ADVANCED PROJECT I

One Step Forecast of the Ether Criptocurrency

By Gari Ciodaro Guerra

Advisor: Prof. Dr. Herbert Jaeger

May 20, 2019

Abstract

In this report, we explore the possibility of predicting the next day average price of the cryptocurrency known as **Ether**. In the first part, we explore a pure time series analysis using **classical decomposition** in combination with machine learning techniques for one step forecast; specifically, we tested **Ridge regression** and **Multilayer Perceptron**. On the second part, we included social media sentiment extracted from **Twitter** and **Google trends** to try to capture human greed and collective fear on the final price. To increase the interpretability of the second part, we only used **Boosting with Regression Trees**.

This document is the final derivable of the course *Advanced Project 1* from *Jacobs University* master on *Data Engineering*. The purpose of the course as expressed in the program syllabus is *to provide the student with an in-depth understanding and command of one of the data analytics or data management techniques that are represented by the research groups of the faculty of Data Engineering.*

1 Introduction

1.1 Ethereum Blockchain technology

Blockchain technology was introduced to the general public in 2009 with the cryptocurrency Bitcoin [1] as a way of transfer/represent value in a decentralized auto-regulated network with the same name (Bitcoin network) as a consequence no single human nor government could control or regulate the exchange of digital assets. The Bitcoin network could be defined as a distributed database that contains encrypted information of the ownership of its unit value, namely the Bitcoin, a detailed explanation can be found in Satoshi Nakamoto's legendary white paper [2]. Let us now introduce a example of a operation on the Bitcoin network that will be generalized for Etherum network.

- Let there be two users of the Bitcoin network, this network is composed of nodes, each node has a single exact copy of the database.
- Both subjects have a private and a public cryptographic key, in the community these pair is referred to as a wallet. In reality, it can be viewed as a record on the public database stating that those private keys are associated with $x \in \mathbb{R}^{\geq 0}$ and $y \in \mathbb{R}^{\geq 0}$ Bitcoin units.
- If user A tries to reassign its x Bitcoin to user B, the Bitcoin network must reach a consensus. First, a validation of the claim *User A has x Bitcoins* must be verified by all the nodes in the network, this done just by checking in the local copy of the database if *User A has x Bitcoins* .
- After this validation, each node would compete to gain the right to *write* this new transaction to their local database and then broadcast to the other nodes. In the first implementations, this was done by iteratively passing a string representation of the operation to a hashing function until the outcome had the first N characters equal to 0. Nodes that actively seek to claim the right to update the database were commonly refers to as miners, and the reward for winning this competition would be the assignation of newly created Bitcoin to the miner wallet.
- The network has some extra coded features, such that the rate of updates of the database cannot exceed a threshold dictated by the number of actives nodes. By adjusting the degree of difficulty, in our example, the N characters beginning with zero, the supply of cryptocurrency gets regulated.

The previous example is just illustrative, there are many more complications regarding any Blockchain, however, we consider that it serves the purpose of introducing terminology in a straight forward manner to a nontechnical reader. For a quite pleasant technical overview of the subject, please refer to [3].

Vitalik Buterin introduced the Ethereum network in the white paper [4] to extend the Bitcoin network functionality by implementing a *Turing-complete programming language* as a consequence any node of the network is no longer limited to store a copy of the database but can also create its own *owernship rules, formats of transactions, and state transition functions* [3]. When a node defines its own rules, any other node can choose to enter the *contract* that would self execute according to the logical rules written in it. With this new artifact, users can create self-executing digital contracts whose unit values(the *Ether*) are publicly available for auditing while preserving the privacy of the users.

In addition to the intended use of the Ethereum Network from the prior discussion, there are platforms such as *Bitstamp* in Europe, *Coinbase* in the U.S.A, and *Bitso* in Mexico where users can speculate over the value of any cryptocurrency against any *fiat* such as the Euro, Dollar or Mexican Peso. It should be noticed that in these *exchanges* when a client buys a token using his or her fiat money, the blockchain in which that token represent value does not get updated as in our example. Instead, the ownership of the token gets updated inside the private records of the exchange giving

the illusion of instant transference times. Only when the user decides to take his or her cryptocurrencies out of the exchange into his own private wallet, for example, the involved Network gets updated.

There are two crucial considerations that ultimately motivate an attempt to predict the average price of the following day of the token Ether. First, we believe that *Smart contracts* applications depicts an excellent future for the long term value of the token, so investing in an intraday fashion is a subjectively risk reduced enterprise. Second, by perfecting a predicting technique, a risk reduced fiat market opens for the modeler, since a token can be freely moved between different exchanges that have separate fiat counterparts.

Name	Market Capitalization	Price	Circulating supply
Bitcoin	$139,157,553,209	$7,856.69	17,711,975 BTC
Ether	$26,234,724,752	$247.15	106,148,488 ETH
XRP	$16,517,864,972	$0.392038	42,133,310,721 XRP

Table 1: Principal economics indicator for Cryptocurrency against USD. Taken from [5]

1.2 Data Sources

Two sources of data were used throughout the present dissertation. Namely **Ether Scan** [6] specifically the price and daily transactions charts and the **Deepblue Bot** [7] for social media information regarding the Ethereum Network.

Figure 1: Ether scan data from 2016 to May 2019 [6]

The **Deepblue Bot** uses APIs request every 30 minutes to Google and Twitter servers. The Google server replies with the percentage ratio of the most popular search in respect to the words *Ether, Ethereum*, naturally 100 relative popularity units means that the word *Ether or Ethereum* were the most popular searches around the word during the last hour. In the case of Twitter, the server replies with the last 25 English twitters with the words *Ether, Ethereum* in them, and compare them with a dictionary [1] of words that imply a favorable and unfavorable sentiment. Every time a word of

[1]The list of words was extracted from [8]

favorable or not favorable sentiment is found in a Tweet the overall score of it is increased or decreased by 1, the results are averaged per request, and the register is inserted in MySQL table. To get a value estimate per day, the daily API request is averaged per day.

- **Favorable sentiment**: bull, good, great, increase, interesting, superior, rise gainer, whistleblower, speedy, dubious, scraps, acknowledge, delisted, downs boding, disappeared, botched, kongs, surely, resurgent, eos, hindered, leapt grapple, heated, forthcoming, standpoint, exacerbated, steer, toptier, braking jackets, featured, overcrowded, saddled, haul, beginning, future.
- **Not favorable sentiment**: bear, bad, horrible, decrease, uninteresting, inferior, dating, birthrate, reacting, lofty, accelerators, falsified, bust, averaging, pages championed, folded, trillions, santa, fourfold, wellknown, perfect, halfowned defaults, bottleneck, cloudy, strains, kicks, doubted, halving, retailings, abandon depressing, specifications, businessmen, diluting, fall, bubble, past, fear.

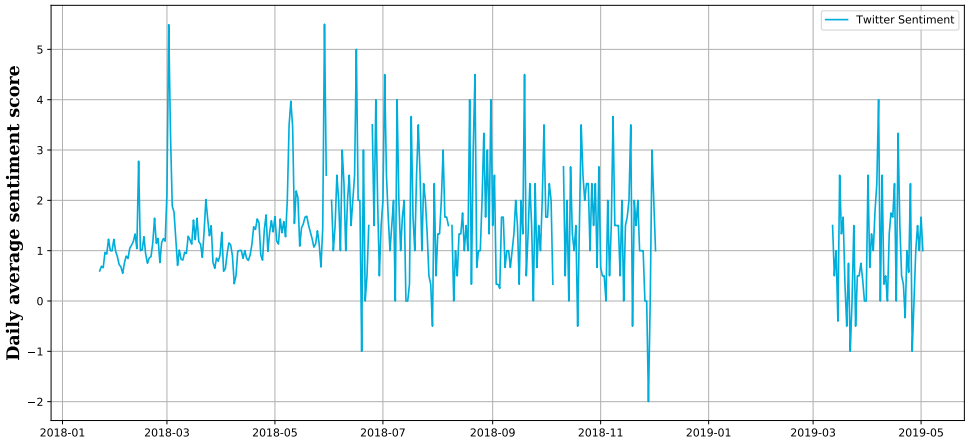

Figure 2: Average Twitter sentiment score regarding the words Ether and Ethereum per day, data source in [7]. Data for the beginning of 2019 not available.

Name	N	μ	σ	Description
Price(y_t)	1379	201.86	260.35	USD price. $y_t \in \mathbb{R}^{>0}$
Transaction(Tr)	1379	3.20x10^5	3.12x10^5	Number of transaction per day. $Tr \in \mathbb{N}^{>0}$
Interest(I)	360	43.17	5.64	Popularity on google search. $I \in [0, 100]$
Twitter(Tw)	360	1.40	1.03	Popularity Score on Twitter. $Tw \in \mathbb{R}$

Table 2: General data statistics.

1.3 Notation and terminology

We used the notation introduced in the lectures notes of machine learning course for fall 2019 [9].

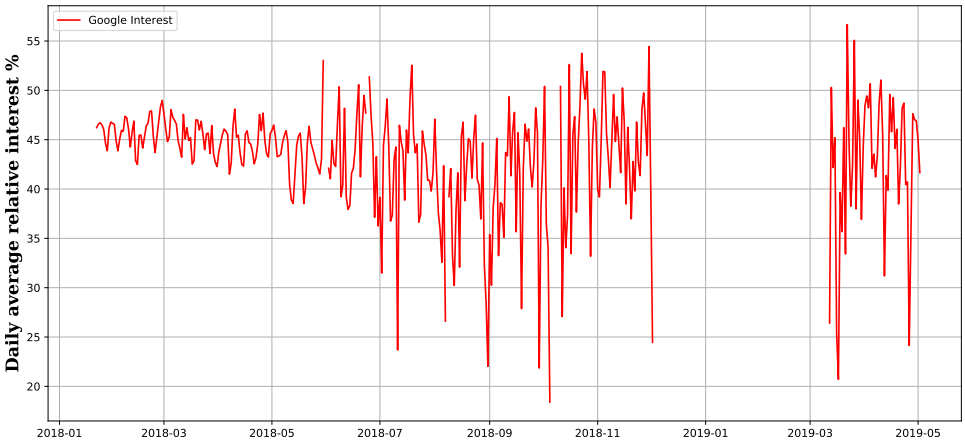

Figure 3: Average Google trend relative popularity score regarding the words Ether and Ethereum per day, data source in [7]. Data for the beginning of 2019 not available.

A pair-wise sequence of random variables X, Y realizations $(x_i, y_i)_{i=1,\dots,N}$ that come from the joint probability distribution $P_{X,Y}$ with data value spaces S_X, S_Y can be used in a supervised learning task to estimated a decision function $D : S_X \longrightarrow S_Y$ from a set of candidates $\mathcal{D}$ that has the lowest empirical R^{emp} commonly known as *training error*. The loss function $L : S_X \times S_Y \longrightarrow \mathbb{R}^{>=0}$ is a measure of the cost of miss predicting a realization of y_i with a decision function D. One last important concept is the notion of *Risk* or *test error*, which is defined as the expected loss over the true distribution $P_{X,Y}$.

$$R^{emp}(D) = \frac{1}{N}\sum_{i=1}^{N} L(D(x_i), y_i) \tag{1}$$

$$D_{opt} = \arg\min_{D\in\mathcal{D}} R^{emp}(D) \tag{2}$$

$$L(D(x), y) = ||D(x) - y||^2 \tag{3}$$

Unless stated otherwise, we used the *quadratic loss* defined in Equation 3 to calculate the R^{emp} to fit the set of parameters Θ of a giving *Statistical technique*, such as *Ridge regression* for example. However to compare between already fitted models we used **Cross-validation** to estimate the *test error*. Selecting the proper metric to make this comparison is of crucial importance, unfortunately and as stated in [10] there is not definitive theoretical background for metric selections, and the most commonly used such as mean absolute percentage error $MAPE$, mean square error MSE, mean absolute error MAE favors models that under forecast. We, therefore, decided to follow the standard metric used in the M3 competition [11], that is, the symmetric mean percentage error $sMAPE$. As shown in [8], the $sMAPE$ can favor under forecasting models in the case of large points error; nonetheless this was not a concern in the present work. The $sMAPE \in [-200, 200]$

$$sMAPE = \frac{100}{N}\sum_{i=1}^{N} \frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2} \tag{4}$$

2 Prior considerations

2.1 Data transformations

As can be observed in Table 2, the units of our data points are on different scales. As discussed in [12] it is not recommended for optimization procedures as machine learning estimations to work with such data directly, as a consequence all the work developed in this document uses Equation 5 to scale the data points.

$$sc(x_i) = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{5}$$

It is commented in [13] that abrupt changes in time series can increase the modeling difficulties due to properties of the respective derivates, as can be seen in Figure 1 y_t explodes near the end of 2017, we, therefore, implemented the natural logarithm transformation to smooth the rate of change of y_t.

$$f(y_t) = ln(y_t) \tag{6}$$

2.2 Data Interpolation

Using the **Deepblue bot** depends on the servers of Google and Twitter replying to the request; therefore, we do not have continuous data for some days. To overcome this difficulty, a third-degree polynomial was fitted between the missing days to fill up the gaps for the year 2018. The full mathematical description of cubic interpolation over one dimension can be found in [14], no further optimization or performance metric was calculated to asses the adequateness of the interpolation[2]. For the year 2019, the data does not have gaps from April to May; however, there is no data available from January to March.

2.3 The Naive Forecast

The naive forecast ($\hat{y}_{t+1} = y_t$) as explained in [13] is one of the simplest models and is typically used as the baseline for assessing the performance of more complicated models.

2.4 Cross-Validation for Time Series: Rolling Forecasting Origin

Using cross-validation on time series is different from other regression problems because there exists a temporal dependency between data points x_i. In [15] the concept of **rolling forecasting origin** is explained as follows:

1. *Select the observation at time $t + i$ where for the test set, and use the observations at times $1, 2, ..., t + i - 1$ to estimate the forecasting model. Compute the error on the forecast for time $t + 1$.*

2. *Repeat the above step for$1, 2, ..., N - t$ where N is the total number of observations.* From here on, the number of test value is defined as $N - t = CV_{Rol}$.

3. *Compute the forecast accuracy measures based on the errors obtained.*

[2] For the year 2018 we have 321 days for analysis, only 13 days needed interpolation

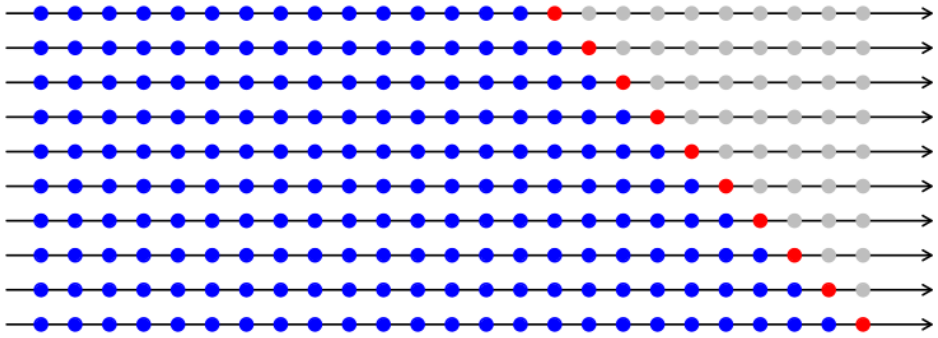

Figure 4: *Rolling forecasting origin scheme. One-step forecasts. The blue points are training sets,the red points are test sets and the grey points are ignored.* (taken from [15])

3 Time Series Analysis using Classical Decompositional model

Here we explore the possibility of predicting the next day average price by only using values of the past. Formally this is known as a one-time step forecast by autoregression models. As explained in [13] the classical Decompositional approach states that the Equation 7 holds.

$$y_t = S_t + T_t + E_t \tag{7}$$

Where S, T, E stand for *Seasonal*, *Trend* and *Error*. To estimate T we explored two methods, namely *odd moving averages* and *simple exponential smoothing*. *odd moving averages* is governed by Equation 8, where y_t is the price at time-step $t \in \mathbb{N}$, $odd \in \mathbb{N}$ the parameter of the steps averages and N the maximum index of the series which can be regarded as the present.

$$T_t^{mv}(odd) = \begin{cases} \sum_{i=(y_t - odd/2)}^{y_t + odd/2} \frac{y_i}{odd}, & \text{if } t < odd. \\ \sum_{i=(y_t - odd/2)}^{N} \frac{y_i}{odd}, & \text{otherwise.} \end{cases} \tag{8}$$

simple exponential smoothing is governed by in Equation 9 Where $\beta \in [0, 1]$ is the smoothing parameter.

$$T_t^{ses}(\beta) = \begin{cases} y_0, & \text{if } t = 0. \\ \beta y_t + (1-\beta) T_{t-1}^{ses}, & \text{otherwise.} \end{cases} \tag{9}$$

For the following section, we will refer by the trend calculated with moving averages and simple exponential smoothing only by T^{mv} and T^{ses} respectively.

3.1 Decomposition, Moving Averages based

In Figure 5 we can see moving averages $odd \in [87, 150, 297]$. Let us select $odd = 150$ moving average as T^{mv}.

By subtracting y and T^{mv} we obtain the combination $S+E$, we call this subtraction the *de-trended* series. We now assume that the *seasonal* component is repeated over the years per month. We also assumed that the error component has a normal distribution [3] with $\mu = 0$, as a result, by averaging the de-trended time series per month over the years we get an estimate of the *seasonal* average per month. In it is mentioned in [13] that this 12 element set is known as the *seasonal indexes*.

We decided to perform a 2-tail t-statistics test between $y = T^{mv} + S$ and $y' = T^{mv}$ to evaluate the null hypothesis H_0 that the difference between y and y' is produced by chance only, rendering S just like noise.

[3] by the central limit theorem

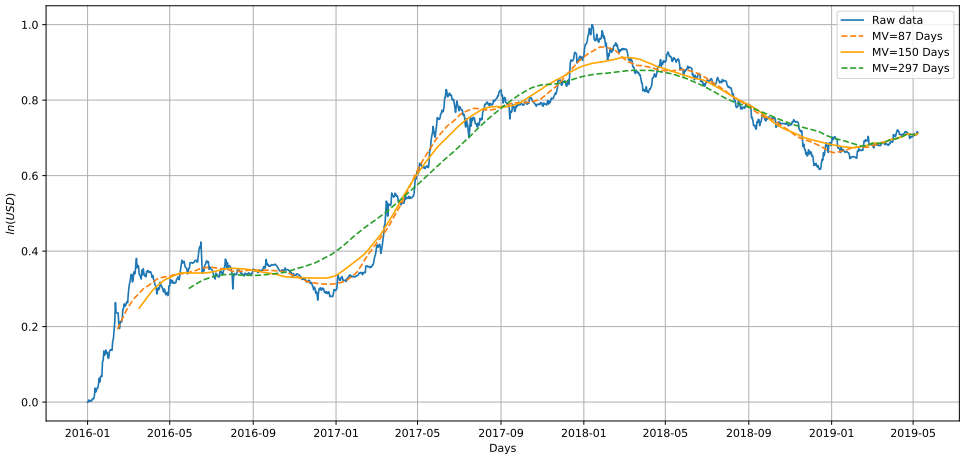

Figure 5: moving averages exploration

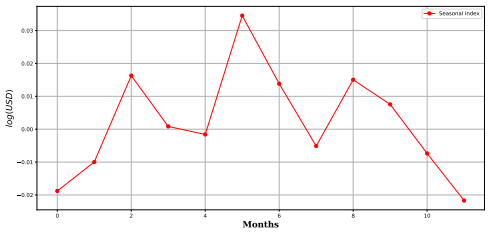

Figure 6: Seasonal indices.

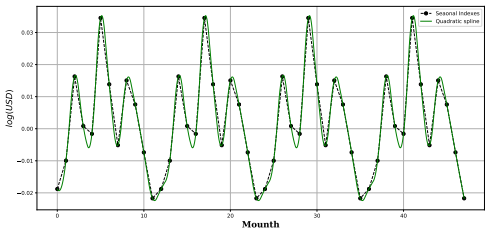

Figure 7: Seasonal interpolation.

RV	t-value	p-value
P_r,P'_r	0.27	0.78

Table 3: T-test for Seasonality.

As can be observed in table 3 the $p - value$ is near one, which tell us not to reject H_0. Following this conclusion we modified 7 to 10

$$y_t = T_t^{mv} + E_t \tag{10}$$

Finally, we obtain the E by calculating the relationship $E = y - T^{mv}$. Figure 8 shows E.
In figure 9 we can observe the full decomposition based on moving averages.

3.1.1 Ridge Regression for T^{mv} and E

In this section we try to model the component T^{mv} and E of Equation 10 using the decision functions $\mathcal{D}$ based on *Ridge regression* [4].

[4]Implemented using [16]

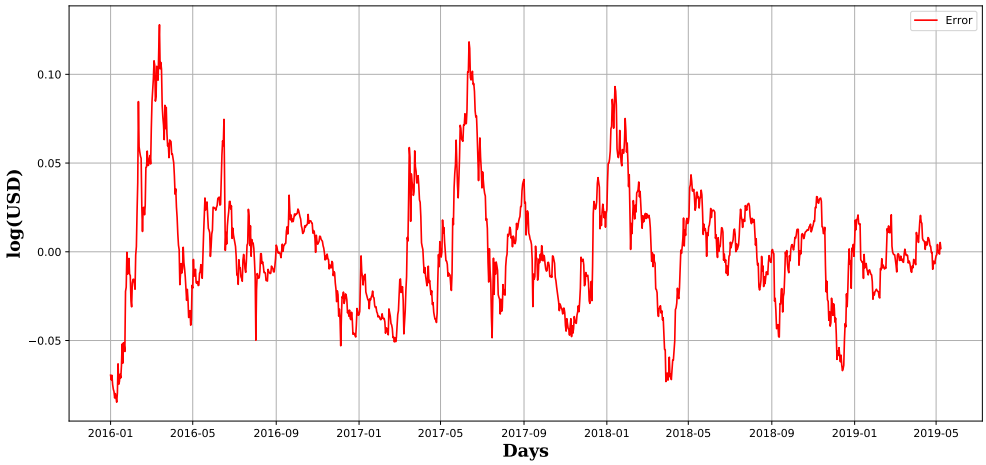

Figure 8: Error plot. E

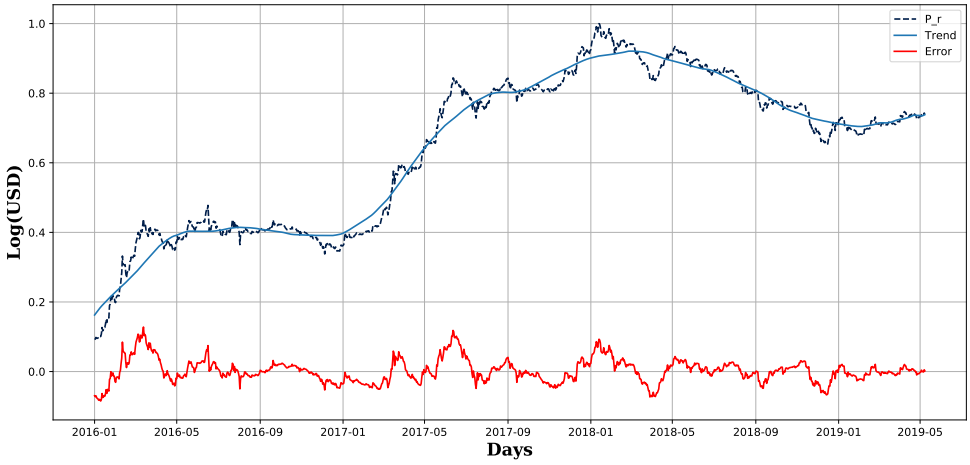

Figure 9: Single plot decomposition moving averages based.

Ridge regression is a supervised learning algorithm that minimized the quadric loss to fit a linear relationship between the features vector and the target variable. Using single value decomposition, the minimum quadric loss parameters can be encoded in a $W'_{opt} \in \mathbb{R}^{m \times m}$ matrix, where m is the number of features. In Equation 11 Φ is a matrix of $\mathbb{R}^{N \times m}$ where N is the number of data points. Z is a column vector $\mathbb{R}^N$ containing the target variable. $\alpha \in \mathbb{R}^{>0}$ is a penalization factor over the fitted parameters to control the variance of the model.

$$W'_{opt} = (\Phi'\Phi + \alpha^2 I)^{-1}\Phi' Z \tag{11}$$

Using this representation, the m dimension of our matrix Φ is our main free parameter. m can be regarded as the number of past time steps that we believe have a linear relationship with the $t+1$ step of our target variable. For the next section will refer to this parameter as lag_{max}. Note a matrix Φ with $\mathbb{R}^{N \times lag_{max}}$ will contain the sequence from t to $t - lag_{max}$ target variable observations.

Let us begin by trying to find the best lag_{max} parameter using Rolling forecasting origin. To do it, we set the number of test points $CV_{Rol} = 100$ (please see section 2.4 for more details) and proceed to evaluate different Ridge regression(with $\alpha = 0$), from less complex $lag_{max} = 1$ to a maximum of $lag_{max} = 19$, the average $\%sMAPE$ per lag_{max} is plotted to compare performance. The results can seem in in Figure 10 and 11 E and T^{mv} respectively.

In the case of T^{mv} is clear that any model with more than one lag outperforms the naive forecast (green line) however for E only the $lag_{max} = 3$ comes close to the naive forecast. In general terms the complexity of this model is giving by the parameter lag_{max} into consideration and as expected the training error is reduced while the complexity of the model increases, the test error, on the other hand, tends to increase. It can also be observed that the variance training $sMAPE$ is very small, 0.054% for E and 0.001% for T which plays in favor of the argument that this problem is very well protected against overfitting [5]. In the next optimization, we experiment with α to see if this is the case.

The minimum testing error is around 58% for E which is a sign that perhaps the real underlying probability distribution P for our problem is far from linear, it can also be argued that there is not enough information about the phenomena in our data set to estimate the distribution. We explore such questions in the following sections.

We now proceed to find the best α giving that$lag_{max} = 3$ is the optimal parameter for E. For the sake of simplicity, we also set $lag_{max} = 3$ to T^{mv}. The results can be appreciated in Figure 12 and 13.

The best models for both random variables are the ones with no regularization, as commented in the previous sections α is used to decrease the variance on the training set while theoretically increasing the generalization performance, nonetheless in our particular task the model has already low variance before using α, so its implementation is actually detrimental for performance. Another reason for this behavior can be attributed, in the case of E_t, to a very none linear problem, so that linear regression is not able to properly learn the patterns in the training data.

3.1.2 Single Layer Perceptron for E

A linear decision function was not appropriated for the variable E, so a non linear method, namely *Single Layer Perceptron* $\mathcal{N} : \mathbb{R}^{lag_{max}} \to \mathbb{R}$ in tested in this section. This machine learning algorithm is characterized by two weighting matrices $W_0 \in \mathbb{R}^{H_{units} \times lag_{max}}$ and $W_1 \in \mathbb{R}^{1 \times H_{units}}$ that connect the input feature vector $\mathbb{R}^{lag_{max}}$ to a hidden layer that contains $H_{units} \in N$ hidden units (a hidden layer

[5] At least the way it was implemented here.

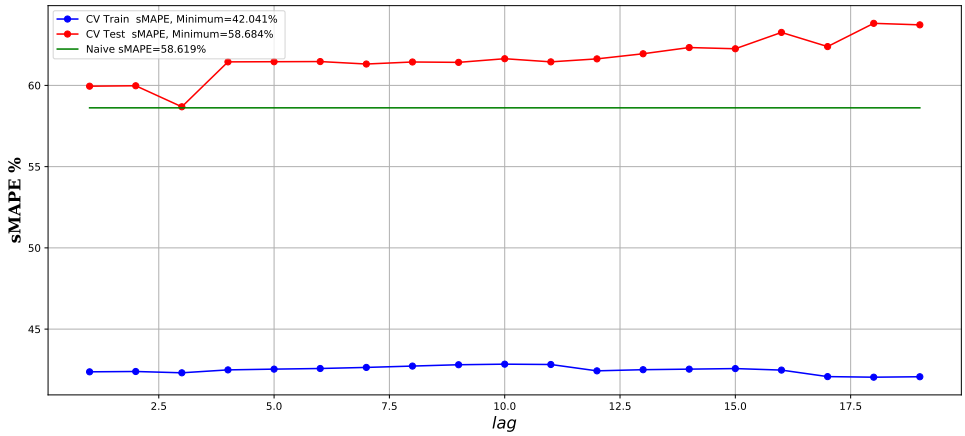

Figure 10: **Random variable E Model**: Ridge regression CV_{Rol}=100 α=0 lag_{max}

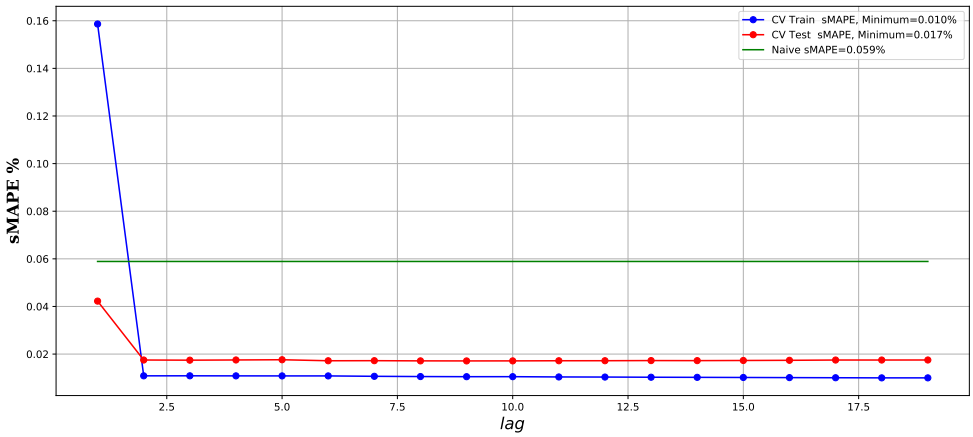

Figure 11: **Random variable T_t Model**: Ridge regression. CV_{Rol}=100. α=0, θ=lag,

can be view as column vector of H_{units} components). Each of the hidden units is a linear combination of the connection dictated by the weighting matrix with the previous layer; this linear combination is passed as an argument to the nonlinear activation function, known as sigmoids, such as the hyperbolic tangent *tanh* used in the present development.

We begin this modeling task by trying to implement the *gradient-decent* algorithm explained in Machine learning lecture notes, where the matrices W_1 and W_0 are randomly initialized and the *back propagation* algorithm is used to estimate the gradient of the loss function, which will in term be used to solve the minimization of loss problem. In this approach an update of the weighting matrices occurs once all the data point have bee cover and is proportional to a *learning rate* $\lambda \in \mathbb{R}^{>0}$, if this update occurs before all the data points have been covered, we have *batch gradient decent* implementation, if this *batch* of data points is randomly selected then we have *stochastic gradient decent*. Unfortunately, this implementation uses standard *cpu* hardware to make gradient updates (also know as *epochs*), making it very slow to converge. To bypass this inconvenience we decided to use *Keras* with *Tensor-flow*[17] to speed up the convergence time using *gpu* hardware while maintaining the same theoretical premises

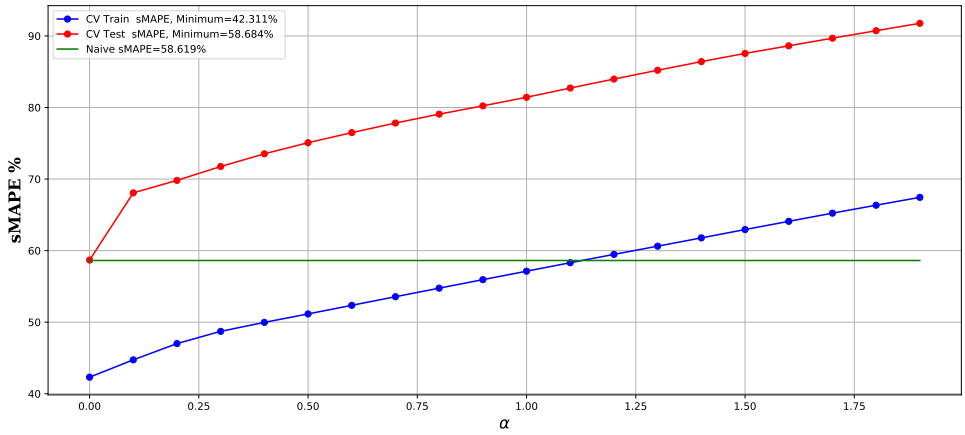

Figure 12: **Random variable E_t Model**: Ridge regression. CV_{Rol}=100. $lag_{max} = 3$, θ=α,

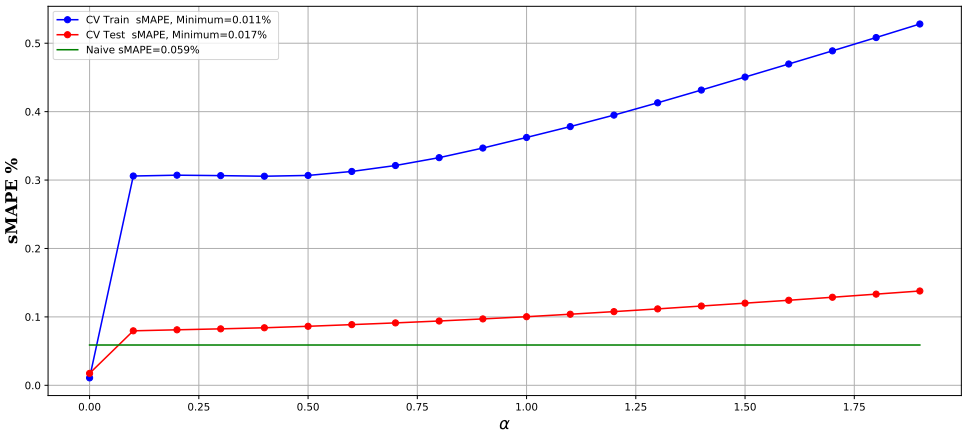

Figure 13: **Random variable T_t Model**: Ridge regression. CV_{Rol}=100. $lag_{max} = 3$, θ=α,

explained in the lecture notes, this means that *gradient decent* was forced(by setting the scholastic gradient descent with batch size equal to the number of data points, and preventing the shuffling of data points) with a fixed λ.

We set somewhat arbitrarily the rule in Equation 12, to have a simple guide for testing some models.

$$H_{units} * lag_{max} + H_{units} < N/10 \tag{12}$$

Following equation 12 is possible to reasonable select a maximum number of units in the hidden layer. Beyond this point setting a testing environment as the one shown in Ridge regression is futile (please see *Comments on* Θ section). Table 4 contains some of the parameters implemented for $\mathcal{N}$ ordered by test $\%(sMAPE)$, we see that best parameters seems to be $H_{units} = 6$, $Lag_{max} = 15$ which is 3.64% better than the *Naive forecast sMAPE: 58.61 %.*

some comments on Θ

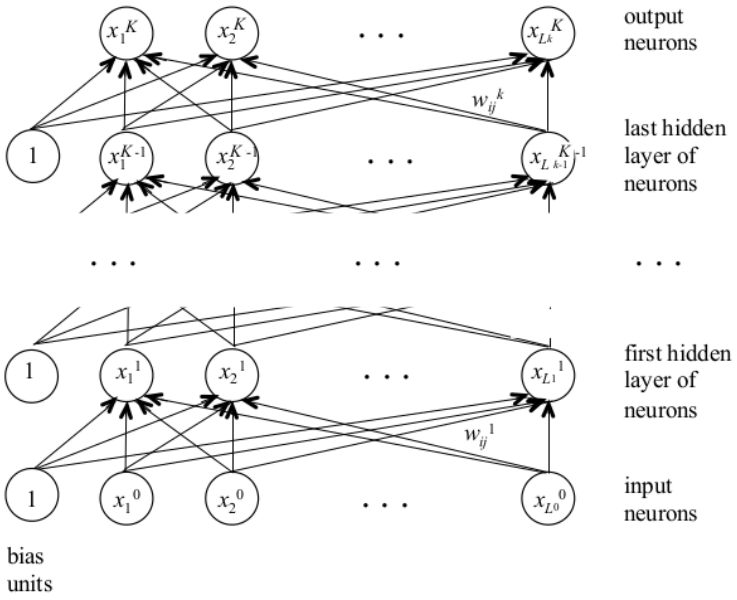

Figure 14: Schematic view of a general feed forward neuron network. Taken from [9].

H_{units}	Lag_{max}	λ	$Iter$	$Train(sMAPE)$	$Test(sMAPE)$
6	15	0.1	100000	42.83%	54.97%
6	15	0.1	12000	43.05%	56.47%
50	1	0.1	2000	43.88%	56.68%
16	5	0.1	8000	42.30%	57.17%
27	3	0.1	8000	42.07%	57.85%
12	7	0.1	5000	42.63%	59.39%
33	2	0.5	3000	42.59%	59.49%
12	7	0.1	10000	43.37%	59.92%
12	7	0.1	9000	42.41%	60.27%
25	3	0.01	10000	64.68%	64.68%
9	10	0.01	10000	49.84%	68.54%

Table 4: Θ Variation. *solver*: Gradient decent. W_o: normal distributed (Random seed:7). f_{act}: *tanh*. *Naive forecast sMAPE: 58.61%*

- A somehow rational approach would be to increment one by one the number of units in the hidden layer and then evaluate the cross-validation error. Each of this steps would have the same λ as well as the same number of iterations to be interpretable by us, however as we increase the number hidden units, keeping the same λ might not be appropriate giving that the vector of parameters ϕ has increased in dimension, this increase might cause the gradient descent update of state not as effective, which in term would might cause the number of iterations to be insufficient for convergence.
- Setting extreme small values λ to cover most scenarios requires massive amount computational time, setting λ relative big, will cause not convergence.

- Giving the random nature of the initialization of weighting matrices and the fact that gradient descent is very likely to wander around local minimal, compering in such straight forwards different manner architecture seems like undefined endeavor.

3.2 Decomposition, Simple Exponential Smoothing based

For simple exponential smoothing[6] trend calculation, the parameter β in Equation 9 determines the amount of exponential decrease of the influence of past observations over the current time step. For small values of β more history of past observations weight in the current time step value, whereas a large β would give more influence of observations near the present. In Figure 15 we can see different values of the smoothing factor β. A smoothing parameter of 0.5 was selected, and the process of decomposition describe in the previous section was repeated. Note that no attempt for calculating the seasonality was carried since this is a property of the time series itself, independent of the tool used for modeling the trend.

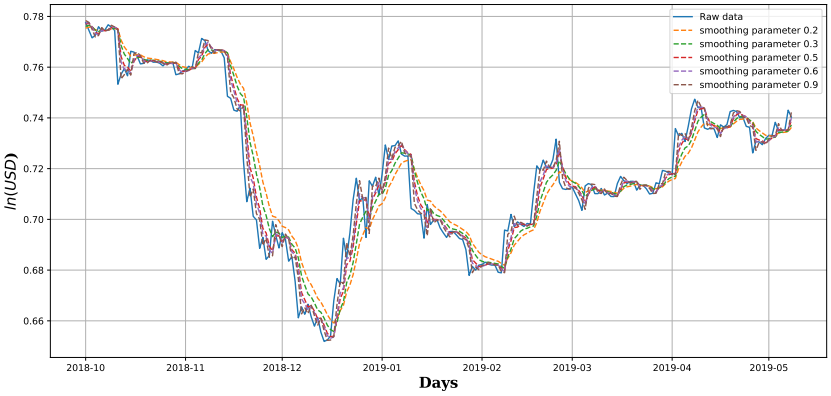

Figure 15: Simple Exponential Smoothing for year 2019

By inspecting Figure 16 where the derived error E^{ses} is plotted a noise behavior is clearly discernible as consequence we decided to apply a two tail statistical test over the null H_o that $y = T^{ses} + E^{ses}$ and $y' = T^{ses}$ are in fact the same random variable.

Table 5 summarizes the t-test. With a p-value so close to 1, it is not possible to reject H_o.The new relationship becomes $y = T^{ses}$.

RV	t-value	p-value
y,y'	0.11	0.90

Table 5: T-test for Error.

In Figure 17 we see cross-validation plot for lag_{max} optimization using Ridge regression, the minimum test $\%sMAPE$ is 0.245%.

3.3 Conclusion on Time Series Analysis

In other to compare both approaches, namely, classical decomposition moving averages(Equation 13) based and exponential smoothings(Equation 14) based, we make consecutive one time step predictions

[6]Implemented using [18]

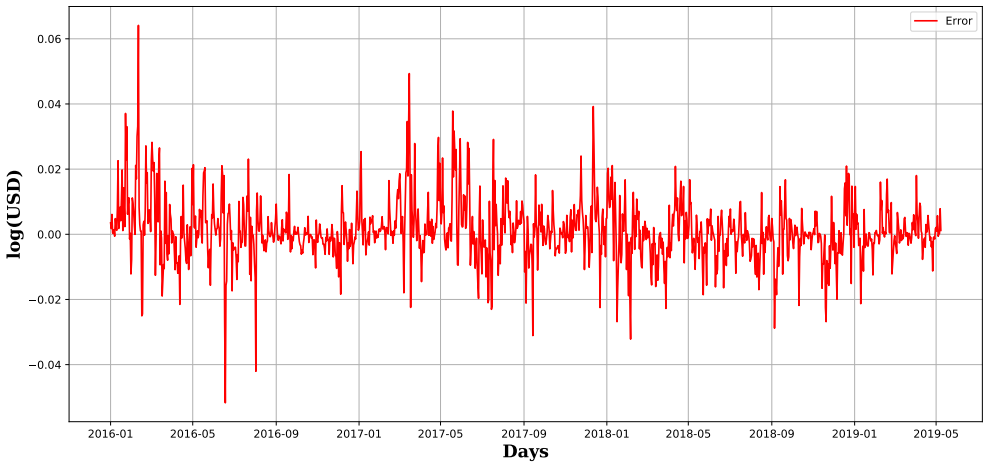

Figure 16: E_t Using exponential smoothing for the T_t

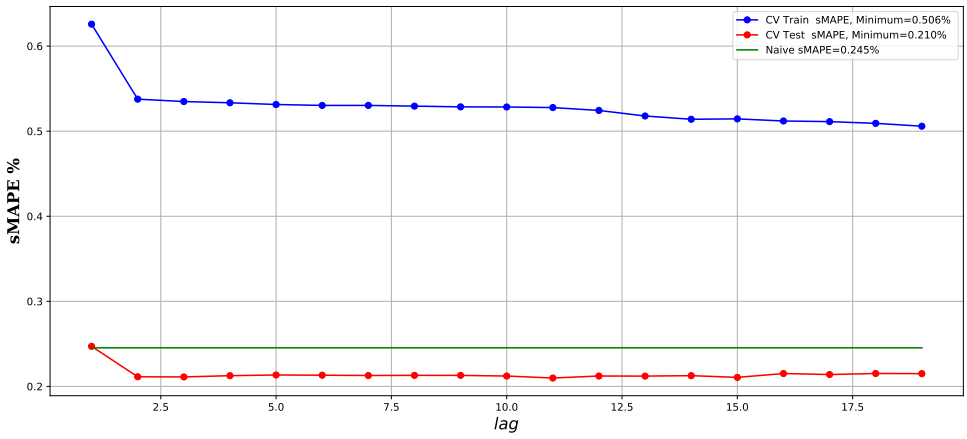

Figure 17: **Random variable** T^{ses} from Simple exponential smoothing **Model**: Ridge regression. CV_{Rol}=100. α=0, θ=lag.

on the year 2019 following Table 6.

$$y_t^{mv} = T_t^{mv} + E_t \tag{13}$$

$$y_t^{ses} = T_t^{ses} \tag{14}$$

RV	$\mathcal{D}$	lag_{max}
T^{mv}	Ridge regression ($\alpha = 0$)	2
E	Single layer perceptron ($H_{units} = 6, \lambda = 0.1, Iter = 12000$)	15
T^{ses}	Ridge regression ($\alpha = 0$)	2

Table 6: Best decision functions per Random variable

In Figure 18 and 19 we plotted the full prediction for the latest three months of 2019. The $\%sMAPE$ for y^{mv} is 0.42% for y^{ses} is 0.62% and the naive forecast is 0.44%. The final results are somehow disappointing; however, not unexpected since it is a well-accepted fact that markets are irrational in the sense that people behave erratically, and this information is not on the time series itself. This could be further amplified, giving that the cryptocurrency market place is accessible to every individual on the planet, which is not true for other assets, making ever more difficult to predict.

4 Modeling the Ether price with social media

We dedicate this section to the inclusion of sentiment in our analysis. As shown in the *Time series* section, the *naive forecast* was never outperformed by a sufficient $\%sMAPE$ to make the effort of implementing more complex decision functions a reasonable endeavor. In this section, we take the naive forecast as the next day predictor plus a level of hysteria hl of the previous day.

$$y_{t+1} = y_t + hl_t \tag{15}$$

Equation 15 tell us that the level of hysteria is defined as $hl_t = y_{t+1} - y_t$. We also theorized that this random variable is not time dependent and that it is the direct consequence of what people express in social media. A final extra factor is the transitions per day T_r, which we believe is an indicator of the next volatility, the idea is that when a cryptocurrency owner start moving their digitals access over the blockchain from private wallets to exchange wallets, for example, it seems reasonable to think they are getting ready to sell it in exchange for *fiat* as USD, which may put extra pressure over the price according to supply and demand schema. Combining all this hypothesis we arrive at Equation 16 where Tr is the number of transactions per day, I is popularity on Google search, and Tw is the popularity score on Twitter, for more details over these variables please see section 1.3.

$$hl_t = y_{t+1} - y_t = Tr_t + I_t + Tw_t \tag{16}$$

A *Decision tree regressor* is a supervised machine learning algorithm that splits the target value space into sectors called *leafs* by continuously splitting the feature value space into binary *branches*. By increasing the maximum level of binary splitting of the features, we get more leaves or terminal nodes, meaning that the hypercube containing target variable get smaller producing easily overfitted models, for this reason, Decision Trees are considered weak estimators. an *Assembly methods* as *Boosting* combine many of those differently weak estimators in sequence based on the residual of the previous week estimators. A complete technical explanation can be found in [19]. As explain section 1.2, we only had data for social media analysis from 2018-01-19 to 2018-12-03 from 2019-03-16 to 2019-05-01.

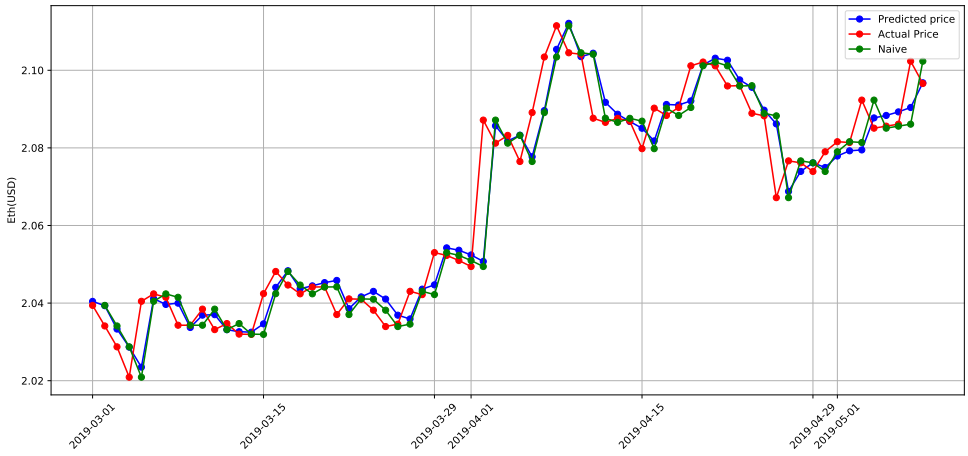

Figure 18: y^{mv} for 2019-03 to 2019-05

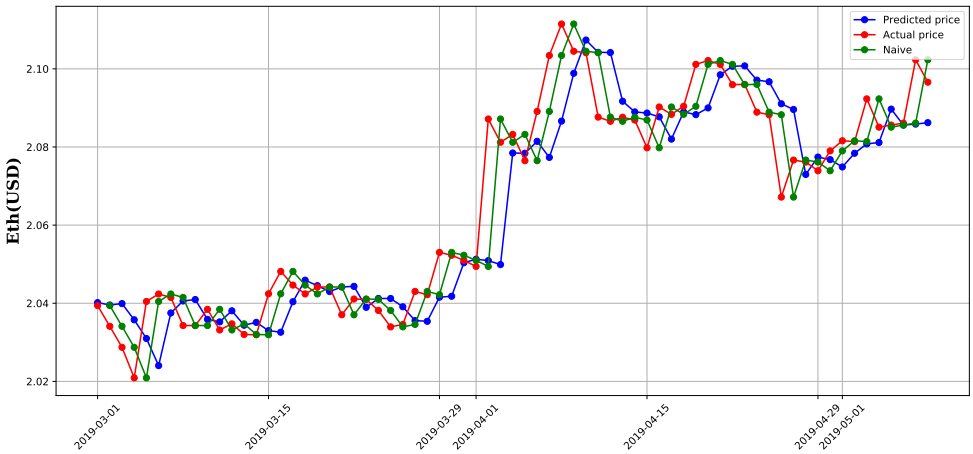

Figure 19: y^{ses} for 2019-03 to 2019-05

Since the data was splited in this manner, we decided to train *Boosting Decision tree regressor*[7] on the data 2018 data interval, and then run a single test of the 2019 interval. For parameter optimization we used grid search sklearn implementation [21].

Parameters of grid search:

- *learning rate*: 0.01,0.02,0.03
- *max depth*: 20,25,30,35
- *n estimators*: 5,10,25,50

where the *learning rate* is the gradient proportional constant for minimizing the loss function in the case de mean square error on each subsequent tree. *max depth* is the maximum level of splitting

[7] We used the sklearn impementation in [20]

of the feature vectors per weak estimator, and *n estimators* is the number of weak estimators to use. Using cross 2-fold cross validation we the best paramters are: *learning rate*: 0.01, *max depth*: 25, *n estimators*: 5.

4.1 Conclusion on Modeling the Ether price with social media

The test $\%sMAPE$ on the price prediction for 2019 was 9.9%. The naive forecast scored 8.4%.

5 Conclusions

- Unfortunately none of the implemented techniques were able to outperform the naive forecast. However, we still believe that through social media, the erratic behavior of the price can be predicted.
- We believe that the reason for the somewhat disappointing results for social media comes from the fact that the scoring systems of sentiment analysis were poorly designed. Techniques related to natural language processing should be implemented.
- Using Single layer perceptrons with enough processing power showed signs of being able to surpass the naive forecast.

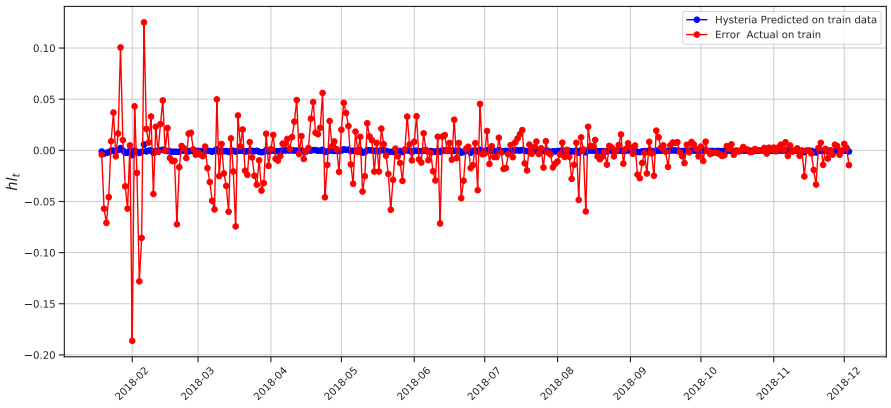

Figure 20: hl_t for 2018-01-19 to 2018-12-03 training set.

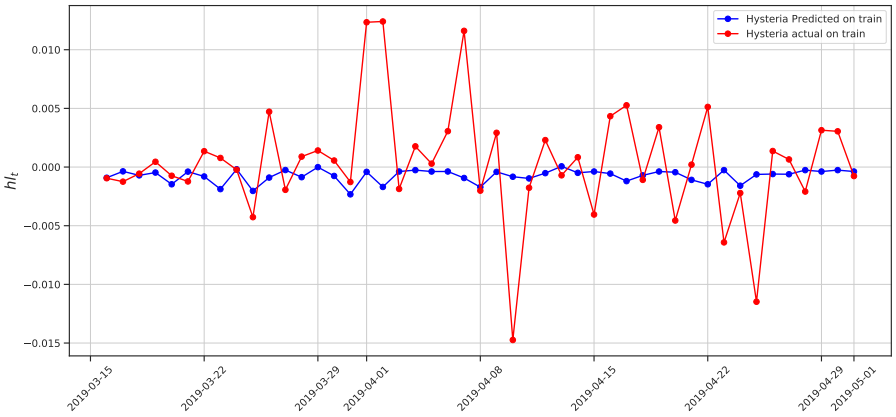

Figure 21: hl_t for 2019-03-16 to 2019-05-01 test set.

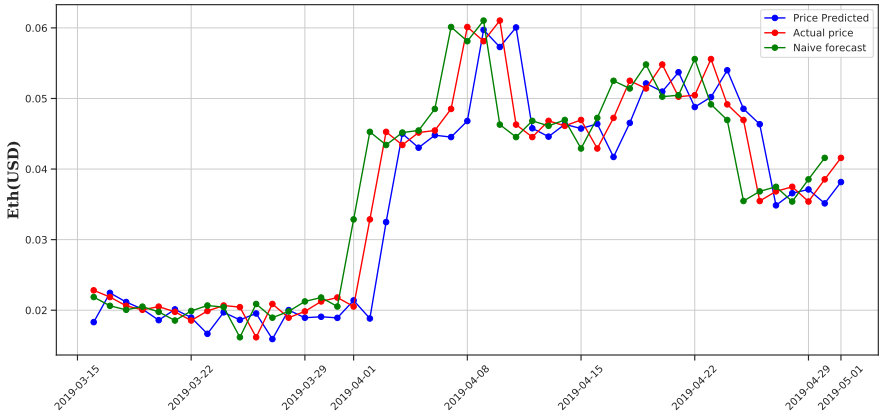

Figure 22: Scaled ether price for 2019-03-16 to 2019-05-01 test set.

Appendix A Code Repository

All the code implemented and utilized during the execution of the data workflow described in this report is available at the GitHub repository https://github.com/gariciodaro/advanced-project.

References

[1] Melanie Swan. *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.", 2015.

[2] Satoshi Nakamoto et al. "Bitcoin: A peer-to-peer electronic cash system." In: (2008).

[3] Dejan Vujičić, Dijana Jagodić, and Siniša Ranđić. "Blockchain technology, bitcoin, and Ethereum: A brief overview." In: *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*. IEEE. 2018, pp. 1–6.

[4] Vitalik Buterin et al. "Ethereum white paper: a next generation smart contract & decentralized application platform." In: *First version* (2014).

[5] Coinmarket. *Principal markets indicator for cryptocurrency*. 2019. URL: https://coinmarketcap.com/coins/ (visited on 05/20/2019).

[6] Matthew Tan. *Ethereum Charts and Statistics*. 2019. URL: https://etherscan.io/charts/ (visited on 05/08/2019).

[7] Ciodaro Gari. *Social media data on Ethereum*. 2019. URL: http://deepblueai.pythonanywhere.com/ (visited on 05/08/2019).

[8] K Goshima, H Takahashi, and T Terano. *Estimating financial words negative-positive from stock prices*. 2015, pp. 395–419.

[9] Herbert Jaeger. *Machine learning lectures notes fall 2019*. 2019. URL: http://minds.jacobs-university.de/teaching/courses/t2019ml/ (visited on 05/08/2019).

[10] Tilmann Gneiting. "Making and evaluating point forecasts." In: *Journal of the American Statistical Association* 106.494 (2011), pp. 746–762.

[11] Spyros Makridakis and Michele Hibon. "The M3-Competition: results, conclusions and implications." In: *International journal of forecasting* 16.4 (2000), pp. 451–476.

[12] Andreas C Müller, Sarah Guido, et al. *Introduction to machine learning with Python: a guide for data scientists*. " O'Reilly Media, Inc.", 2016.

[13] Spyros Makridakis, Steven C Wheelwright, and Rob J Hyndman. *Forecasting methods and applications*. John wiley and sons, 2008.

[14] Steven C Chapra and Raymond P Canale. *Numerical methods for engineers*. Vol. 2. Mcgraw-hill New York, 1998.

[15] Rob J Hyndman and Anne B Koehler. "Another look at measures of forecast accuracy." In: *International journal of forecasting* 22.4 (2006), pp. 679–688.

[16] The Scikit-Learn community. *sklearn Ridge*. Apr. 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html.

[17] François Chollet et al. *Keras*. https://keras.io. 2015.

[18] The Statsmodels community. *Statsmodels Simple Exponential Smoothing*. Apr. 2019. URL: https://www.statsmodels.org/dev/generated/statsmodels.tsa.holtwinters.SimpleExpSmoothing.html.

[19] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[20] The Scikit-Learn community. *sklearn Gradient Boosting Regressor*. Apr. 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html.

[21] The Scikit-Learn community. *sklearn Grid SearchCV*. Apr. 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.