Time Series Analysis of Cryptocurrency Data and Design of a Trading Algorithm using Ethereum Coin

Authors: Atacan Kavdır, Furkan Kasapoğlu

Date: January 3, 2021

Introduction

With advancements in technology, virtual currencies started to become a part of daily life. The cryptocurrencies aim to eliminate financial intermediaries by direct peer-to-peer transactions. With the lead of Bitcoin, Ethereum, Ripple and Litecoin, more investor started to invest for cryptocurrencies every day. Cryptocurrencies had $850 billion market size in 2020 and it is projected to have $1,758 billion by 2027. Nowadays, there are thousands of cryptocurrencies exist and the number is increasing every day. On the contrary of other financial instruments, cryptocurrencies do not represent tangible assets. For this reason, forecasting their future values are more difficult than stock price of a company with declared financial records. Therefore, highlighting potential patterns of the exchange prices by learning from their short history becomes crucial. The aim of the project is to predict and trade on closing price of Ethereum Coin, which is one of the most popular cryptocurrencies. There is a common belief that all cryptocurrencies are affected fluctuations of giant cryptocurrency Bitcoin. For this reason, adding Bitcoin prices changes to data can be helpful for Ethereum price forecasting. In addition, if cryptocurrencies become more popular, more people will wonder them and search them via search engine. The theory is that more popularity of cryptocurrencies in search engines, Google in this case, create more demand towards them and eventually this will increase their prices. To investigate that, Google Trends search popularity dataset for some keywords are added to the dataset. By using these data sources, making more profit than Buy&Hold strategy is aimed. To do that, after developing a forecast model, a trading algorithm will be designed, and the model outputs will be used through this system. In a designed system, because of complexity issues, we decided to predict closing price on daily basis. According to predictions, our trading algorithm will decide to do 3 different commands*: buy, sell,* and *do nothing.* With these commands and power of the model, the project aim to get a sustainable profit in Ethereum market.

Data

Data consists of multivariate time series of two different cryptocurrencies: Bitcoin and Ethereum.

Each data set consists of 5 columns:

- *high*: Highest price level of related date
- *low*: Lowest price level of related date
- *open*: Opening price of related date
- *close*: Closing price of related date
- *volume*: Total volume of transaction in terms of related coin

Original frequency of data is 1 minute. Due to practical issues like processing power and inability to trade so frequent, we aggregated the data into frequency of day.

After day aggregation, total number of rows was 1683. However, after a few data cleaning operations like missing value treatment, our final raw data properties are:

```
Start Date:  2016-08-10
End Date:  2020-12-23
Number of Data Points: 1597
```

Another important data source is Google search trends. Our data solely depends on the transaction history and date variables. For a more comprehensive model, additional data sources should be included. Google trends data shows the relative popularity of searches compared to its own historical trend. Our weekly trends data consists of 3 distinct words: ‘cryptocurrencies’, ‘ethereum’ and ‘bitcoin’. We are going to aggregate and analyze these features. Data interval is between 0 and 100. 0 means, there are no sufficient information and 100 means the keyword search is at its highest popularity level.

There are a few critical points to discuss about the data:

First of all, if we are going to use this data to create our models, we need to know time zone. If the user does not use the same time zone information to train or trade, she creates a certain bias so trading will be most likely to fail.

Secondly, source of data. If trading algorithm will work in real time with streaming data, system must use and also trade at the same data source due to possible data compatibility issues. In the data used for the project, features are generated as applicable to real time models and all are publicly available.

Exploratory Data Analysis (EDA)

Target variable of the data is daily close price of Ethereum, others are independent variables. We are going to deep dive and try to understand the data.

Data Summary

Features

date	high	low	volume	open	close	high_btc	low_btc	volume_btc	open_btc	close_btc
2016-08-10	12.33	11.000	137979.0	11.00	11.92	614.50	579.00	21352.0	579.00	590.28
2016-08-11	12.10	11.569	101344.0	11.93	11.71	599.75	584.59	15296.0	590.50	591.27
2016-08-12	11.99	11.565	75096.0	11.71	11.69	593.00	584.00	6288.0	591.26	585.50
2016-08-13	11.76	11.400	40661.0	11.69	11.49	590.02	582.00	6420.0	585.50	583.73
2016-08-14	11.67	11.003	52993.0	11.49	11.16	583.75	564.00	7530.0	583.73	569.41

Summary Info

```
Index: 1597 entries, 2016-08-10 to 2020-12-23
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   high        1597 non-null   float64
 1   low         1597 non-null   float64
 2   volume      1597 non-null   float64
 3   open        1597 non-null   float64
 4   close       1597 non-null   float64
 5   high_btc    1597 non-null   float64
 6   low_btc     1597 non-null   float64
 7   volume_btc  1597 non-null   float64
 8   open_btc    1597 non-null   float64
 9   close_btc   1597 non-null   float64
```

We can observe that, there are no null values in our data and all of our features are floats. Features at "_btc" suffix are about Bitcoin prices, the others are about Ethereum prices.

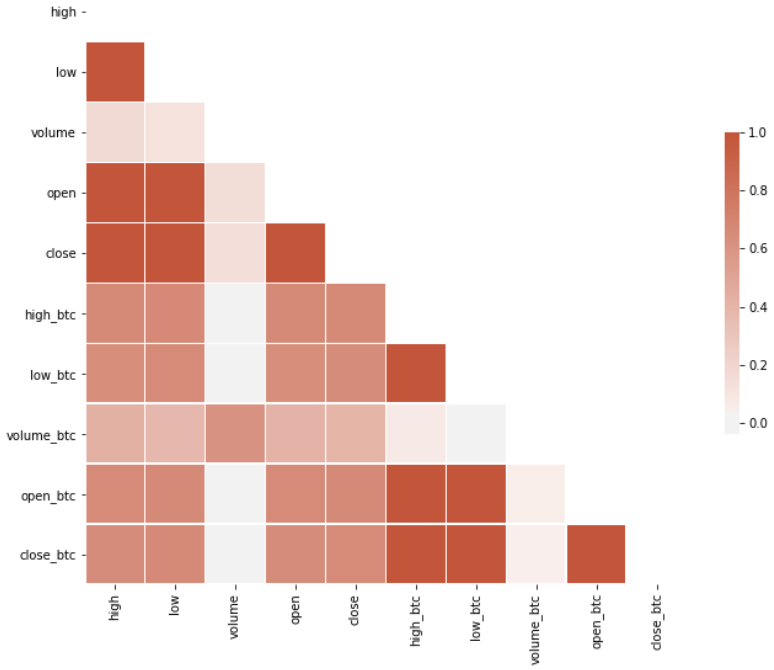
	high	low	volume	open	close	high_btc	low_btc	volume_btc	open_btc	close_btc
count	1597.000000	1597.000000	1.597000e+03	1597.000000	1597.000000	1597.000000	1597.000000	1597.000000	1597.000000	1597.000000
mean	280.414611	257.325976	1.644958e+05	269.697817	270.073033	6986.845463	6572.665873	21144.241703	6788.405898	6802.646296
std	236.418298	212.157856	1.722693e+05	225.631449	225.710160	4412.059475	4115.559747	23254.835670	4271.830797	4289.292352
min	7.323900	5.861000	2.393000e+03	6.776300	6.786400	574.540000	562.990000	813.000000	564.550000	564.640000
25%	140.400000	129.700000	5.449500e+04	135.300000	135.400000	3734.000000	3546.900000	5753.000000	3639.100000	3642.300000
50%	223.000000	206.770000	1.137930e+05	214.600000	215.120000	7153.600000	6670.000000	12448.000000	6880.600000	6884.500000
75%	368.400000	343.280000	2.055340e+05	354.260000	354.340000	9628.600000	9185.100000	28090.000000	9402.100000	9408.000000
max	1424.300000	1264.000000	1.630369e+06	1380.000000	1379.900000	24244.000000	23354.000000	192768.000000	23804.000000	23808.000000

Interval of the data features are quite different, for example while max and min of high feature are 7 and 1424, max and min of high_btc feature are 574 and 24244. These indicates a scaling operation would be beneficial.

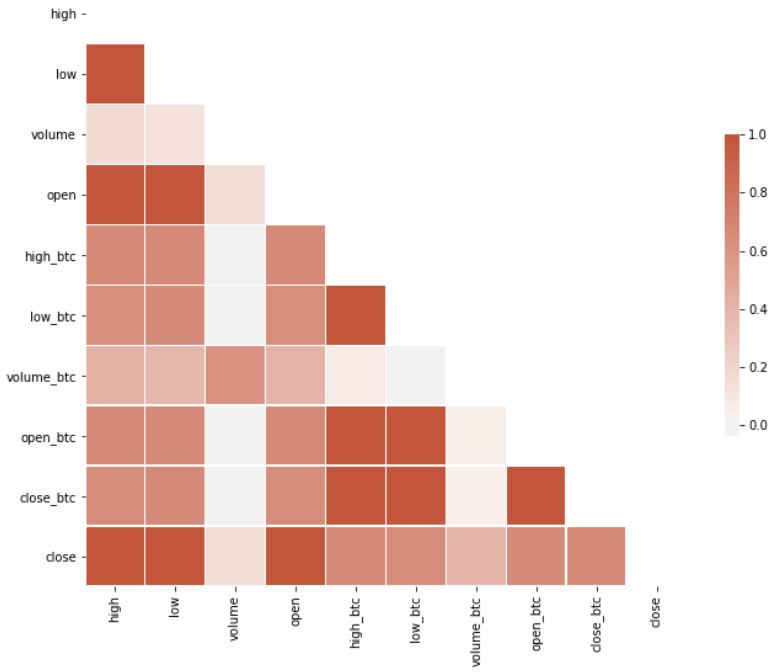
Correlations

Independent features of our data show a strong correlation with our target “close”. However, we are going to use lagged values of our features. So, we also need to visualize the relation between target and lagged independent features. According to below correlation matrices, lagged values also show a strong correlation with target variable.

Correlation of Actual values



Correlation of Lagged values



SFA of each independent variable compared to target

According to graphs below:

- The correlation between ethereum high, low, open, and close are high as expected. In addition, it would be useful to use lags of those values.
- Volume columns do not follow the same distribution and trend with close price. Feature engineering would be beneficial to extract related information.
- Bitcoin price features show a lagged correlation with Ethereum close price. In the data, Bitcoin acts as a leading indicator.
- Unlike Ethereum volume feature, Bitcoin volume feature shows better fit with Ethereum closing price.

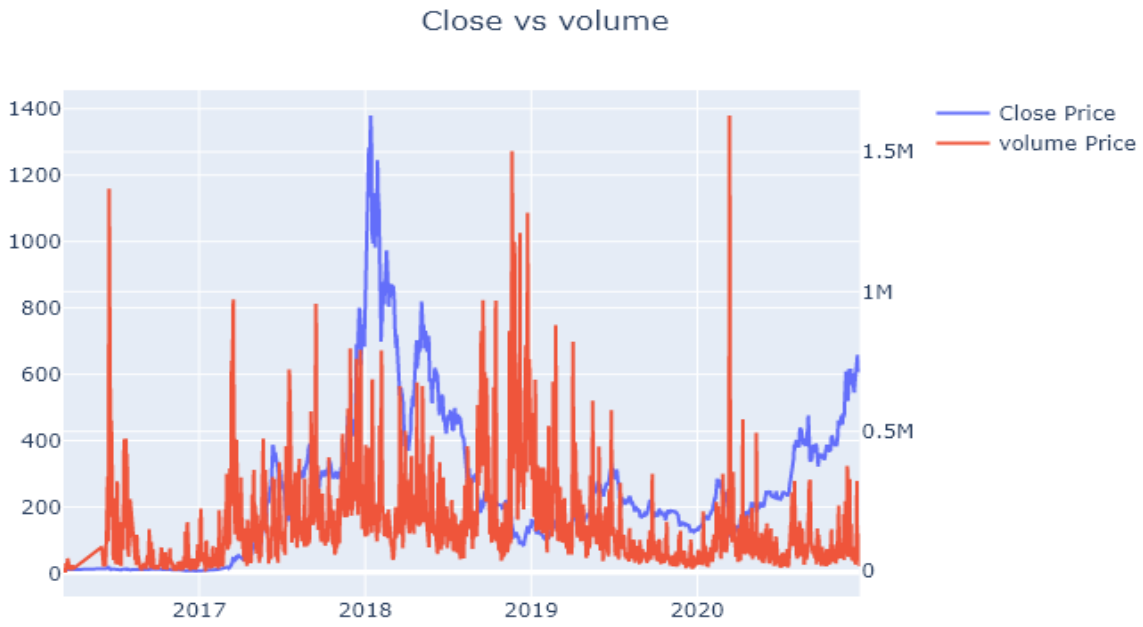

Close vs high_btc
1400
1200
1000
800
600
400
200
0
25k
20k
15k
10k
5k
0
2017
2018
2019
2020
Close Price
high_btc Price

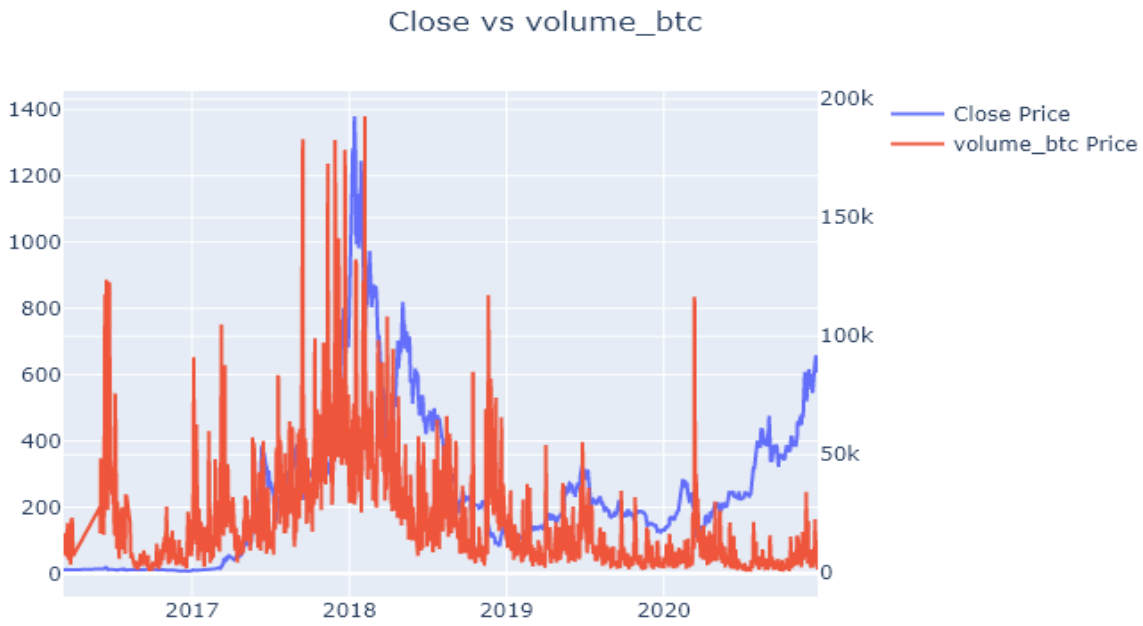

Close vs low
1400
1200
1000
800
600
400
200
0
1200
1000
800
600
400
200
0
2017
2018
2019
2020
Close Price
low Price


Close vs low_btc
Close Price
low_btc Price
1400
1200
1000
800
600
400
200
0
20k
15k
10k
5k
0
2017
2018
2019
2020


Close vs open
Close Price
open Price
1400
1200
1000
800
600
400
200
0
2017
2018
2019
2020


Close vs open_btc
1400
1200
1000
800
600
400
200
0
2017
2018
2019
2020
25k
20k
15k
10k
5k
0
Close Price
open_btc Price


Close vs volume
1400
1200
1000
800
600
400
200
0
2017
2018
2019
2020
1.5M
1M
0.5M
0
Close Price
volume Price


Close vs volume_btc
1400
1200
1000
800
600
400
200
0
200k
150k
100k
50k
0
2017
2018
2019
2020
Close Price
volume_btc Price

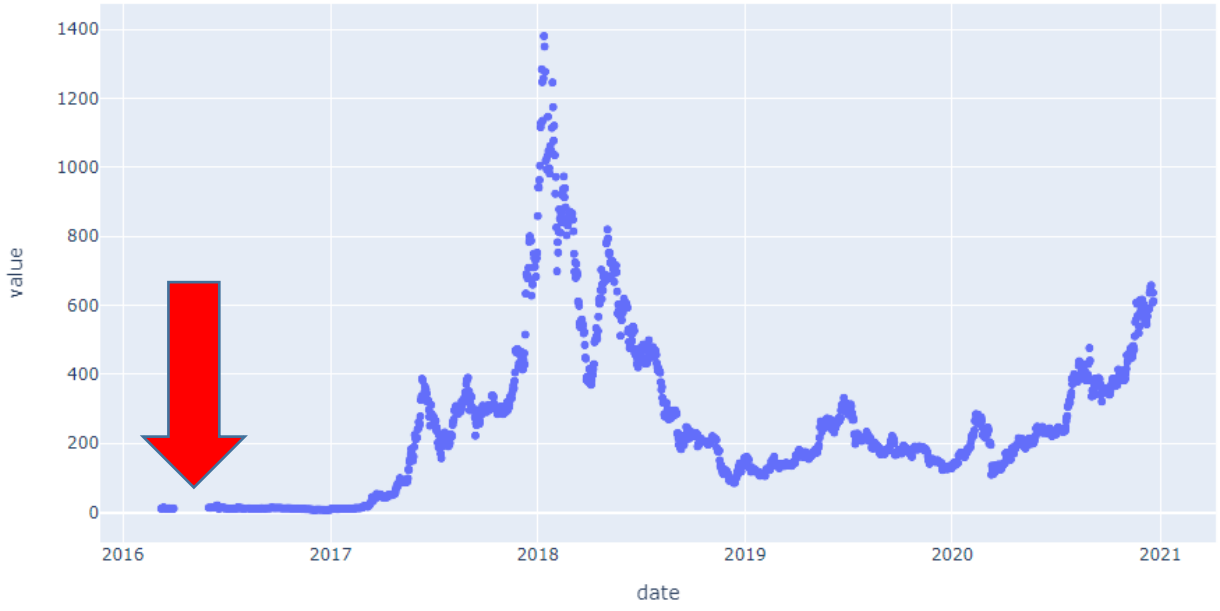

Close vs close_btc
1400
1200
1000
800
600
400
200
0
25k
20k
15k
10k
5k
0
2017
2018
2019
2020
Close Price
close_btc Price

Checking missing values

One of the main assumptions is the time series data shows a continuous distribution, so we need to check if data exhibits any gaps.

```
Start Date:  2016-03-09
End Date:  2020-12-23
Number of Days: 1750 days, 0:00:00
Number of Data Points: 1683
```

Our data starts at 2016-03-09 and ends at 2020-12-23. According to interval of start and end date it is re quired to have 1750 data points but there are only 1683. There is a gap or gaps in the time series data. By looking at the graph of our target, we could point out a gap inside the data.



According to above scatter plot, we observe 2 gaps in the beginning of the time series. First one ends at '2016-06-01' and the second one ends at '2016-08-10'. It looks like we can filter the starting part of the data without losing too much information. So, the initial part is dropped.

How final data interval looks like:

```
Start Date:  2016-08-10
End Date:  2020-12-23
Number of Days: 1597 days 00:00:00
Number of Data Points: 1597
```

Models

Autoregression and Moving average based models

Scaling

We scaled the data including target with MinMaxScaler using the interval of (0,1) for continuous varaibles. Regression based models perform better with scale data because of coefficient and cost optimizations.

Train-Test Split

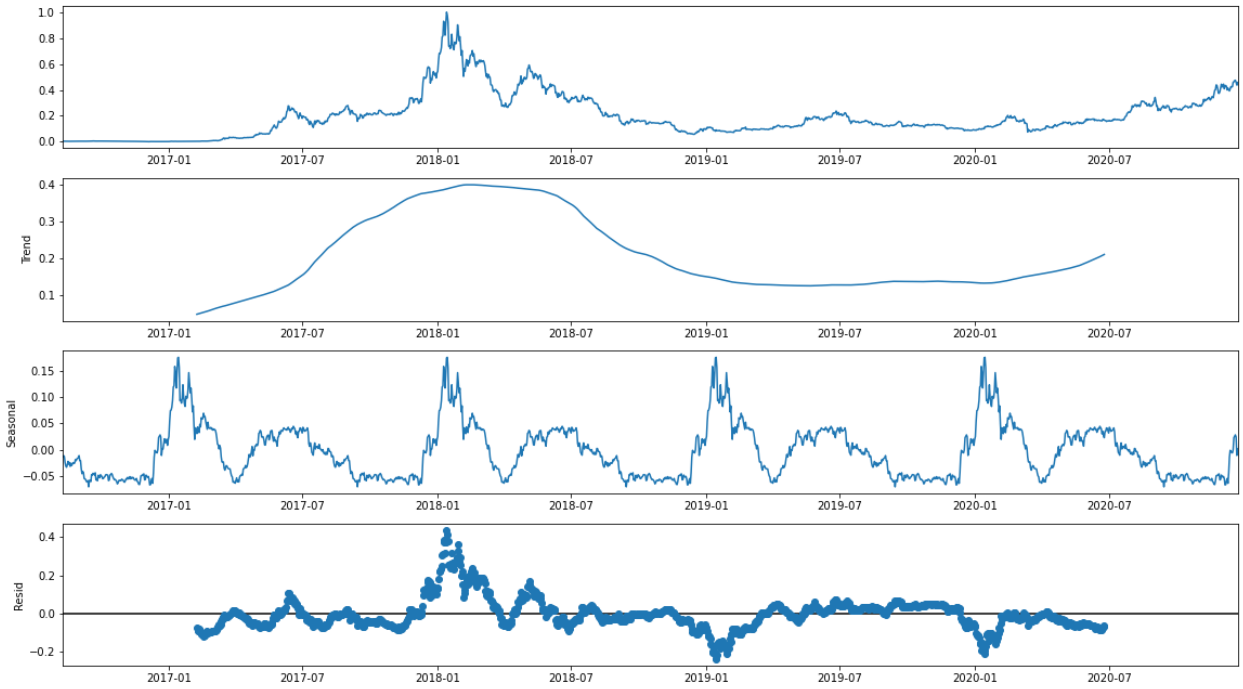
We split the data as 80% train and 20% test. While splitting, we did not use any shuffle operations because of the structure of time series data. Time series data is continuous by nature and shuffle will create a certain bias. The setting should be training with prior periods and test with next ones. A couple of problems that we would face if we shuffle the data, test dataset may include information from training dataset which can be memorized by machine learning algorithm. It creates bias and unsustainable model performance for incoming data.

Decomposition

One of the main assumptions of ARIMA is using stationary data makes predictions more accurate. We need to check if our data is stationary or not, according to next analysis we need to set p,q,d hyperparameters of the model.

Stationary check

We already know that our data is not stationary due to our plots. The data shows a trend and seasonal components. In our original graph seasonal components were not that clear to observe. However, according to decomposition plot, the coin prices are heavily affected by seasonality.



Adfuller

Augmented Dickey-Fuller test will be beneficial for validate stationary of the dataset. By examining results of Adfuller test, we can conclude that our data is not stationary. Our p-value is 0.21 so we are not able to reject null hypothesis.

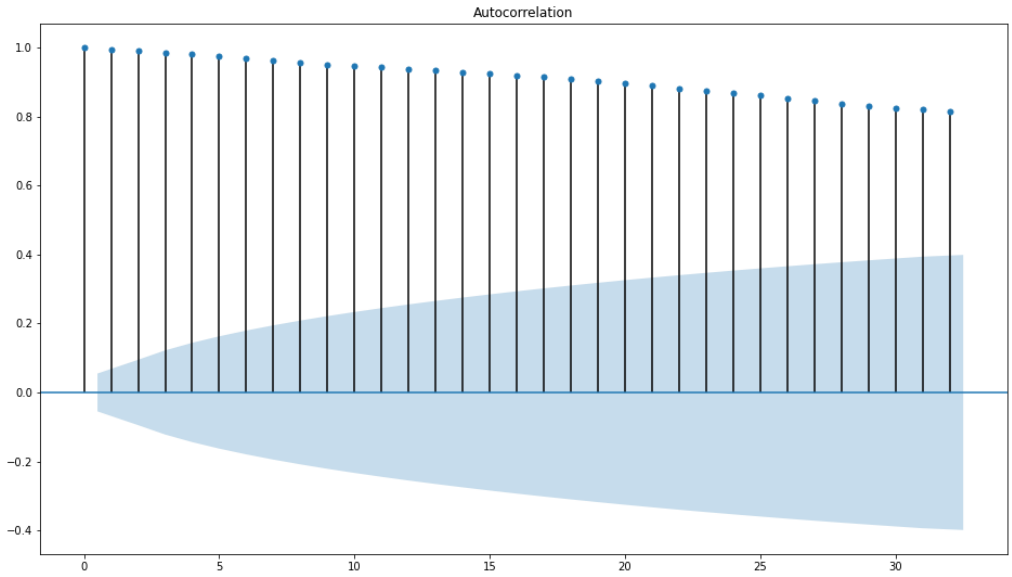
```
(z = -2.1866443670479336, p = 0.21112844072097503)
```

One of the solutions that we can implement is taking difference.
By taking 1st order difference, differentiated data shows stationary behavior.

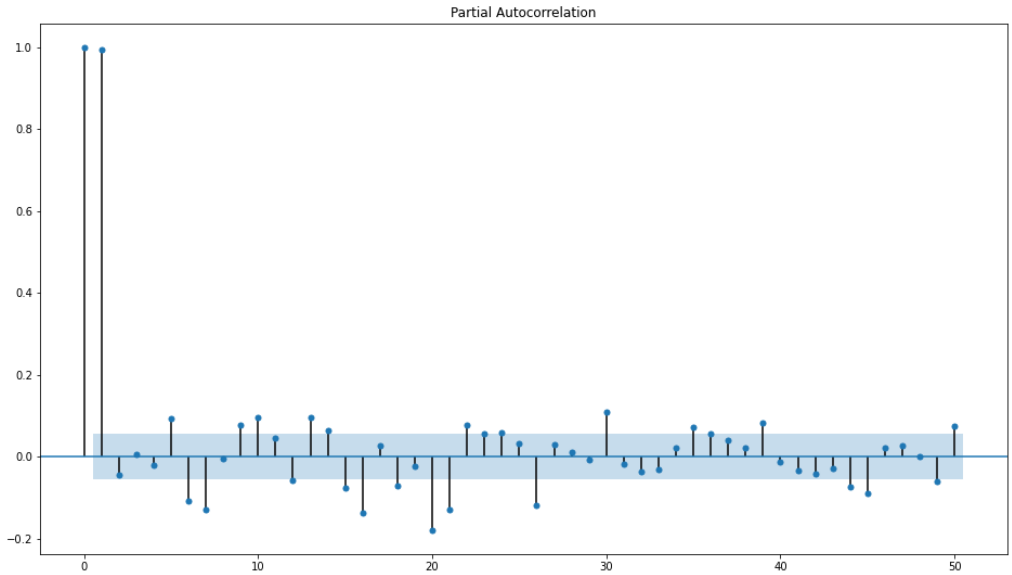
```
(z = -6.378907445382618, p = 2.247663037363501e-08)
```

After decomposition, ACF and PACF plots are needs to be checked to understand Auto regression and moving average patterns.

ACF & PCF



All lagged values are strongly correlated according to ACF.



PACF shows the benefit of adding one more lag to the data. According to above PACF plot, q = 1 looks quite convenient.

We have already learned that the data has seasonality, in addition to that, we also have additional exogenous features. As a result of those, SARIMAX can satisfy the requirements.

Also, we need to add one more term to consider the effect of seasonality: Fourier Terms.

Fourier terms basically mean encoding a time interval (hour, day, year etc.) using sine and cosine transformation. According to the decomposition, the data has a yearly seasonality. So, we added Fourier Terms for year. However, these Fourier features poorly performed because the seasonal components are not firmly periodic enough to follow a Fourier pattern.

After many trials, auto_arima searches and testing, the final ARIMA model is:

```
==============================================================================
Dep. Variable:                      0   No. Observations:                 1277
Model:                 ARIMA(0, 0, 1)   Log Likelihood                3380.102
Date:                Sun, 03 Jan 2021   AIC                          -6736.203
Time:                        10:23:03   BIC                          -6674.376
Sample:                    08-10-2016   HQIC                         -6712.985
                         - 02-07-2020
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0004      0.002      0.154      0.878      -0.004       0.005
0              1.0478      0.039     27.031      0.000       0.972       1.124
1              0.3168      0.032      9.909      0.000       0.254       0.379
2             -0.0094      0.009     -1.019      0.308      -0.027       0.009
3             -0.4122      0.050     -8.235      0.000      -0.510      -0.314
4             -0.7072      0.053    -13.289      0.000      -0.811      -0.603
5             -0.4438      0.043    -10.341      0.000      -0.528      -0.360
6              0.0163      0.008      2.086      0.037       0.001       0.032
7              0.5276      0.038     13.853      0.000       0.453       0.602
8              0.6449      0.037     17.565      0.000       0.573       0.717
ma.L1          0.2354      0.027      8.651      0.000       0.182       0.289
sigma2         0.0003   4.26e-06     69.358      0.000       0.000       0.000
===================================================================================
Ljung-Box (Q):                      234.18   Jarque-Bera (JB):             31381.56
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.60   Skew:                            -1.23
Prob(H) (two-sided):                  0.00   Kurtosis:                        27.16
===================================================================================
```

It looks like, by using lagged values of low, high, open, btc_open, btc_high, btc_low, btc_close features, we had already covered the auto regression information of the prior periods.

There is another point necessary to discuss. Our model solely depends on lagged_1 values. All exogenous and even moving average features are originated from lagged_1. This indicates a poor variety of features, so there is a possibility that we are not able to extract all available information from the data. Instead, we are creating a model that mimic -almost- every move of one day before. It is possible to understand by checking the graphs.

According to graph above, we can conclude an overfit by eye. To test the prior hypothesis, we can look into it a little bit closer.

Chart: Zoomed versions of the close price and predictions of Ethereum as x, x2 and x3 respectively

As it can be observed, the data exhibit a certain imitation of the prior day. The model performs well on the theory but, it is not certain about using this model on production because of the possible inconveniences.

- Model mimics the movements of one day prior, and not robust to sudden changes. Drastic decreases in a single day are very usual in crypto currency market. So, our model will not be able to predict and take precautions to avoid these unexpected losses by stop loss options.

Results of our model in terms of selected evaluation metrics:

```
Train MSE:  554.2822071533269
Train MAPE:  5.141077815979016
Test MSE:  288.87857100521114
Test MAPE:  3.8091107192647335
```

According to the results, the model predicts test dataset better which is related with the issue already discussed, less volatility means better predictions for the model.

We are going to discuss how to create a trading algorithm and optimize it according to trends in the last section.

Gradient Boosting Models for Time-Series

Boosting is an ensemble machine learning approach for regression and classification problems which turns weak prediction models into a stronger model. Generally, tree-based models are used for boosting. Each new tree is fitted on a penalized version of original dataset based on loss function results of previous trees. Gradient boosting use optimization of differentiable loss function on stage-wise structure of boosting techniques. In the project, regression and classification methods of XGBoost, which is an optimized gradient boosting library will be used. For time-series problem, regression is used to forecast next closing price of Ethereum, and classification is used to predict whether there will be an increase or decrease in the next day's closing price.

Feature generation

Time series features can be used to regenerate new features which carry extensive knowledge about historical changes and patterns which can be recognized by supervised machine learning algorithms to predict future dates. In feature generation, created features from historical data should be available in the prediction moment. Each data row must not have information from the future. Otherwise, such features cannot be applicable because user cannot provide this related information to the model at prediction moment because it will not be available for relevant day.

The main goal of feature generation is increasing the performance of XGBoost by using available or additional data. To create new features for XGBoost several techniques are used:

- Datetime features,
- Lag features,
- Statistical functions,
- Search engine trend features.

Datetime Features

Datetime features are created from date index of the dataset to represent seasonality and trend. Only date information is available because the data is aggregated on daily basis. With datetime transformation to create new features, the model may learn seasonality and trend in the time-series by understanding its relationship with not only target variable but also other variables. The features are created from datetime index are as below:

- Day of month (1-31),
- Month value (1-12),
- Quarter value (1-4),
- Semester value (1-2),
- Year,
- Day of week (1-7),
- Day of year (1-365),
- Week of year (1-52),
- Is weekend? (0-1)

Datetime features are an important feature set for time-series forecasting compared to other supervised models based on assumption of time-based relationship of observations, which may not exist in other type of problems.

Lag Features

In the dataset, there is daily information of close, high, and low prices and volume level for both Ethereum and Bitcoin. Daily lag features are created to represent previous days' information in the next rows as features. The main assumption is that previous events can contain a pattern or an information about the future. For example, after three consecutive increase on the price of Bitcoin, investors may do profit sales to earn their profits and invest to Ethereum. If such behavioral patterns exist in the data, it can be understood with lag features. Lag feature generation is also known as sliding window method. Such features represent previous features' summary about previous days.

Up to 20 days lagged features are generated from following features:

- Close price of Ethereum (target variable),
- Close price of Bitcoin,
- Daily volume of Ethereum,
- Daily volume of Bitcoin,
- Daily highest price of Ethereum,
- Daily highest price of Bitcoin,
- Daily lowest price of Ethereum,
- Daily lowest price of Bitcoin

Open price features are not generated as lagged features because they are almost the same price with previous day's close price. This is because cryptocurrency stock markets are 7/24 open, which is different than financial stock markets.

Additionally, lagged features are also created from the datasets from search engine trends, which will be covered in the following topics.

Statistical functions

Close price of Ethereum and Bitcoin are encoded with some statistical functions to understand the level of prices over the period. To do that, rolling window and expanding window statistics are applied.

Like lag features, a date range is defined to calculate a pre-defined statistics' minimum, maximum and mean values on the given date range. Moving average is used for rolling window features. For close price of Ethereum and Bitcoin, three features created in the last 5 days window:

- Min: minimum value observed in the last 5 days,
- Mean: average of values observed in the last 5 days,
- Max: maximum value observed in the last 5 days.

Expanding Window are features that include all previous historical data. It gives an advantage to understand historical information, which can be reached in the future too. For close price of Ethereum and Bitcoin, three features created by using all historical data exists previous than observation day for each row:

- Min: minimum value observed until the row's date,
- Mean: average of values observed until the row's date,
- Max: maximum value observed until the row's date.

To understand fluctuations in the last days some binary features are also generated. For example, if there is an increasing trend in the last 3 days, the feature "lag3_deep" turns to 1.

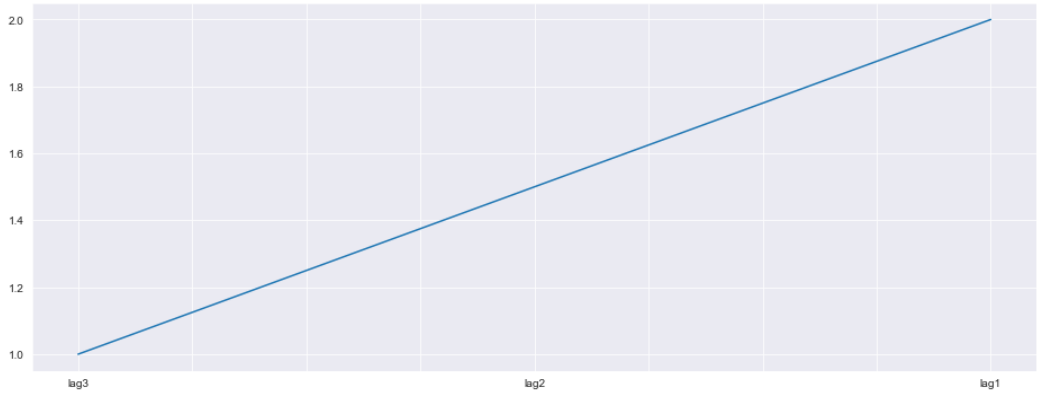


Chart: If the fluctuation in the last three days as above, "lag3_deep" feature returns 1

Another example is that if lag 3 has the lowest value and lag2 has the highest value, the feature "lag3_deep_lag2_peek" turns to 1.

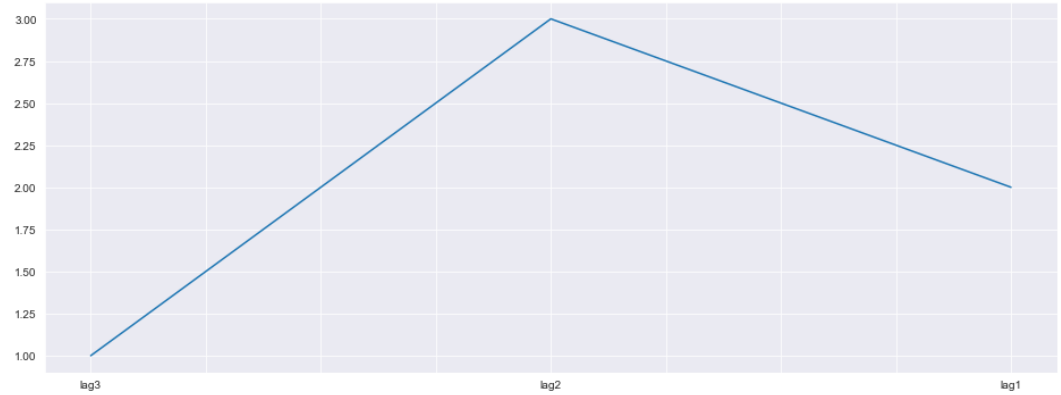


Chart: If the fluctuation in the last three days as above, lag3_deep_lag2_peek" feature returns 1

All possible combinations for the state of the last 3 days are generated as features to reflect possible patterns in the historical data.

Search Engine Features

Google Trends data is imported to understand the search popularity changes on the Web about "cryptocurrency", "Ethereum" and "Bitcoin" keywords, which may highlight major changes on investors' attention (especially new investors') about Ethereum before it happens. Google Trends ranks frequency of the searches between 0 and 100. 0 means lowest frequency and 100 means highest frequency, the most popular moment.

Google Trends provides weekly data for 5 years period. To make sure about availability of the Google Trends data after implementation, 7 days lagged values are used for each observation. Therefore, there will not be an information from future and all information will be available for the observation moment. 20 days lagged features are created for both these three keyword features. Fluctuation of these three keywords compared to close price of Ethereum even with 7 lagged days looks promising. For example, Ethereum reached its historical highest level on January 13, 2018 with $1380. On December 24, 2017,

Ethereum was $660 and 7 days lagged Google trends value of “cryptocurrency” reached to its highest level 96 until that moment. In 20 days, Ethereum doubled its price.

Chart: Close price of Ethereum and Google search trends of “cryptocurrency” keyword on the world

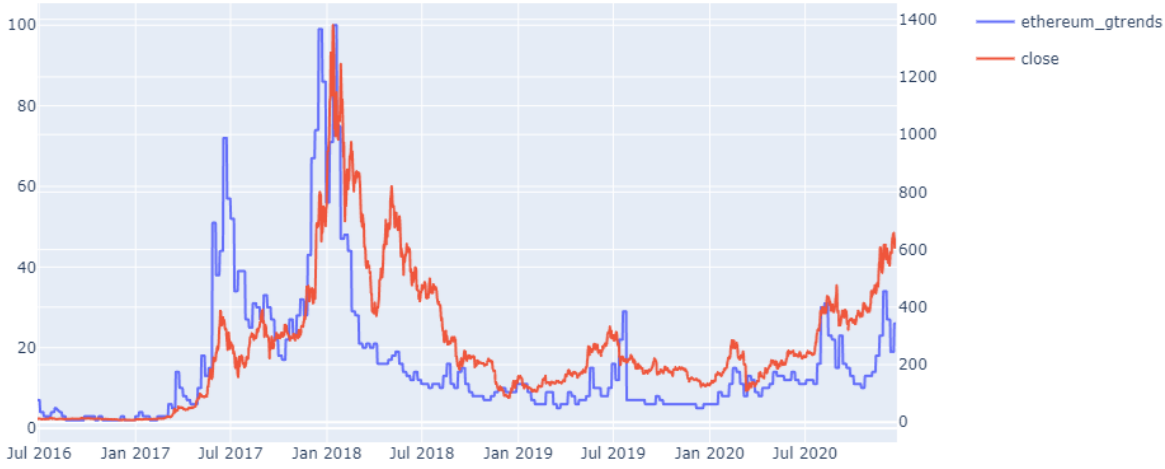


Chart: Close price of Ethereum and Google search trends of “ethereum” keyword on the world

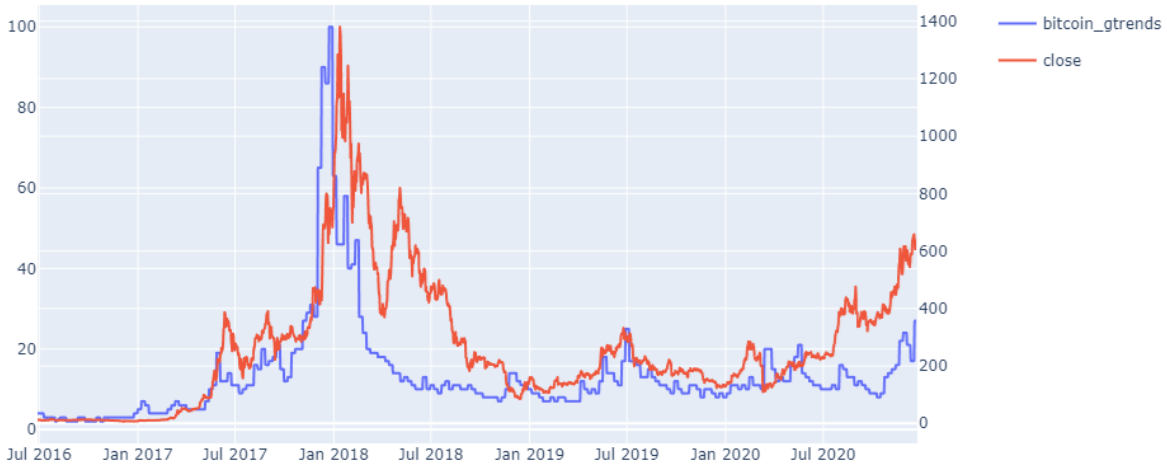


Chart: Close price of Ethereum and Google search trends of "bitcoin" keyword on the world

XGBoost Pipeline

Overview

There are two different approach applied as model. Firstly, a forecasting problem is solved by predicting next day's Ethereum close price. For this problem, XGBoost Regressor is used. Secondly, the problem is turned to a classification problem by identifying whether today's price is higher than yesterday's or not. Therefore, it may bring an advantage to reach higher profit levels in trading algorithm step. For classification problem, XGBoost Classifier is used.

After feature generation step, a pipeline is introduced including following steps:

- Pre-processing step including encoding and scaling
- Feature selection step
- Model

The purpose of the pipeline is to assemble several steps that can be cross validated together while setting different parameters. Pipeline implied to RandomizedSearchCV to optimize hyperparameters with cross-validation.

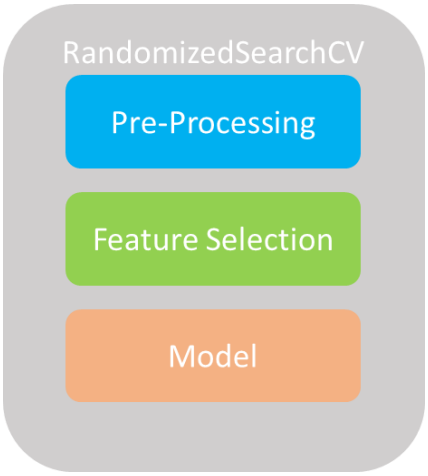


Figure: Pipeline steps in RandomizedSearchCV

Pre-Processing

In pre-processing step, features are grouped as categorical, circle and numerical features. Each group has sub-pipelines to pass through.

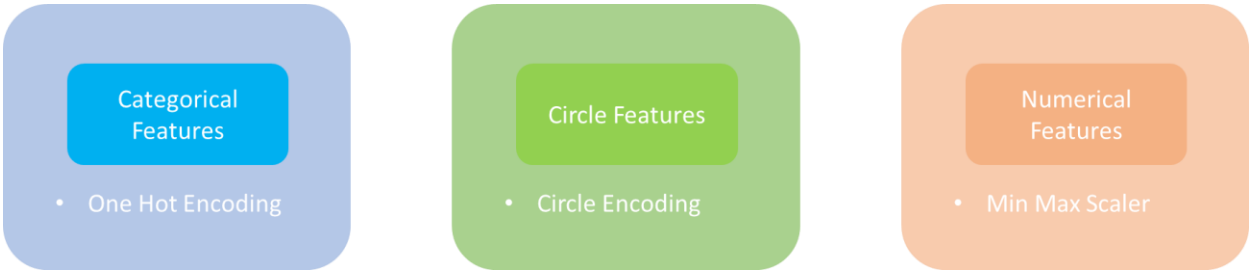


Figure: Pre-processing sub-steps

There are some categorical features from datetime feature generation such as month, day of week and quarter. For these features One Hot Encoding is applied. This creates a binary column for each category.

Some datetime features have a cycle such day. Day value 30 is closer to value 1 than value 15. If it is grouped as a numerical feature, it may lose the information of closeness of 30 to 1. For this reason, circle encoding (Cyclical encoding) is applied. It returns sine and cosine values on related cycle (30 for day value).

For the last group, Min Max Scaler is applied on numerical features, which transforms to range 0 and 1.

Feature Selection

For feature selection step, XGBoost is applied related to problem type. For given hyperparameters, feature importance weights are ranked, and features are selected based on threshold hyperparameter which is provided to the pipeline.

Model

In model step, XGBoost is applied related problem type with selected features in feature selection step. With RandomizedSearchCV, hundreds of iterations are done with 10 folds cross validation. Here, not shuffling rows is crucial because keeping consecutive dates in the same fold brings advantage of time-series and avoids from bias. Otherwise, validation dataset may include information from training dataset, which create optimistic bias and a potential for overfitting. With optimistic bias, we would fall into trap of systematically thinking the model is better than it is.

Results

Regression Problem

For regression problem, the model is developed with 17 features over 288 generated features. As expected, close price from 2 days ago and yesterday's high price have around 70% of relative feature importance. Cryptocurrency_lag16 feature indicates Google trends popularity for "cryptocurrency" keyword approximately 3 weeks ago. Month_1 feature indicates that whether month is January or not with 5% importance.

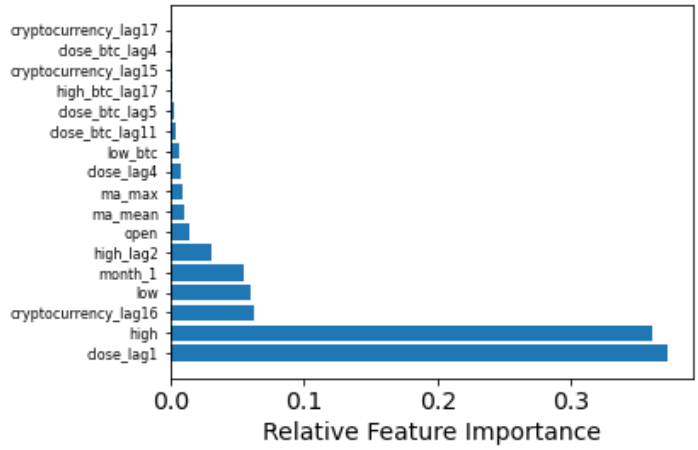


Chart: Relative feature importance of regression

The fluctuations of close price of Ethereum and predictions for both model development and out-of-time datasets are at below.

Chart: Close price and prediction in model development and out-of-time datasets

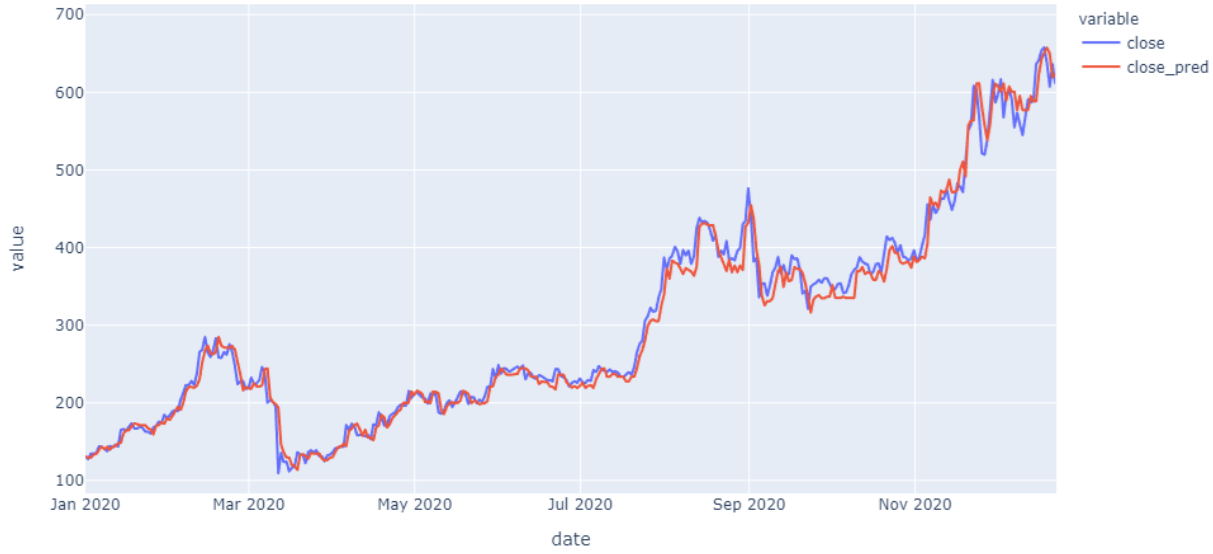


Chart: Close price and prediction in out-of-time datasets (After January 2020)

In distribution of residuals over date in x-axis. There is an increase in the level of residuals between January 2018 and July 2018, which is the period of sharp increase and decreases. In test dataset, there is an increase trend of the Ethereum price after March 2020. During this period model underestimates the

price, which is shown with positive residuals during this consecutive increase period. Mean absolute percentage error for training dataset is 3.52% and for test dataset 4.13%.

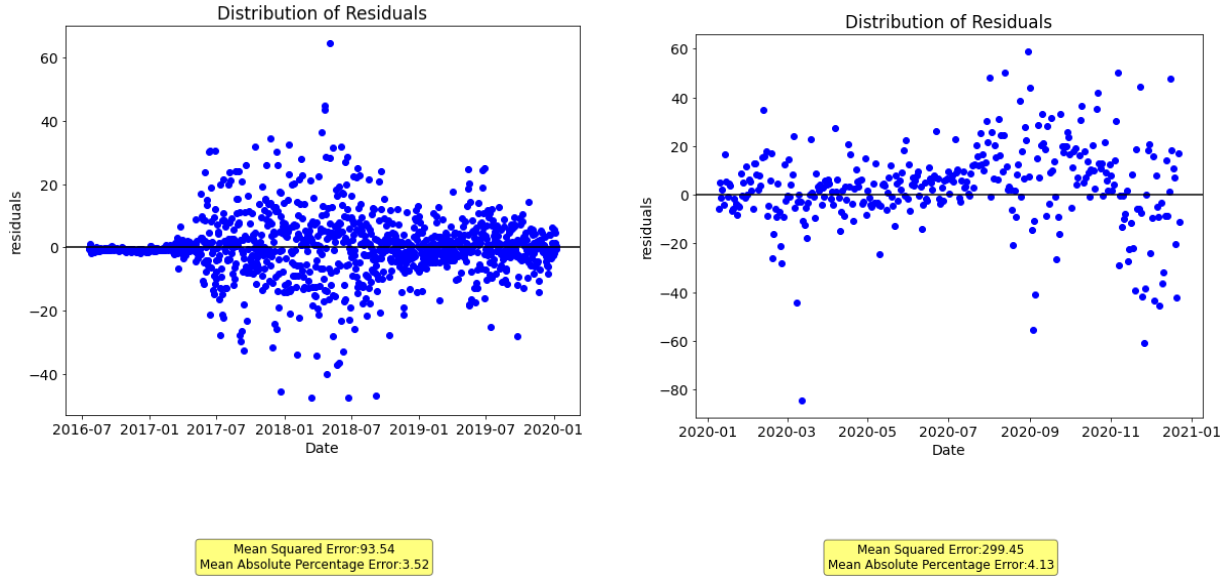


Chart: Residual line plot for training and test datasets respectively

The forecast errors to be normally distributed around a zero mean is expected. When residual histogram and kernel density estimation charts are drawn, the distribution does have a Gaussian look showing a positive kurtosis. There is a slight skew. A larger skew would be an opportunity for performing a transform to the data prior to modeling, such as taking the log or square root.

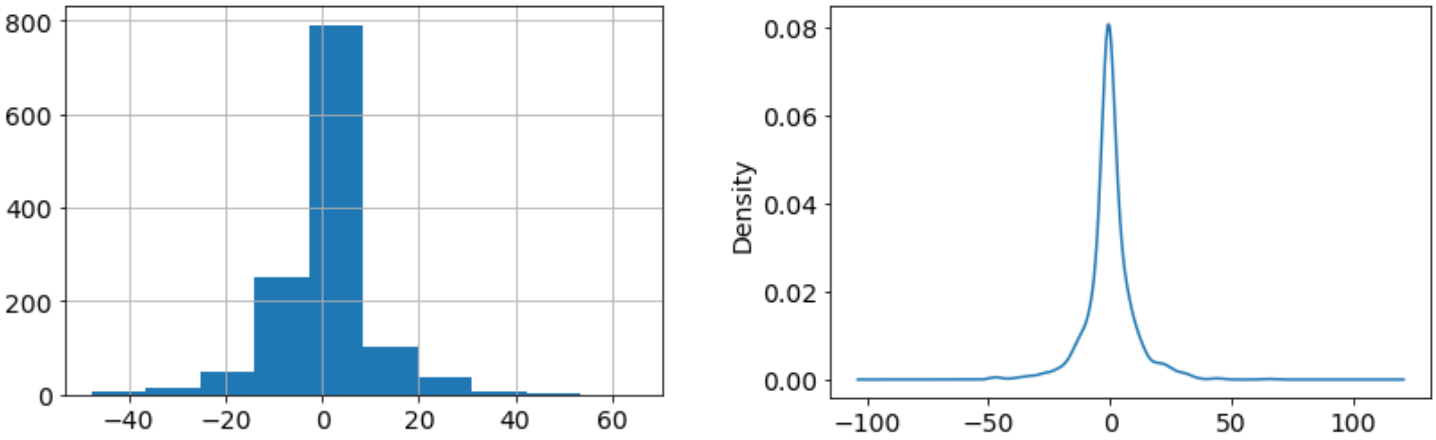


Chart: Residual histogram and kernel density estimation for training dataset

In test dataset, similar shape exists with more asymmetry on the left-hand side. It reaches to -150 while on the right-hand side value 100 is shown.

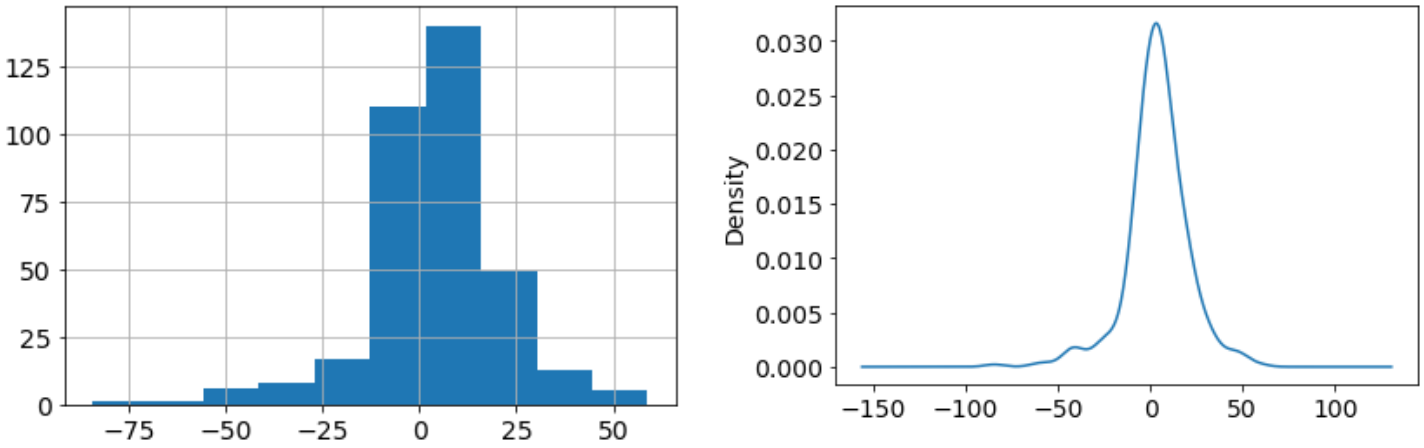


Chart: Residual histogram and kernel density estimation for test dataset

Classification Problem

As a classification problem, target feature "close" and other price related features are converted to binary features. If current row's value is higher than previous row's value, then it returns 1 otherwise it returns 0. It is applied for all price related features. After this transformation, average of target variable is 0.49 in training dataset and 0.54 in test dataset. It shows that dataset is balanced.

Even though classification with target transformation fits with trading strategy, it could not overperform regression model. In test dataset, the model stuck with randomness. Training dataset has huge overfit. Train ROC-AUC score was 0.78 and test ROC-AUC score was 0.54. Although ROC-AUC score has huge randomness for test dataset, the model is tried on trading algorithm to see its potential, which will be discussed under Trading Algorithm topic.

According to confusion matrix at below, the classification model can distinguish decreasing moves in the prices well. However, it is not able to detect increasing moves in the test dataset. There is an increasing trend in test dataset, therefore, it is not expected perform well as much as the regression model.

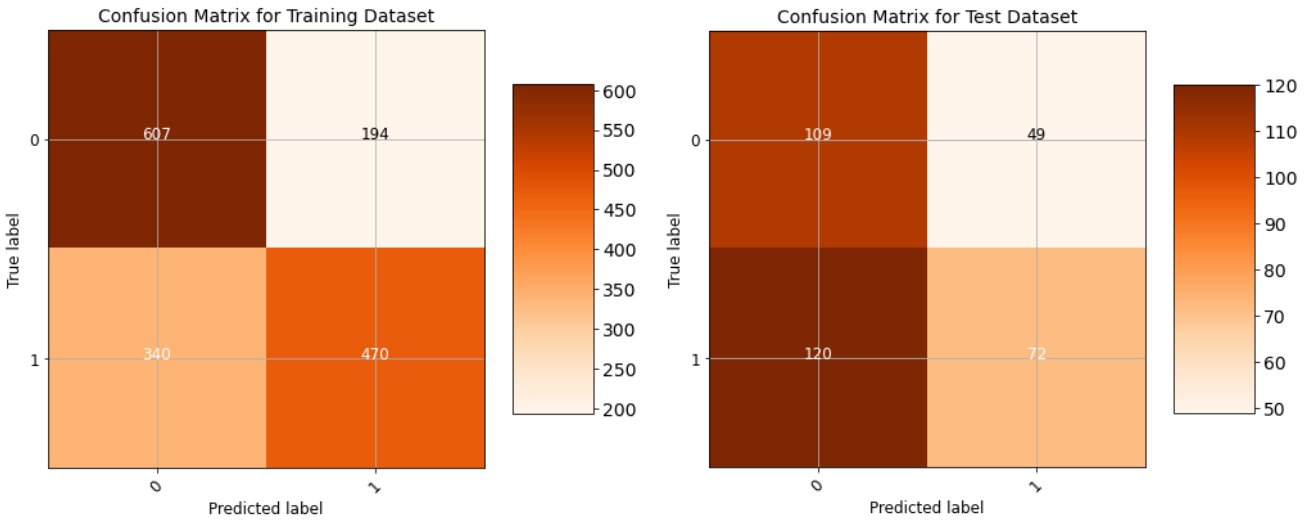


Chart: Confusion matrix for training and test datasets respectively

For classification problem, the model is developed with 16 features over 288 generated features. It is a good sign that selected features have similar feature importance in the model.

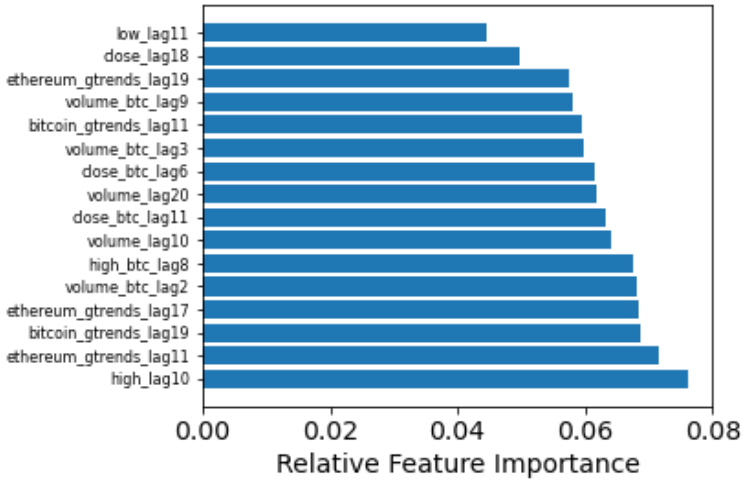


Chart: Relative feature importance of classification

Trading Algorithm

Overview

Besides model development, trading decisions play an important role to make profit in the market. For this reason, an optimized and well-designed trading algorithm turns a good model to successful buy and sell actions. The algorithm decides to buy and sell points for user according to predictions from the model. It runs daily basis and returns “BUY”, “SELL” or “DO NOTHING”. To do that, it uses buy and sell multipliers which are optimized by a grid search on training dataset in Optimization topic. On the table below, trading algorithm is explained as pseudo-code.

Algorithm: Trading Algorithm

- Inputs:
  - buy_multiplier: multiplier for the moment of buy decision
  - sell_multiplier: multiplier for the moment of sell decision
  - commission_rate: commission rate applied in the market
  - amount: number of Ethereum in the account, 0 at the start
  - budget: budget in the account, >0 at the start
- BEGIN with first row of data
- IF amount = 0 and today's prediction > yesterday's price * buy_multiplier*(1+ commission_rate):
  - Give "BUY" order
  - amount = budget / (yesterday's price*(1+ commission_rate))
  - buy_price = yesterday's price*(1+ commission_rate)
- ELIF amount> 0 and buy_price > prediction* (sell_multiplier^2):
  - Give "SELL" order
  - budget = amount* yesterday's price * (1-comission_rate)
  - sell_price = yesterday's price * (1-comission_rate)
  - amount = 0
- ELIF amount> 0 and yesterday's price* (1-comission_rate) > prediction* sell_multiplier:
  - Give "SELL" order
  - budget = amount* yesterday's price * (1-comission_rate)
  - sell_price = yesterday's price * (1-comission_rate)
  - amount = 0
- REPEAT for next row until the last row
- END

Optimization

After deciding trading algorithm, buy and sell multiplier need to be optimized in training dataset, then they can be applied through test dataset. The optimized values can be found with a grid search on multiplier parameters. For this reason, an objective function with constrain is not created to solve the optimization problem.

For both buy and sell multipliers a reasonable range is defined, and each combination is tried to reach highest budget level at the end of the date range for training dataset and applied to test dataset. There is commission rate parameter in some cases. In real world usual commission rate for each transaction is 0.25%. It effects optimized multipliers. For this reason, both with commission and without commission cases are drawn and will be explained under Evaluation topic.

Evaluation

Individual Models

For ARIMA, XGBoost regression and XGBoost classification models, potential earnings can be observed at below compared to a Buy&Hold strategy as a benchmark model. Budget_w_Commission line stands for budget with commission at 0.25% commission rate with optimized multipliers. Budget_wo_Commission line stands for budget without commission with optimized multipliers. Budget_then_Commission line stands for budget with 0.25% commission with optimized multipliers of Budget_wo_Commission line. For comparability, each model started with $1000 of budget. The charts at below show changes in the budget throughout test dataset.

ARIMA Model

Arima model can overperform Buy&Hold strategy in test data. However, it gives BUY order only two times. At the beginning of the data, it made these buy orders and holds Ethereum until and of the data. For this reason, it behaves like Buy&Hold strategy. Moreover, the difference between Budget with and without commission is limited because of small number of transactions.

XGBoost Classification Model

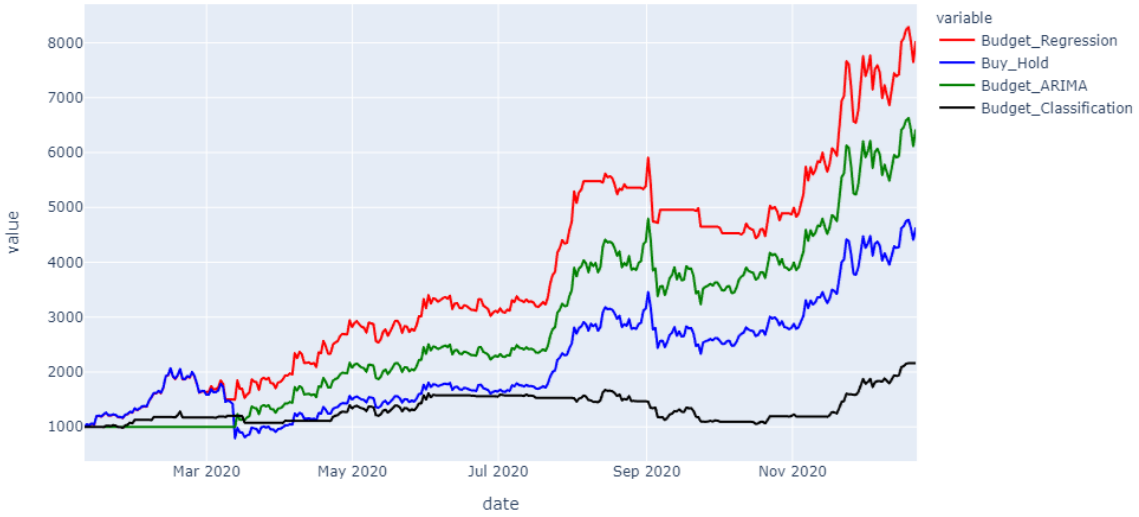
XGBoost classification model could not overperform Buy&Hold strategy. For this reason, it is not a preferable way to develop the model. As price change did not work well in ARIMA models, it also failed XGBoost Classification model.

XGBoost Regression Model

XGBoost Regression Model performs as the most profitable model among alternatives. While Buy&Hold strategy can increase the budget from $1000 to around $4000, XGBoost Regression Model can reach to almost $8000 at the end of the test dataset with and without positive commission rates. While number of transactions throughout the dataset, the gap between Budget_wo_Commission and Budget_then_Commission increases, which they have the same multipliers. Budget_w_Commission has its optimized buy and sell multiplier, which helps to overperform even Budget_wo_Commission.

Comparison

As explained in the previous chapter, XGBoost regression overperforms other strategies. The chart at below shows the strategies with 0.25% commission rate. XGBoost regression's success can also be identified at the sharp decrease in April 2020. Unless Buy&Hold strategy, it reacted and sold its assets to preserve potential negative outcomes. While Buy&Hold strategy would bring around $4000 with $1000 starting budget, XGBoost regressor reached around $8000 after commissions deducted.



Discussions

Does It Possible to Beat the Market?

Beating the market is difficult but possible. An obstacle of beating the market is transactional commission ratios, which create a gap between buy and sell prices in the market. It makes difficult to make a profit with short term buy and sell actions. Especially in old and well-established markets commission rates are too high that creates a barrier to efficiently and frequently trade. However, in cryptocurrency markets, these markup costs are noticeably lower compared to conventional ones. This was one of the main reasons to choose a cryptocurrency forecast project.

Furthermore, there are numerous examples that investors beat different financial markets. In the project, we also showed a short profit window by doubling that can be earned in Ethereum market. However, one of the struggles of beating market is keeping it sustainable. Even there are profit windows for the markets due to its inefficiency, it is difficult to say that one solution can always overperform the market in different market conditions. Examples which prove a profit with different techniques are just a small part of all attempts, which is called survivorship bias. Failed attempts do not prevail their efforts. For this reason, a belief that market can be beaten easily appears. However, when the number of failed attempts is compared to successful ones, it may be explained with randomness. As long as you try to beat the market, you will beat it for a short period.

Another discussion point, does it worth it? For example, an investor achieves 13% of gain in a market environment with 10% increase. The effort, stress and risks taken to overperform with 3% more than

market average may not exceed the opportunity cost of the investor. Moreover, taking such risks may cause an underperformance in different trading windows.

Further Research

Short Positions

In current setting, our trading algorithm optimizes max amount using buy decisions. In cryptocurrency markets, it is also possible to give sell orders-taking short positions-.
For example: In current setting,

- If our algorithm predicts an increase for the next day, it purchases.
- If it predicts a decrease and we are currently holding an asset, it sells the assets.
- If it predicts a decrease and we are not holding an asset, it waits.

Using short positions is like:

- If our algorithm predicts a decrease for the next day and we are currently holding an asset, it sells the assets and opens a sell position with the same amount.
- If our algorithm predicts a decrease for the next day and we are not currently holding an asset, it opens a sell position with the current amount.

The main idea is benefit from both increases and decreases. This is necessary because of the market shape changes. Our current setting performs well in bull markets. However, If the market direction is downward sloping -bear market- it can only hold the cash and wait until the price increases. But, if it can use those price decreases using short positions, investor can benefit from both bull and bear markets.

Different Coins

The model is based on ethereum price forecast, but there are around 8000 different coins in the market. If possible coins are examined which are more explainable by our features or use their data as inputs, we can create a better performing algorithm.

Market Manipulation

Crypto market has a high volatility due to a few reasons like small market cap, high variety of assets, relatively less historical information, and big investors.

Coin Whales

Whale is a crypto market term which refers to investors that hold large amount of bitcoin. Whales can manipulate market prices by using their relatively dominant asset amounts.
However, at crypto currency market, holder names are not publicly available, but all addresses and transactions are publicly available. So, it is possible to track the whale transactions to build a system that benefits from this information.

Social Media & News

The social media and news are two of the most important components of market volatility. Prices are very sensitive to any upcoming information. For example, Elon Musk tweeted about Doge Coin and it soared %20 in a few hours.
Another example is news about SEC (The Securities and Exchange Commission) is going to investigate XRP coin because of illegal coin sales and the same day XRP prices decreased around %50 in two days. This kind of speculative events should be added to the model. It is possible to do this in real time by web scraping and it can explain some of drastic changes in our time series.

Appendix

- Data Source for Ethereum and Bitcoin prices: https://www.kaggle.com/tencars/392-crypto-currency-pairs-at-minute-resolution
- Data Source for Google Trends search engine popularity data: https://trends.google.com/trends/