increased, the average MSE ratio decreased, but in no scenario, any model was able to outperform the benchmark. The detailed results can be found in the appendix.

7 Conclusions & Future Research

This paper has investigated whether three flexible ML algorithms (RT, SVR and ANN) could help predict the Bitcoin price five minutes ahead using eleven technical indicators. The results were that, on average, all ML algorithms performed worse than the naïve benchmark, while RT was the fastest and most accurate ML algorithm, followed by SVR and ANN (in terms of prediction performance). We suspected that the overall under-performance is either due to the incorrect setting of hyper-parameters, due to news shocks, market manipulation or periods in which the market is trading sideways. None of these hypotheses could be completely confirmed, but we believe it is most likely that we did not find the optimal hyper-parameter combinations.

Assuming that, as time passes, computers will become more and more powerful, we would recommend future researchers of this topic to explore the space of hyper-parameters more thoroughly[36], to treat every arbitrarily set parameter as an additional hyper-parameter and to investigate whether block-chain variables are useful predictors.

Moreover, simply knowing whether ML algorithms can beat a naïve benchmark might still not be useful enough for professional investors, because beating a naïve benchmark does not guarantee a profitable investment strategy. Therefore, we also suggest to develop and implement a trading strategy based on the predictions generated by the ML algorithms and compare its performance with e.g. a simple buy-and-hold strategy.

If predicting Bitcoin prices does not work, future researchers could investigate whether it is possible to predict the volatility of Bitcoin, because predicting volatility of financial assets has been successful to some extend in the past (Andersen and Bollerslev, 1998; Harvey and Whaley, 1992). If predicting volatility of Bitcoin is indeed possible, one could pursue e.g. a *straddle* or *strangle* strategy[37] to make a profit.

If none of the above mentioned recommendations for future research work, we would advice professional traders not to invest into Bitcoin at all and to find assets for which accurate predictions can be made. If that does not work either, one could pursue a more passive strategy and invest in e.g. simple, non-leveraged ETFs tracking a stable index.

Finally, this study concludes that prediction remains a difficult task due to sizeable uncertainties involved in financial asset prices (Elliott and Timmermann, 2013) and the ongoing

[36] I.e. decrease the intervals between two subsequent hyper-parameter values and increase the range of each hyper-parameter at the same time

[37] A straddle (strangle) strategy involves buying both a call and a put option at (out of) the money which becomes profitable if the volatility of the underlying financial asset becomes high enough

competition in the market. Assuming that we have set all hyper-parameters correctly, we would conclude that the Bitcoin market is efficient, because according to the EMH, it is "impossible to make economic profits based on signals produced from a forecasting model"[38] (Timmermann and Granger, 2004).

[38]That is assuming that an implemented trading strategy based on our configurations of RT, SVR and ANN will not make economic profits

8 Appendix

Total word count: 7,910 excluding footnotes, tables, figures, captions and appendix (app.TexCount.php)

8.1 Robustness Checks

Table 6: Results of Regression Trees (dependent variable: $Close_{t+3}$)

Window	MSE	Training	CP	Minbucket	MSE Naive	MSE ratio
1	7,125.770	200	9.313e-10	2	71.197	100.086
2	12.859	200	9.313e-10	7	64.559	0.199
3	1.626	800	5.000e-01	1	107.988	0.015
4	54.484	600	1.953e-03	1	30.260	1.801
5	11.951	600	1.562e-02	79	6.609	1.808
6	72.001	200	9.313e-10	66	7.900	9.114
7	41.871	200	9.313e-10	95	119.205	0.351
8	27.893	200	1.000e+00	1	213.578	0.131
9	74.063	400	3.052e-05	1	68.397	1.083
10	134.918	200	9.766e-04	1	85.090	1.586

Notes: MSE refers to the model's test MSE and Training refers to the optimal training set size. CP is the cost complexity parameter equivalent to to alpha in equation (2). Any split that does not decrease the overall lack of fit by a factor of CP is not attempted. Minbucket is the minimum number of observations in any terminal node. The MSE ratio is the model's MSE divided by the benchmark's MSE

Table 7: Results of Support Vector Regression (dependent variable: $Close_{t+3}$)

Window	MSE	Training	Cost	Epsilon	Gamma	MSE Naive	MSE ratio
1	2,891.181	1,000	4.000e+00	1.000e+00	8.000e+00	71.197	40.608
2	121.856	200	1.024e+03	3.125e-02	3.125e-02	64.559	1.888
3	30.517	400	1.600e+01	2.500e-01	6.250e-02	107.988	0.283
4	99.440	200	8.000e+00	2.500e-01	9.766e-04	30.260	3.286
5	9.854	800	2.560e+02	5.000e-01	6.250e-02	6.609	1.491
6	26.948	1,000	6.250e-02	1.000e+00	1.250e-01	7.900	3.411
7	20.930	200	1.600e+01	1.000e+00	3.906e-03	119.205	0.176
8	79.088	800	1.250e-01	1.000e+00	1.600e+01	213.578	0.370
9	5.356	400	1.024e+03	2.500e-01	1.953e-03	68.397	0.078
10	49.384	1,000	1.600e+01	7.629e-06	9.766e-04	85.090	0.580

Notes: MSE refers to the model's test MSE and Training refers to the optimal training set size. Cost controls the trade-off between the flatness of $f(\mathbf{x}_t)$ and the maximum number of observations which deviate by more than Epsilon (see equation 7). Epsilon controls the size of the ϵ-insensitive tube (see equation 6), and Gamma is the width parameter of the RBF Kernel (see equation 16). The MSE ratio is the model's MSE divided by the benchmark's MSE

Table 8: Results of Artificial Neural Networks (dependent variable: $Close_{t+3}$)

Window	MSE	Training	Epochs	LR	Cost	MSE Naive	MSE ratio
1	7,807.180	200	1,000	1e-03	1e-03	71.197	109.657
2	7,798.751	600	1,000	1e-03	1e-03	64.559	120.801
3	7,798.771	1,000	1,000	1e-03	1e-03	107.988	72.219
4	7,799.739	200	1,000	1e-03	1e-03	30.260	257.754
5	7,806.074	1,000	1,000	1e-03	1e-03	6.609	1,181.142
6	7,798.817	200	1,000	1e-03	1e-03	7.900	987.201
7	7,800.599	600	1,000	1e-03	1e-03	119.205	65.439
8	7,798.764	600	1,000	1e-03	1e-03	213.578	36.515
9	7,798.737	400	1,000	1e-03	1e-03	68.397	114.022
10	7,799.291	400	1,000	1e-03	1e-03	85.090	91.659

Notes: MSE refers to the model's test MSE and Training refers to the optimal training set size. Epochs is the number of gradient descent steps taken towards the minimum of the cost function. LR is the learning rate (see equation 27 and 28), and Cost is the L1 regularization parameter (see equation 31). The results of each run of ANN may differ, because the weight initialization is random. The MSE ratio is the model's MSE divided by the benchmark's MSE

Table 9: Average Results (dependent variable: $Close_{t+3}$)

Model	Comp. Time Model	Avg. MSE Model	Avg. MSE Naive	Avg. MSE ratio
RT	5.53 Minutes	755.744	77.478	9.754
SVR	7.41 Hours	333.455	77.478	4.304
ANN	7.28 Minutes	7,800.672	77.478	100.682

Notes: Results of RT, SVR and ANN averaged over all rolling windows. "Comp". refers to "computation" and "Avg." refers to "average". The MSE ratio is the model's average MSE divided by the benchmark's average MSE

Table 10: Results of Regression Trees (dependent variable: $Close_{t+10}$)

Window	MSE	Training	CP	Minbucket	MSE Naive	MSE ratio
1	5,922.521	200	1.000e+00	1	452.584	13.086
2	7.833	200	9.313e-10	4	87.088	0.090
3	1.626	800	5.000e-01	1	93.255	0.017
4	37.730	600	9.313e-10	46	130.877	0.288
5	174.837	400	2.441e-04	5	88.725	1.971
6	80.173	800	9.313e-10	197	39.857	2.012
7	150.386	200	9.313e-10	55	216.027	0.696
8	133.361	1,000	9.313e-10	284	789.447	0.169
9	12.208	800	1.000e+00	1	255.326	0.048
10	282.949	200	4.883e-04	3	227.480	1.244

Notes: MSE refers to the model's test MSE and Training refers to the optimal training set size. CP is the cost complexity parameter equivalent to to alpha in equation (2). Any split that does not decrease the overall lack of fit by a factor of CP is not attempted. Minbucket is the minimum number of observations in any terminal node. The MSE ratio is the model's MSE divided by the benchmark's MSE

Table 11: Results of Support Vector Regression (dependent variable: $Close_{t+10}$)

Window	MSE	Training	Cost	Epsilon	Gamma	MSE Naive	MSE ratio
1	6,904.406	200	2.560e+02	9.537e-07	1.562e-02	452.584	15.256
2	38.112	800	5.000e-01	5.000e-01	1.953e-03	87.088	0.438
3	408.674	200	1.024e+03	6.250e-02	3.125e-02	93.255	4.382
4	9.955	200	2.560e+02	7.812e-03	7.812e-03	130.877	0.076
5	19.931	800	2.500e-01	5.000e-01	6.250e-02	88.725	0.225
6	37.044	1,000	3.125e-02	4.883e-04	1.000e+00	39.857	0.929
7	46.314	400	1.000e+00	1.000e+00	3.906e-03	216.027	0.214
8	114.868	400	4.000e+00	1.000e+00	1.562e-02	789.447	0.146
9	54.234	1,000	2.560e+02	6.250e-02	1.562e-02	255.326	0.212
10	58.700	800	6.400e+01	5.000e-01	7.812e-03	227.480	0.258

Notes: MSE refers to the model's test MSE and Training refers to the optimal training set size. Cost controls the trade-off between the flatness of $f(\mathbf{x}_t)$ and the maximum number of observations which deviate by more than Epsilon (see equation 7). Epsilon controls the size of the ϵ-insensitive tube (see equation 6), and Gamma is the width parameter of the RBF Kernel (see equation 16). The MSE ratio is the model's MSE divided by the benchmark's MSE

Table 12: Results of Artificial Neural Networks (dependent variable: $Close_{t+10}$)

Window	MSE	Training	Epochs	LR	Cost	MSE Naive	MSE ratio
1	7,807.158	200	1,000	1e-03	1e-03	452.584	17.250
2	7,798.744	600	1,000	1e-03	1e-03	87.088	89.551
3	7,804.214	800	1,000	1e-03	1e-03	93.255	83.687
4	7,800.830	200	1,000	1e-03	1e-03	130.877	59.604
5	7,798.754	200	1,000	1e-03	1e-03	88.725	87.898
6	7,806.959	1,000	1,000	1e-03	1e-03	39.857	195.875
7	7,798.769	800	1,000	1e-03	1e-03	216.027	36.101
8	7,799.070	800	1,000	1e-03	1e-03	789.447	9.879
9	7,798.869	200	1,000	1e-03	1e-03	255.326	30.545
10	7,799.639	400	1,000	1e-03	1e-03	227.480	34.287

Notes: MSE refers to the model's test MSE and Training refers to the optimal training set size. Epochs is the number of gradient descent steps taken towards the minimum of the cost function. LR is the learning rate (see equation 27 and 28), and Cost is the L1 regularization parameter (see equation 31). The results of each run of ANN may differ, because the weight initialization is random. The MSE ratio is the model's MSE divided by the benchmark's MSE

Table 13: Average Results (dependent variable: $Close_{t+10}$)

Model	Comp. Time Model	Avg. MSE Model	Avg. MSE Naive	Avg. MSE ratio
RT	5.7 Minutes	680.362	238.066	2.858
SVR	6.97 Hours	769.224	238.066	3.231
ANN	7.4 Minutes	7,801.301	238.066	32.769

Notes: Results of RT, SVR and ANN averaged over all rolling windows. "Comp". refers to "computation" and "Avg." refers to "average". The MSE ratio is the model's average MSE divided by the benchmark's average MSE

References

Andersen, T. G. and Bollerslev, T. (1998), 'Answering the skeptics: Yes, standard volatility models do provide accurate forecasts', *International economic review* pp. 885–905.

Ballings, M., Van den Poel, D., Hespeels, N. and Gryp, R. (2015), 'Evaluating multiple classifiers for stock price direction prediction', *Expert Systems with Applications* **42**(20), 7046–7056.

Bergstra, J. and Bengio, Y. (2012), 'Random search for hyper-parameter optimization', *Journal of Machine Learning Research* **13**(Feb), 281–305.

bitcoincharts.com (2018), ''.
URL: *http://api.bitcoincharts.com/v1/csv/*

Bossaerts, P. and Hillion, P. (1999), 'Implementing statistical criteria to select return forecasting models: what do we learn?', *The Review of Financial Studies* **12**(2), 405–428.

Botes, E. and Siepman, D. (2010), 'The vortex indicator', *Technical Analysis of Stocks & Commodities* **28**(1), 20–30.

Campbell, J. Y. (2000), 'Asset pricing at the millennium', *The Journal of Finance* **55**(4), 1515–1567.

Campbell, J. Y. (2008), 'Estimating the equity premium', *Canadian Journal of Economics/Revue canadienne d'économique* **41**(1), 1–21.

Cherkassky, V. and Ma, Y. (2004), 'Practical selection of svm parameters and noise estimation for svm regression', *Neural networks* **17**(1), 113–126.

Clarke, J., Jandik, T. and Mandelker, G. (2001), 'The efficient markets hypothesis', *Expert financial planning: Advice from industry leaders* pp. 126–141.

Duyck, J., Lee, M. H. and Lei, E. (2014), 'Modified dropout for training neural network', *School Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, USA, Advanced Introduction to Machine Learning Course, Tech. Rep* pp. 10–715.

Elliott, G. and Timmermann, A. (2013), *Handbook of economic forecasting*, Elsevier.

Glorot, X., Bordes, A. and Bengio, Y. (2011), Deep sparse rectifier neural networks, *in* 'Proceedings of the fourteenth international conference on artificial intelligence and statistics', pp. 315–323.

Goyal, A. and Welch, I. (2003), 'Predicting the equity premium with dividend ratios', *Management Science* **49**(5), 639–654.

Granville, J. E. (1964), *Granville's new key to stock market profits*, Prentice-Hall.

Greaves, A. and Au, B. (2015), 'Using the bitcoin transaction graph to predict the price of bitcoin', *No Data* .

Griffin, J. M. and Shams, A. (2018), 'Is bitcoin really un-tethered?'.

Harvey, C. R. and Whaley, R. E. (1992), 'Market volatility prediction and the efficiency of the s&p 100 index option market', *Journal of Financial Economics* **31**(1), 43–73.

Hsu, C.-W., Chang, C.-C., Lin, C.-J. *et al.* (2003), 'A practical guide to support vector classification'.

James, G., Witten, D., Hastie, T. and Tibshirani, R. (2013), *An introduction to statistical learning*, Vol. 112, Springer.

Jang, H. and Lee, J. (2018), 'An empirical study on modeling and prediction of bitcoin prices with bayesian neural networks based on blockchain information', *IEEE Access* **6**, 5427–5437.

Jarrett, K., Kavukcuoglu, K., LeCun, Y. *et al.* (2009), What is the best multi-stage architecture for object recognition?, *in* 'Computer Vision, 2009 IEEE 12th International Conference on', IEEE, pp. 2146–2153.

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012), Imagenet classification with deep convolutional neural networks, *in* 'Advances in neural information processing systems', pp. 1097–1105.

Ładyżyński, P., Żbikowski, K. and Grzegorzewski, P. (2013), Stock trading with random forests, trend detection tests and force index volume indicators, *in* 'International Conference on Artificial Intelligence and Soft Computing', Springer, pp. 441–452.

Lo, A. W. (2005), 'Reconciling efficient markets with behavioral finance: the adaptive markets hypothesis'.

Malkiel, B. G. and Fama, E. F. (1970), 'Efficient capital markets: A review of theory and empirical work', *The journal of Finance* **25**(2), 383–417.

Malkiel, B. G. and McCue, K. (1985), *A random walk down Wall Street*, Norton New York.

Mount, J. (2017), 'Why do decision trees work?'.
URL: *https://www.r-bloggers.com/why-do-decision-trees-work/*

Murphy, J. J. (1999), *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*, Penguin.

Nair, V. and Hinton, G. E. (2010), Rectified linear units improve restricted boltzmann machines, *in* 'Proceedings of the 27th international conference on machine learning (ICML-10)', pp. 807–814.

Nakamoto, S. (2008), 'Bitcoin: A peer-to-peer electronic cash system'.

Nielsen, M. A. (2015), *Neural networks and deep learning*, Determination Press.

Pan, X. and Srikumar, V. (2016), Expressiveness of rectifier networks, *in* 'International Conference on Machine Learning', pp. 2427–2435.

Patel, J., Shah, S., Thakkar, P. and Kotecha, K. (2015), 'Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques', *Expert Systems with Applications* **42**(1), 259–268.

quandl.com (2018), 'Bitcoin markets (bitstampusd)'.
URL: *https://www.quandl.com/data/BCHARTS/BITSTAMPUSD-Bitcoin-Markets-bitstampUSD*

Rooney, K. (2018*a*), 'Bitcoin is still too 'unstable' to become mainstream money, ubs says'.
URL: *https://www.cnbc.com/2018/08/02/bitcoin-still-too-unstable-to-be-mainstream-money-ubs-says.html*

Rooney, K. (2018*b*), 'Bitcoin rises, shaking off sec's denial of winklevoss etf'.
URL: *https://www.cnbc.com/2018/07/27/bitcoin-loses-steam-after-sec-denies-winklevoss-etf.html*

Russom, P. and Flanigan, K. (2018), 'Bitcoin is still too 'unstable' to become mainstream money, ubs says'.
URL: *https://www.cnbc.com/2018/08/02/bitcoin-still-too-unstable-to-be-mainstream-money-ubs-says.html*

Smola, A. J. and Schölkopf, B. (2004), 'A tutorial on support vector regression', *Statistics and computing* **14**(3), 199–222.

Snoek, J., Larochelle, H. and Adams, R. P. (2012), Practical bayesian optimization of machine learning algorithms, *in* 'Advances in neural information processing systems', pp. 2951–2959.

S&P (2018), 'S&p 500'.
URL: *https://eu.spindices.com/indices/equity/sp-500*

Tay, F. E. and Cao, L. (2001), 'Application of support vector machines in financial time series forecasting', *omega* **29**(4), 309–317.

Timmermann, A. and Granger, C. W. (2004), 'Efficient market hypothesis and forecasting', *International Journal of forecasting* **20**(1), 15–27.

Welch, I. and Goyal, A. (2007), 'A comprehensive look at the empirical performance of equity premium prediction', *The Review of Financial Studies* **21**(4), 1455–1508.

Wen, Q., Yang, Z., Song, Y. and Jia, P. (2010), 'Automatic stock decision support system based on box theory and svm algorithm', *Expert Systems with Applications* **37**(2), 1015–1022.

xpertup.com (2018), 'Loss functions and optimization algorithms'.
URL: *http://www.xpertup.com/2018/05/11/loss-functions-and-optimization-algorithms/*

Yeh, C.-Y., Huang, C.-W. and Lee, S.-J. (2011), 'A multiple-kernel support vector regression approach for stock market price forecasting', *Expert Systems with Applications* **38**(3), 2177–2186.

Yellen, J. (2017), 'Yellen says bitcoin is a highly speculative asset'.
URL: *https://www.bloomberg.com/news/articles/2017-12-13/yellen-says-cryptocurrency-bitcoin-is-highly-speculative-asset*

Żbikowski, K. (2015), 'Using volume weighted support vector machines with walk forward testing and feature selection for the purpose of creating stock trading strategy', *Expert Systems with Applications* **42**(4), 1797–1805.