,,

Universiteit Leiden
The Netherlands

Bachelor Computer Science

Prediction and technical analysis of the Bitcoin crypto currency using machine learning

Stan van der Avoird

Supervisors:
Jan N. van Rijn Holger H. Hoos

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

13/08/2020

Abstract

The price of Bitcoin is volatile, which could bring great possibilities in terms of profit. This research focuses on predicting the price of Bitcoin short-term using machine learning to eventually use the model in a trading strategy to generate profit or loss. To produce a machine learning algorithm that predicts the price with the highest accuracy, we need to pre-process the dataset. The goal is to achieve better model scores by using technical indicators (TI) and optimizing 63 hyperparameters. With technical analysis, we gain more insight into the movement of the price. The addition of these technical indicators increase the accuracy of the models. The goal was to expose each technical indicator hyperparameter to our machine learning algorithm to optimize results. Various trading strategies have been tested along with various machine learning models. The results show that with the right strategy, using an optimized regression model and trading on hour time series interval, a return on investment (ROI) of +17.1% can be made on an investment over a period of 15 months.

Contents

1 Introduction

In this section, we will provide information about the existence of Bitcoin on the market together with its price trends. We explain what we currently know in terms of scientific researches, where we see opportunities in terms of new research, and our contribution to the current research field and how we will address these problems.

1.1 History of Bitcoin

The cryptocurrency market is a relatively young market, it currently exists for 11 years. Bitcoin is the oldest yet most popular cryptocurrency currently in existence. Bitcoin has been around since 2009, but was not traded on any exchange in 2009. The first recorded price was in 2010. Over the course of 10 years a lot has happened with the price of Bitcoin [9]. In 2010, the price started at $0.07 and peaked at $20,000 in 2017, see Figure 1. The price then dropped back below $4,000, clearly indicating that Bitcoin's price is volatile. Looking at the logarithmic scale, the price has generally been on the rise over the past 10 years. One of the main differences between the foreign exchange market and the cryptocurrency market is that the cryptocurrency market is open 24 hours a day, 7 days a week.

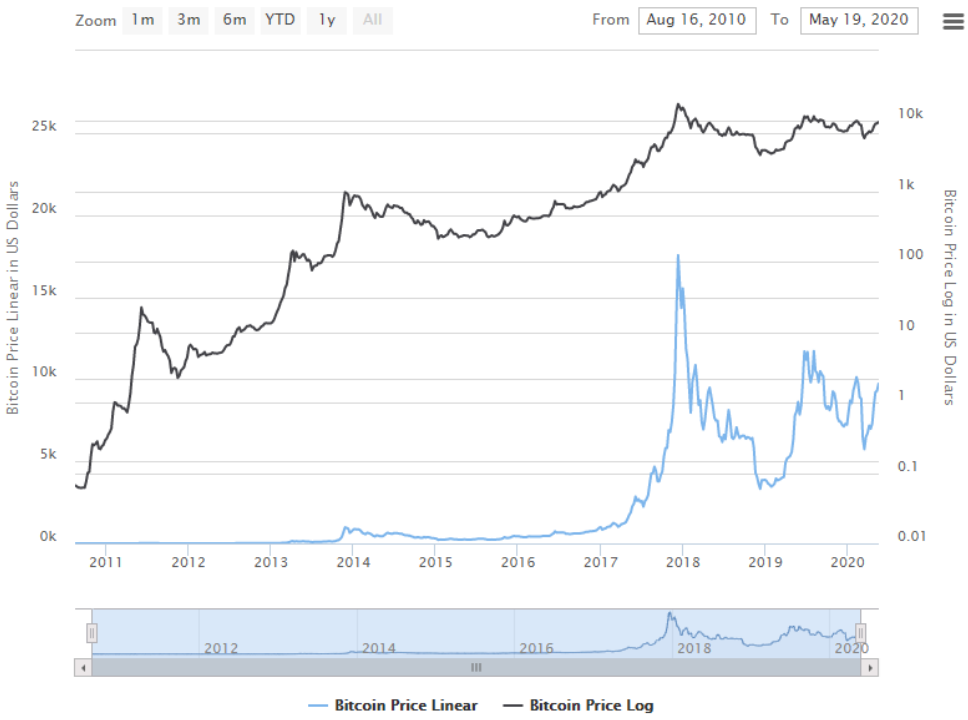

Figure 1: Bitcoin price movements 2010 until 2020. (Source: www.buybitcoinworldwide.com/price/)

1.2 What is known

It was early determined that different hyperparameter configurations tend to work best for different data sets [10]. Today it is also widely recognized that tuned hyperparameters improve from the default offered by regular machine learning libraries [13, 17, 19, 23]. According to previous research, the performance of stock price prediction can be improved using technical analysis [20, 12]. It is the principle that past trading activity and price changes can be valuable indicators to predict future price movements.

1.3 Research question

There are several ways to make machine learning predictions and to optimize these models. Our main question is: Does optimizing the hyperparameters of technical indicators with automated machine learning (AutoML) techniques improve our machine learning model scores? If this is the case, how do we optimize these hyperparameters?

1.4 Our hypothesis

Our hypothesis is that if we optimize the hyperparameters of our technical indicators, we get a more optimized and balanced model and therefore get better scores than previous studies. To do this, we first need to pre-process our default Bitcoin price dataset with a great number of technical indicators. Next, we have to expose all these technical indicator parameters as individual hyperparameters.

Furthermore, is it possible to implement a specific trading strategy to gain profit during a specific time span?

Banking institutions may have found high-performing machine learning models that can deliver a high return on investment. In order to democratize the access to these type of techniques, we do not want to limit the use of potentially better-performing machine learning models to banking institutions with the simple goal of just making money. We have data available on Bitcoin prices from February 2013 to 2020. By using automated machine learning techniques, we can optimize our technical indicators to increase the accuracy of our models. Our optimized models can be used for trading strategies to make a profit on our investment.

1.5 Our contribution

We want to expand the current research area of price predictions for the Bitcoin cryptocurrency by using machine learning. Our most important goals are the following:

- Is it possible to create a profitable trading strategy based on our classification or regression models?
- How can we improve machine learning scores for the prediction of Bitcoin's price?
- Important that these knowledge is openly available, not only available for banking institutes.

1.6 Thesis overview

This chapter contains the introduction; Section 2 includes the definitions; Section 3 discusses related work; Section 4 describes the experiments and their results; Section 6 concludes. This is the bachelor thesis for computer science at the University of Leiden, Prediction and technical analysis of the Bitcoin cryptocurrency using machine learning.

2 Definitions

2.1 Open, High, Low, Close, Volume

Starting with a default dataset which means that there was no pre-processing executed on this dataset and containing only the following 6 columns: Timestamp, Open, High, Low, Close and Volume as explained in Section 2. The Timestamp indicates the prices on that specific date and time. The Open, High, Low and Close columns are prices of Bitcoin during that specific interval where Open is the opening price at the beginning of that interval and Close the closing price. During this interval the price reaches a lowest and highest price which will be represented in the Low and High columns. The Volume column is a reflection of all transactions during a specific time period.

2.2 Taker and maker fee

Trading Bitcoin on any exchange entails fees. There are so called taker and maker fees. Maker orders are orders that provide liquidity, orders that take liquidity are taker orders. Liquidity means how quickly you can get your money. The total fees are generally calculated based on the trading volume from your last 30 days of trading. We have compared the most popular trading platforms and found that Kraken[1] is currently charging the lowest fees.
Taker fee. Starts at 0.25% and can be as low as 0.10%.
Maker fee. Starts at 0.16% and can be as low as 0.00%.
If we would execute our trading strategy on a 1 minute interval time series, we would make a lot of trades within these 30 days, boosting up our total trading volume and potentially lowering our fees to as low as 0.10% or even 0.00%. Since we want to trade on a 1 minute interval, our trades needs to be processed immediately. We will use the taker fee of 0.10% for backtesting purpose.

[1]Kraken. Maker and taker fees. `https://support.kraken.com/hc/en-us/articles/360000526126-What-are-Maker-and-Taker-fees-`

2.3 Technical indicators

During this research we make use of technical indicators to pre-process our default dataset as seen in Table 2. Technical indicators are mathematical values that are calculated by looking back a specific number of intervals in the dataset. The technical indicators are calculated based on the Open, High, Low, Close and sometimes the Volume columns. An example of a technical indicator is the Simple Moving Average. The calculation of this technical indicator is done in the following way:

$$\boxed{SMA_i^c = \frac{C_{i-n+1} + C_{i-n+2} + ... + C_i}{n}} \quad (1)$$

Where C is the closing price, the current interval is denoted with index i and n is the time window. Technical analysis can be performed on graphs to predict a trend or price based on past data. A great number of technical indicators are available, each having their own value and calculation.

2.4 Random forest

We have used Random Forest [2] as our machine learning algorithm. Random Forests is a machine learning algorithm for classification, regression and other tasks that works by constructing a multitude of decision trees during training and executing the class which is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

3 Related Work

3.1 Technical analysis

Predicting the price of Bitcoin using machine learning is not something new. A research [16] yielded an accuracy of 50-55% with Random Forest on a 10 minute time interval by using independent features related to the Bitcoin price. These features are different from technical indicators and are features like block size, cost per transaction, hash rate, market capitalization, miners revenue etc. They also tested on a daily interval and managed to get an accuracy of 98.7% with binomial logistic regression.

Another research [6] had quite similar results to the previous named research and found an accuracy of roughly 55% for classification models for 10 minutes interval data.

Neha Mangla (2019) studied different machine learning techniques to predict the price of Bitcoin. They focused on the prediction on a daily interval to get the highest accuracy as possible. Logistic regression, Support Vector Machine, Auto Regressive Integrated Moving Average (ARIMA) and Recurrent Neural Networks (RNN) have been compared in terms of performance. They made no use of pre-processing their dataset when applying machine learning. Among the four methods, ARIMA performs well for the next days predictions but performs poor for long term and yielded an accuracy of 53%.

A sentimental analysis with machine learning prediction for a specific set of cryptocurrencies was done in 2019 [24]. This study compared various machine learning techniques to research which technique would yield the best results in terms of classification accuracy. They compared three machine learning techniques; Multi-layer perceptrons (MLP), Support Vector Machines (SVM),

and Random Forest (RF) where SVM and RF outperformed MLP. They researched the following cryptocurrencies in order to conclude which machien learning technique performed the best: Bitcoin, Ethereum, Ripple, and Litecoin. SVM and RF most of the time outperformed MLP, but there was no significant performance difference found between SVM and RF for their specific analysis.

Schut (2019) [20] was making use of technical indicators to improve the performance of his models. Schut (2019) found that making use of technical indicators as features for machine learning models would increase the overall performance of the models for foreign exchange price predictions. We want to research whether we can apply this technique to improve our Bitcoin price prediction models.

3.2 Hyperparameter optimization

This work [4] introduced a robust new AutoML system based on scikit-learn. Known as auto-sklearn, this system improves on existing AutoML methods by automatically taking into account previous performance on similar data sets and building ensembles based on the models evaluated during optimization. Auto-sklearn [4, 5] is a popular and fast-growing way to optimize machine learning and hyperparameters. Auto-sklearn frees a machine learning user from algorithm selection and hyperparameter tuning. It takes advantage of recent advances in Bayesian optimization, meta-learning and ensemble construction.

Results from a study [1] show that randomly chosen trials are more efficient for hyperparameter optimization than trials on a grid as in Grid Search. Random Search is able to find models that are as good or better within a small fraction of the computation time than neural networks configured by pure grid search.

Despite the successes of Bayesian optimization using Gaussian process models, this optimization does not scale well to many hyperparameters. This study [21] managed to present a general approach for using neural networks for Bayesian optimization, staying as close to a truly Bayesian treatment as possible to obtain scalability.

4 Experimental setup

In this section we will explain what techniques were being used to perform our results.

4.1 Machine learning

For this specific research we have access to three time series intervals for the Bitcoin price; 1 day, 1 hour and 1 minute interval. We will test and review all time series intervals to check differences. We have chosen to run all tests with Random Forest. Random Forest consists of a large number of decision trees. Each decision tree outputs a class prediction, the class with the most votes becomes the model's prediction. The generalization error for Random Forests will converge to a limit as the number of trees in the forest becomes large [2]. Random Forest is a machine learning method for classification, regression and other tasks that work by constructing a multitude of decision trees during training and executing the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.
We will perform all tests on a train/test split of 80% training and 20% testing. This decision is

based on running various basic tests with our dataset and based on previous researches in this area [24, 16, 6]. Training on 80% and testing on 20% of the data means in most cases we are training on the years 2013-2018 and testing on 2019-2020.

For this particular research, we used three time series intervals for the Bitcoin price; 1 day, 1 hour and 1 minute interval. We will test and review all time series intervals to check for differences. We chose to run all tests with Random Forest on a train / test distribution of 80% training and 20% testing. This decision is based on conducting several basic tests with our dataset and based on previous research in this area. Training on 80% and testing on 20% of the data means that in most cases we train our model on the years 2013-2018 and testing on 2019-2020.

4.2 Target

The first thing we need to do is calculate the target in the default Bitcoin dataset. The target column contains the value for each row that the machine learning algorithm must predict. We consider two different targets in this research, a target for classification and a target for regression. The classification target is defined as either *true* or $false$. True if the *close* price from Bitcoin is going up in the next interval, and false when the *close* price is going down or stays the same in the next interval. The regression target is defined as the difference in the *close* price between two consecutive intervals. These two intervals may be consecutive. A small part of our dataset with their corresponding targets is shown in Table 1 below to demonstrate the target column. We have left out the Open, Low and High prices in this example since we use the Close price to generate our target.

Table 1: First 6 rows in our default Bitcoin dataset with their corresponding target for classification and for regression.

Timestamp	**Close**	**Target (Classification)**	**Target (Regression)**
2013-01-01 00:00:00	458.39	TRUE	27.73
2013-01-01 01:00:00	486.12	TRUE	6.73
2013-01-01 02:00:00	492.85	FALSE	-62.63
2013-01-01 03:00:00	430.22	FALSE	-17.74
2013-01-01 04:00:00	412.48	TRUE	31.74
2013-01-01 05:00:00	444.22	...	...

4.3 Pre-processing component

The second step is to pre-process our current default dataset with a useful set of technical indicators. Since we want to optimize our models and technical indicator hyperparameters later, the decision was made to create a pre-processing component that relies on the scikit-learn library. This component can be found on GitHub[2]. By creating this component, we can expose all technical indicator hyperparameters.
Our pre-processing component (later referred to as technical indicator component) generates 63

[2]Scikit-learn Bitcoin Pre-processing Component GitHub Repository. `https://github.com/Aurugorn/Scikit-learn-Bitcoin-component`

technical indicators. These technical indicators have their own hyperparameters that indicate how many intervals to look back to calculate their value. Since we want to optimize our hyperparameters later, we split our technical indicator component into two components, one with 63 hyperparameters and one with 9 hyperparameters.
No relation hyperparameters. This component is the main component, it generates all the technical indicators and considers every single technical indicator hyperparameter as its own hyperparameter. Thus using this component, our algorithm has to optimize 63 hyperparameters.
Relation hyperparameters. This component is a child of the main component, it generates all the technical indicators and links some hyperparameters based on their default technical indicator value. This child component reduces the amount of hyperparameters to 9.
The goal here is to test whether our relation component or no-relation component performs better.

To generate all 63 technical indicators in our technical indicator component, we used the TA library [3] for Python. We chose the most popular technical indicators plus a set technical indicators based their calculation speed. The speed of calculating the technical indicators is important because the technical indicators must be generated automatically every time we optimize our hyperparameters. Let us compile a list with the technical indicators we use in our technical indicator component, along with the linked technical indicator hyperparameters. Please note that some technical indicators might have more than one hyperparameter. These technical indicators with their corresponding amount of hyperparameters are the following: Stochastic Oscillator (2), Moving Average Convergence Divergence (3), Parabolic SAR (2), Parabolic SAR - Extended (8), Absolute Price Oscillator (2), Percentage Price Oscillator (2), Ultimate Oscillator (3), Chaikin A/D Oscillator (2). The total number of hyperparameters for technical indicators is 63 as seen in Table 2.

[3]Python wrapper for TA-LIB. https://github.com/mrjbq7/ta-lib

Table 2: 63 technical indicator hyperparameters used in our technical indicator component together with their related hyperparameters.

Technical indicator	Relation parameter	Technical indicator	Relation parameter
Simple Moving Average (Close)	Fast period (Default: 3)	Variance Close	var_t3 (Default: 5)
Simple Moving Average (High)		Variance Open	
Simple Moving Average (Low)		Triple Exponential Moving Average	
Simple Moving Average (High + Low)		Double Exponential Moving AVG	Dema_trema paramaters (Default: 30)
Simple Moving Average (High + Low + Close + Open)		Exponential Moving Average	
Stochastic		Kaufman Adaptive Moving AVG	
Momentum		Moving average	
Mean (Open, Close)		Triple Exponential Moving Average	
Chaikin A/D Oscillator		Triangular Moving Average	
Moving AVG Conv/Div	Long term (Default: 26)	Weighted Moving Average	
Absolute Price Oscillator		1-day ROC of Triple Smooth EMA	
Percentage Price Oscillator		Price Rate Of Change	Short term (Default: 12)
Stochastic	Mid term (Default: 14)	Moving AVG Conv/Div	
Williams %R		Absolute Price Oscillator	
MidPoint over period		Percentage Price Oscillator	
MidPoint Price over period		Parabolic SAR (Acceleration)	Zero (Default: 0)
Average Directional Movement Index		Parabolic SAR (Maximum)	
Average Directional Movement Index Rating		Parabolic SAR - Extended (Start value)	
Aroon		Parabolic SAR - Extended (Offset on reverse)	
Aroon Oscillator		Parabolic SAR - Extended (Acceleration init long)	
Chande Momentum Oscillator		Parabolic SAR - Extended (Acceleration long)	
Directional Movement Index		Parabolic SAR - Extended (Acceleration max long)	
Money Flow Index		Parabolic SAR - Extended (Acceleration init short)	
Minus Directional Indicator		Parabolic SAR - Extended (Acceleration short)	
Minus Directional Movement		Parabolic SAR - Extended (Acceleration max short)	
Plus Directional Indicator		Rate of change Percentage	ROC period (Default: 10)
Plus Directional Movement		Rate of change Ratio	
Relative Strength Index		Rate of change ratio 100 scale	
Ultimate Oscillator		Chaikin A/D Oscillator (Slow period)	
Bollinger Bands	bb_cci (Default: 20)		

4.4 Hyperparameter optimization

There are many ways to optimize machine learning models. We used Random Search to optimize our models. Random Search is widely used for hyperparameter optimization [1]. This algorithm randomly tests values in a specific range and reviews which set of hyperparameters achieves the best results. In contrast to Grid Search, not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specified distributions.

To optimize, we performed a large number of tests with four different configurations. The configurations will be shown in the results in Section 4. We will see the four different optimizing configurations in the table headers. We will explain all configurations in depth.
(1) Default: In this test no optimization was performed. All technical indicators and Random Forest hyperparameters were set on default.

(2) RF Only: In this test only the 5 Random Forest hyperparameters have been optimized. The hyperparameters of the technical indicators were set by default. The following 5 hyperparameters with their configuration space are shown below. We have narrowed the configuration search space by running several basic tests with a wider range of values. In the end, we chose same hyperparameters as auto sklearn: max depth, max features, min samples leaf, min samples split and the amount of estimators (Sklearn RandomForestClassifier Information, 2019).

- Max depth is the maximum depth of the tree. If this value was set to None then nodes are expanded until all leaves are pure or until all leaves contain less than the min sample split samples.
- Max features is the number of features to consider when looking for the best split.
- Min samples leaf is the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min samples leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
- Min samples split is the minimum number of samples required to split an internal node.
- The amount of estimators is the number of trees in the forest

Table 3: 5 Random Forest hyperparameter configuration search space.

Hyperparameter	**Range**
max depth	[80, 90, 100, 110]
max features	[2, 3]
min samples leaf	[3, 4, 5]
min samples split	[8, 10, 12]
amount of estimators	[100, 200, 300, 1000]

(3) 68 hyperparameters: In this test both 5 Random Forest hyperparameters plus 63 technical indicators hyperparameters were optimized. This is considered the no-relation component as described in Section 4.3 where we consider every single technical indicator hyperparameter as its own hyperparameter. The following search configuration space was being used. For all technical indicators we chose the range $loguniform(2, 100)$. With $loguniform$ we can make an approximate distribution to describe a variable that may take a very wide range. We chose the logarithmic distribution from 2 to 100 since all default technical indicator hyperparameters are within this range.

Table 4: 5 Random Forest plus 63 technical indicator no relations hyperparameter configuration search space.

Hyperparameter	**Range**	**Hyperparameter**	**Range**
max depth	[80, 90, 100, 110]	rocp period	[2, 100] (log scale)
max features	[2, 3]	rocr100 period	[2, 100] (log scale)
min samples leaf	[3, 4, 5]	rocr period	[2, 100] (log scale)
min samples split	[8, 10, 12]	rsi period	[2, 100] (log scale)
n estimators	[100, 200, 300, 1000]	sar acceleration	[2, 100] (log scale)
adosc fastperiod	[2, 100] (log scale)	sar maximum	[2, 100] (log scale)
adosc slowperiod	[2, 100] (log scale)	sarext accelerationinitlong	[2, 100] (log scale)
adx period	[2, 100] (log scale)	sarext accelerationinitshort	[2, 100] (log scale)
adxr period	[2, 100] (log scale)	sarext accelerationlong	[2, 100] (log scale)
apo fastperiod	[2, 100] (log scale)	sarext accelerationmaxlong	[2, 100] (log scale)
apo slowperiod	[2, 100] (log scale)	sarext accelerationmaxshort	[2, 100] (log scale)
aroon period	[2, 100] (log scale)	sarext accelerationshort	[2, 100] (log scale)
aroonosc period	[2, 100] (log scale)	sarext offsetonreverse	[2, 100] (log scale)
bb periods	[2, 100] (log scale)	sarext startvalue	[2, 100] (log scale)
cci periods	[2, 100] (log scale)	sma close timeperiod	[2, 100] (log scale)
cmo period	[2, 100] (log scale)	sma h l c o period	[2, 100] (log scale)
dema period	[2, 100] (log scale)	sma handl period	[2, 100] (log scale)
dx period	[2, 100] (log scale)	sma high period	[2, 100] (log scale)
ema period	[2, 100] (log scale)	sma low period	[2, 100] (log scale)
kama period	[2, 100] (log scale)	so d n	[2, 100] (log scale)
ma period	[2, 100] (log scale)	so n	[2, 100] (log scale)
macd period longterm	[2, 100] (log scale)	t3 period	[2, 100] (log scale)
macd period shortterm	[2, 100] (log scale)	tema period	[2, 100] (log scale)
macd period to signal	[2, 100] (log scale)	trima period	[2, 100] (log scale)
mean o c period	[2, 100] (log scale)	trix period	[2, 100] (log scale)
mfi period	[2, 100] (log scale)	ultosc period1	[2, 100] (log scale)
midpoint period	[2, 100] (log scale)	ultosc period2	[2, 100] (log scale)
midprice period	[2, 100] (log scale)	ultosc period3	[2, 100] (log scale)
minus di period	[2, 100] (log scale)	var close period	[2, 100] (log scale)
minus dm period	[2, 100] (log scale)	var open period	[2, 100] (log scale)
momentum period	[2, 100] (log scale)	wma period	[2, 100] (log scale)
plus di period	[2, 100] (log scale)	wr lookback period	[2, 100] (log scale)
plus dm period	[2, 100] (log scale)		
ppo fastperiod	[2, 100] (log scale)		
ppo slowperiod	[2, 100] (log scale)		
roc period	[2, 100] (log scale)		

(4) 14 hyperparameters: In this test both 5 Random Forest hyperparameters plus 9 technical indicators hyperparameters were optimized. This is considered the relation component as described in Section 4.3 where we link some technical indicator hyperparameters to reduce the total amount of hyperparameters.

Table 5: 5 Random Forest plus 9 technical indicator with relations hyperparameter configuration search space.

Hyperparameter	**Range**
max depth	[80, 90, 100, 110]
max features	[2, 3]
min samples leaf	[3, 4, 5]
min samples split	[8, 10, 12]
amount of estimators	[100, 200, 300, 1000]
fastperiod	[2, 100] (log scale)
longterm	[2, 100] (log scale)
midterm	[2, 100] (log scale)
shortterm	[2, 100] (log scale)
bb cci	[2, 100] (log scale)
var t3	[2, 100] (log scale)
dema trema	[2, 100] (log scale)
zero	[2, 100] (log scale)
rocperiod	[2, 100] (log scale)

4.5 Classification experiment

We will explain our classification setup in four steps as demonstrated in Figure 2. The first step is to read our Bitcoin price dataset, which contains data of the price of Bitcoin from 2013 to 2020.

Next, the full dataset will be splitted in a training and testing dataset. The training dataset will be used to train our models and will be used for Random Search later. We will test on our test dataset which will return a specific accuracy. The training data consists of 80% of the data, which is 2013 to 2018, the test set consist of 20% of the data, which is 2019 to 2020.

In step three we run our Random Search for a numerous amount of times. Before each search iteration, a pipeline is executed. The first step in the pipeline is our technical indicator component, which will generate all 63 technical indicators and extend this to our Bitcoin price dataset as shown in step one. The technical indicators will be generated based on a specific hyperparameter configuration as seen in Table 4 or Table 5. Next, we impute missing values into our dataset and then forward our dataset to Random Forest to perform the machine learning. Random Search is set to execute a 3 k-fold cross validation, which means that the training dataset will be splitted into three new training parts and we will try out these hyperparameters on this new splits.

We then report the best hyperparameter configuration to our test data which will return a specific accuracy.

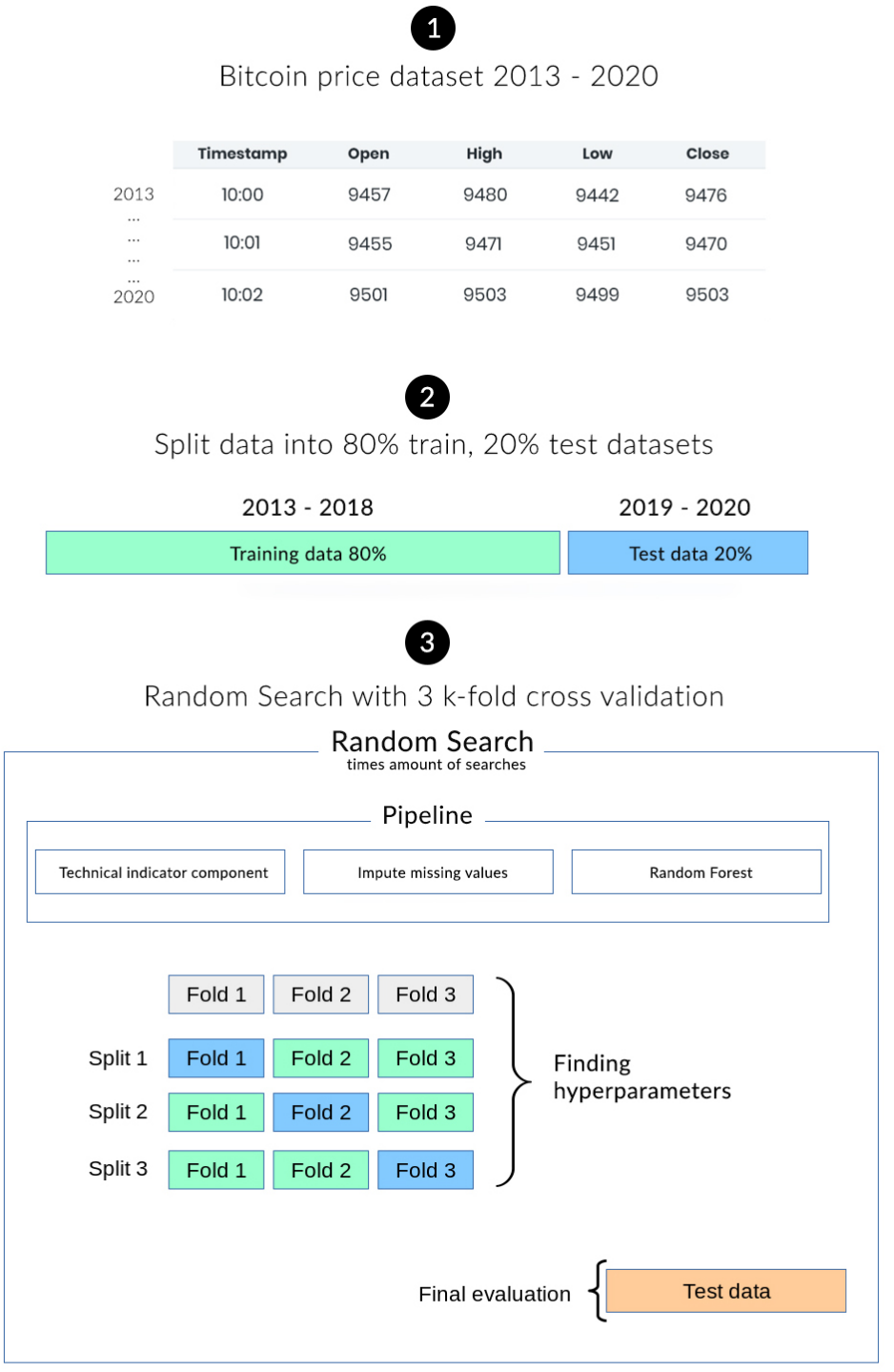

Figure 2: Backtesting 1 minute interval classification versus baseline, taker fee included

4.6 Backtesting

When the algorithms found good performing models we want to verify these models in terms of real life profit and losses. With backtesting we can examine how much profit or loss we would have make if we were to use our models in the real world. We have created different trading strategies that gave us different results. The most important thing to mention is the maker and taker fees. Trading Bitcoin is not free since exchanges charges fees. As defined in Section 2 we will use the taker fee for backtesting purpose and the taker fee can be as low as 0.10%. With real life backtesting we have to take the fee into account. With classification we only predict whether the price goes up or down in the next interval, we do not know how much it will increase or decrease. This may cause in losing your investment due to the fees. We can resolve this problem by training the classification dataset based on the price rise including the taker fee. This way the *target* column would return *true* when the price of Bitcoin in the next interval is higher than the current interval plus the taker fee already included, and else $false$. For example: if the price of Bitcoin in the current interval is \$10.000 and the price of Bitcoin in the next interval is \$10.005 we would normaly output *true* since the price is \$5 higher, but taking the taker fee into account: $\$10.005 * (1 - (0.10/100)) \approx \$9.995$ we will output $false$ since with the taker fee we would lose money on this trade. Since the taker fee is dependent on the exchange you trade on and the total trading volume in the past 30 days we will test both classification models trained with and without taker fee. Table 6 demonstrates how the target column is calculated in both classification methods. The *Change* column indicates the percentage change between the closing price in the next interval versus the closing price in the current interval.

Table 6: First 6 rows in our default Bitcoin dataset with their corresponding target for classification with and without taking the taker fee of 0.10% into account.

Timestamp	**Close**	**Change**	**Target (Without fee)**	**Target (With fee** 0.10%)
2013-01-01 00:00:00	458.39	+6.05%	TRUE	TRUE
2013-01-01 01:00:00	486.12	+0.03%	TRUE	FALSE
2013-01-01 02:00:00	486.25	−2.06%	FALSE	FALSE
2013-01-01 03:00:00	476.22	+0.24%	TRUE	TRUE
2013-01-01 04:00:00	477.34	+0.09%	TRUE	FALSE
2013-01-01 05:00:00	477.79	...	...	...

With regression we predict how much a price will increase or decrease, this way we can determine to only invest if our model predicts that the price will increase more than the taker fee, resulting in not losing your investment due to fee costs.
For backtesting we picked the best performing models for classification and regressions.

5 Results

In the next section we show the results of our different models at different time series intervals compared to the baselines.

5.1 Baselines

Because we have different time series intervals, we also have multiple baselines for each interval. We choose the best performing baselines to challenge ourselves to generate models that will perform better than the best performing baseline. There are several baselines available shown in Table 7.

Table 7: All used baselines per machine learning type classification or regression.

Machine Learning	**Baseline**	**Explanation**
Classification	No-change	Predicts same as last interval
	Stratified	Predicts by respecting the training set's class distribution.
	Most-frequent	Predicts the most frequent label in the training set
	Prior	Predicts the class that maximizes the class prior
Regression	No-change	Predicts same as last interval
	Mean	Predicts the mean of the training set
	Median	Predicts the median of the training set

We will evaluate all results compared to the baselines per time series interval. First, we look at the classification scores and use the accuracy of the models to compare. A total of 360 tests were carried out, namely 3 (classification, classification for backtesting, regression) factor 3 (day interval, hour interval, minute interval) factor 10 (various random states) factor 4 (various optimizing configurations) totalling 360 tests.

5.2 Classification

We first trained our classification models without any maker or taker fee. That means that if the price increases in the next interval, we must predict $true$ else $false$. All classification accuracy scores for different tests are located in Table 8. The very first column tells us the time series interval: day, hour and minute. The last column contains the best performing baseline accuracy scores. The configurations of the table header are explained in detail in Section 4.4.

Table 8: Classification accuracy score results

Interval	**Baseline**	**(1) Default hyperparameters**	**(2) RF Only**	**(3) 68 hyperparameters**	**(4) 14 hyperparameters**
Day	53.91%	52.74% ±1.24	55.22% ±0.80	55.02% ±0.79	55.29% ±1.40
Hour	50.70%	52.64% ±0.42	52.92% ±0.26	54.54% ±0.43	53.35% ±0.55
Minute	57.79%	58.75% ±0.06	59.69% ±0.29	60.23% ±0.14	59.54% ±0.34

Day interval. Let us start with the daily interval. We have train/test 80%/20% split from 2013 to 2020 February. As seen in Table 8 the stratified baseline for daily interval is 53.91%. Configuration (1) with default paramaters was worse than our baseline, but after optimizing our

models in configurations (2), (3) and (4) we get consistently an accuracy score better than the baseline (55.22% > 53.91%). The first question we need to ask ourselves is whether optimizing the technical indicators helped increase the accuracy. There is no significant difference between optimizing only the Random Forest hyperparameters versus optimizing technical indicator hyperparameters on a daily interval. The first row of Table 8 (daily interval) is elaborated in a boxplot shown in Figure 3. Here you can see that the stratified baseline was the best performing baseline, hence this value is shown in results Table 8. Based on 10 random states iterations the 14 hyperparamater model (relation model) contains the most outliers. Increasing the number of tests for this particular test allows us to find a more specific mean, but due to time and resources limited, we decided to stick with ten random states. Based on these tests we get consistently better accuracy scores by optimizing hyperparameters. There is no increase in performance when optimizing the technical indicator hyperparameters, thus optimizing Random Forest alone (2) was enough to get the highest possible accuracy score.

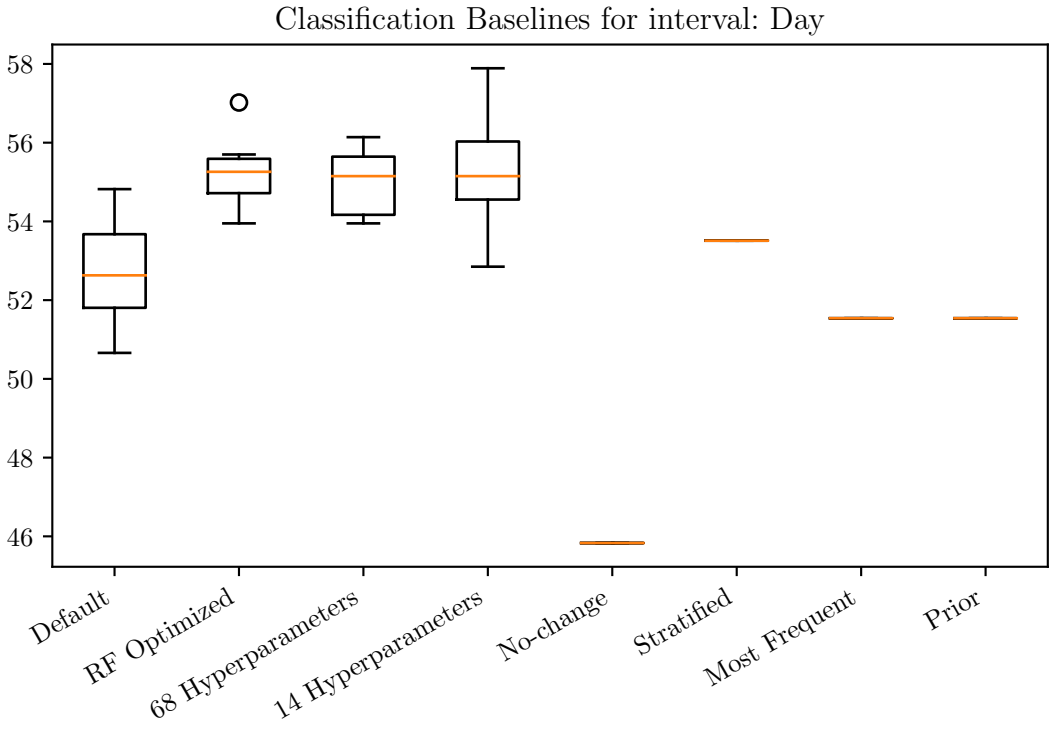

Figure 3: Classification accuracy scores for day interval compared to all baselines

Hour & Minute interval. We have train/test 80%/20% split from 2013 to 2020 February on hour interval, for minute we did this from 2019 January to 2020 February to reduce intensive computation time for training and testing the entire dataset. Both of our optimized models for hour and minute intervals outperformed the baselines. We can see that optimizing all 68 hyperparameters separately gave us the best accuracy score for hour and minute intervals respectively 54.54% and 60.23% as seen in Table 8.

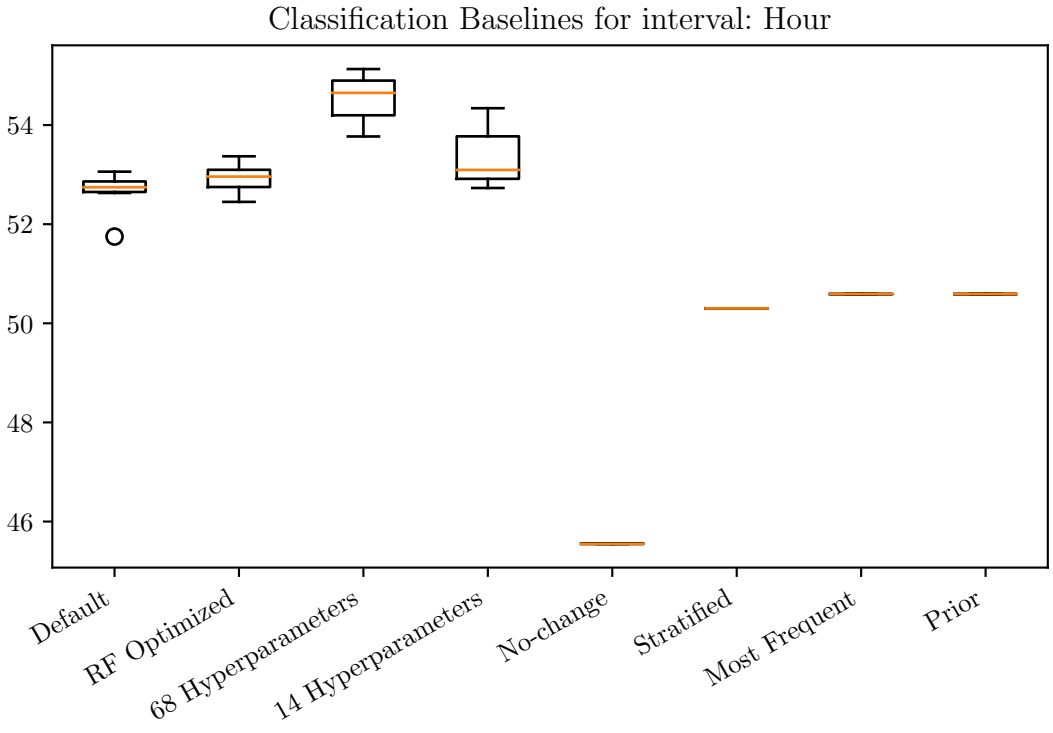

Figure 4: Classification accuracy scores for hour interval compared to all baselines

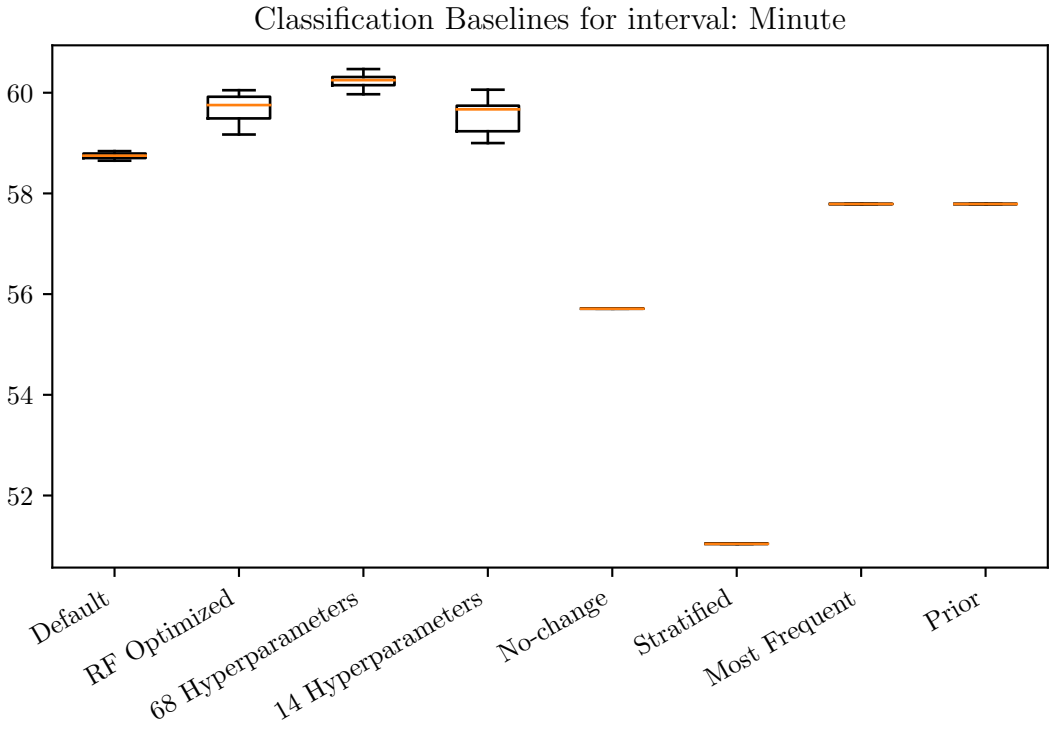

Figure 5: Classification accuracy scores for minute interval compared to all baselines

5.3 Classification for backtesting

We have also trained our classification model with taking into account the taker fee of 0.10%. This means we have to predict *true* if the price in the next interval minus the fee costs is higher than the current price as explained in Section 4.6. Since our target column is now different, we need to recalculate all baselines and compare the results with these new baselines. The results of these tests are shown in Table 10. The most noticeable difference is the minute interval. By including the taker fee in our training model, the accuracy of the most-frequent baseline has increased to 95% just like our own optimized models. Predicting the most frequent classification label in the dataset on a minute interval means in our case predicting $false$, since it is less common for Bitcoin's closing price to increase more than 0.10% during a minute interval. Hereby we eliminate *true* values in the target column as described in Section 5.5. Executing random forest machine learning on our models yielded the same accuracy score as the most-frequent baseline, meaning that our models likely predicted the most common classification label to get the highest possible accuracy score.

The day and hour intervals stay close to the same as results in Section 5.2. We can explain this by assuming that the price would usually increase by at least the amount of the 0.10% taker fee on the day and hour intervals.

Table 10: Classification with 0.10% taker fee accuracy score results

Interval	Baseline	(1) Default hyperparameters	(2) RF Only	(3) 68 hyperparameters	(4) 14 hyperparameters
Day	52.85%	53.39% ±1.41	55.67% ±0.80	54.60% ±0.93	55.37% ±1.24
Hour	60.72%	56.94% ±0.69	56.19% ±0.70	58.09% ±0.62	57.45% ±1.49
Minute	95.28%	95.24% ±0.01	95.27% ±0.01	95.28% ±0.01	95.27% ±0.00

Day & Hour interval. The accuracy score on a day interval stays at around 55% which is quite the same as training on classification without any fees included in Section 5.2.
The baseline for hour interval increased from 50.70% to 60.72%. Our optimized model scores increased by around 4%, so our own models are now worse than the most-frequent baseline.

5.4 Regression

We have trained our regression models with the Random Forest regressor on a 80% train, 20% test split. Regression has not not the same complication as classification. Our model is trained to predict the difference in price between the current interval and the next interval. There are several regression metrics that we can use to compare out models to the baselines. We will pick the mean absolute error (MAE), it is the simplest metric for regression. To calculate the MAE we will calculate the residual for every data point and taking the absolute value of each so that negative and positive residuals do not cancel out. Then we take the average of all these residuals (Pascual, 2020). The goal here is to find a model with a MAE that is lower than the best performing baseline. The lower the MAE the better our model predicts. We ran the same tests for regression as mentioned in Section 5.2. All results can be found in Table 12.

Table 12: Regression mean absolute error (MAE) score results

Interval	Baseline	(1) Default hyperparameters	(2) RF Only	(3) 68 hyperparameters	(4) 14 hyperparameters
Day	194	264.38 ±10.15	205.82 ±3.09	204.05 ±2.38	202.87 ±2.56
Hour	34	58.90 ±1.37	42.32 ±0.12	42.46 ±0.24	42.72 ±0.45
Minute	2.63	3.04 ±0.03	3.09 ±0.02	3.07 ±0.07	3.00 ±0.06

Day & Hour interval. We have train/test 80%/20% split on 2013 to 2020 February. Let us first look at the day interval for regression. We used three baselines for regression namely no-change, mean and median baselines as described in Table 7. The best performing baselines here are the mean and median so we picked the mean baseline with a MAE score of 194. Interestingly we did not find any (optimized) model that was performing better than this baseline. Training our regression model with default Random Forest and technical indicator hyperparameters yielded a MAE of 264.38 (1). We can clearly see that optimizing the hyperparameters of Random Forest only (2) increased our score to a MAE of 205.82. Further optimizing the technical indicator hyperparameters gave us a slightly better MAE of 204.05 (3). By linking some technical indicator hyperparameters in configuration (4) our MAE decreased to 202.87. Based on 10 random state test runs we can say that optimizing the hyperparameters of Random Forest plus linked technical indicator hyperparameters gave us the best MAE score as seen in Figure 6. However, the baseline was still better. An explanation for this could be that our regression model sometimes predicts outliers, while the mean baseline predicts the mean of all target values, thus the mean absolute error is lower with the baseline.
Results for the hourly interval could be interpreted the same as for day interval, the only difference is that there was no significant difference between optimizing only 5 Random Forest hyperparameters (2) versus optimizing hyperparameters of technical indicators (3), (4).

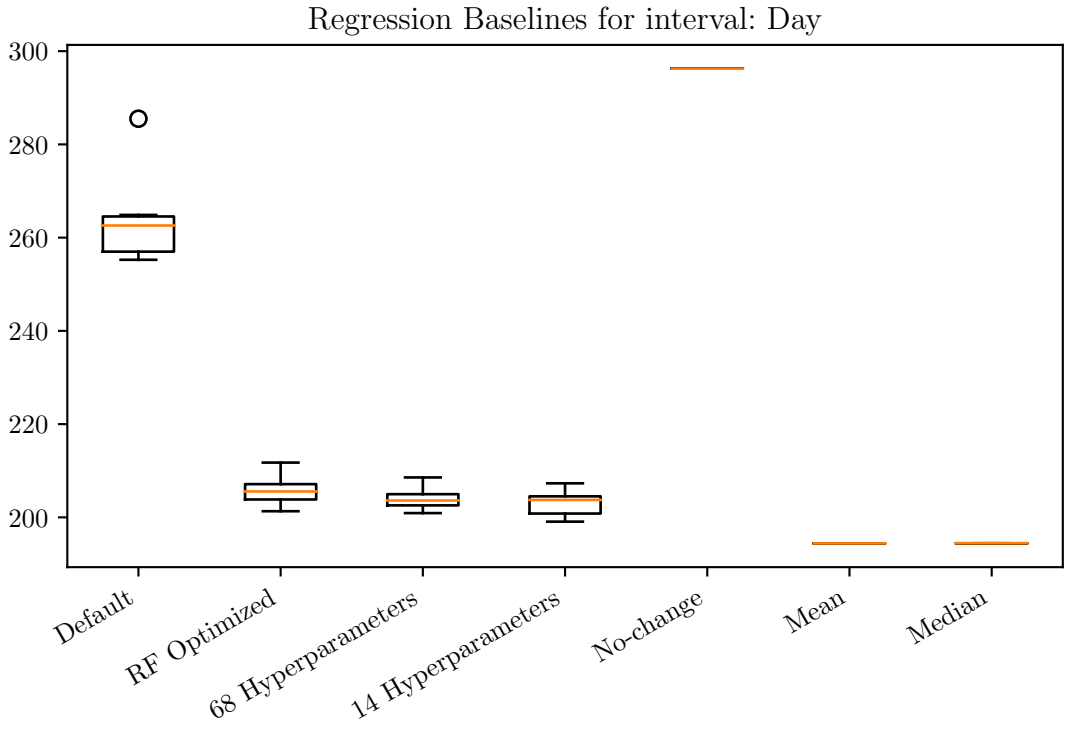

Figure 6: Regression mean absolute error (MAE) for day interval compared to all baselines

Minute interval. We have train/test 80%/20% split on 2019 January to 2020 February. The mean baseline still outperforms our optimized model MAE scores. While optimizing hyperparameters

for the day and hour intervals gave a better MAE score in general, the default settings for the minute interval gave us the same results as for the optimization, as seen in Table 12. Unfortunately we are again not better than the best performing baselines.

5.5 Backtesting Classification

Because we want to verify whether we would make profit in a realistic selling it is important to include the taker fee into our calculations. As already described in Section 4.6 we have also trained our classification models on the target values which includes the taker fee already.

Trading on a 1 minute interval. By taking this fee into account the amount of *true* values in our *target* column reduces from 236.733 to 35.924 *true* values for our 1 minute interval classification dataset. This amount of *true* values means that there are less intervals where the price of Bitcoin raises with at least 0.10%. We create a scenario where we invest $10.000 at November 2019 and we buy whenever our model predicts *true* and sell whenever our model predicts $false$. We have tested our classification minute interval model on November 2019 to February 2020 and will use this time span to trade on. In this time span there were 111.671 intervals to trade on. With our best optimized model seen in Table 10 no-relation pre-processing component (3) on a minute interval with an 95.28% accuracy, we used 121 intervals to buy Bitcoin since our model predicts the price is going up in the next interval. We have compared the backtesting results with backtesting the no-change baseline which always predicts the previous interval target. With our own model we have managed to generate a loss of -5% as seen in Figure 7. The no-change baseline resulted in losing nearly our whole investment (-98%). The x-axis represents the date, the y-axis is the price in dollars. We have four lines in our graph. The dashed line is the break-even line, which is a horizontal line at $10.000, your investment. The dotted line is the price of Bitcoin. The colored lines represents our portfolio investment for that specific classification model, whereas the blue line is our own optimized model and the yellow line is the baseline (holding strategy). We can safely conclude that trading on a 1 minute interval with our classification model that was trained to predict *true* if the next interval was higher than the current interval plus the taker fee will not give us any profit. We have also tried to backtest on our classification model 1 minute interval without including the taker fee, this resulted in losing your whole investment within the first month due to reasons mentioned in Section 5.5.

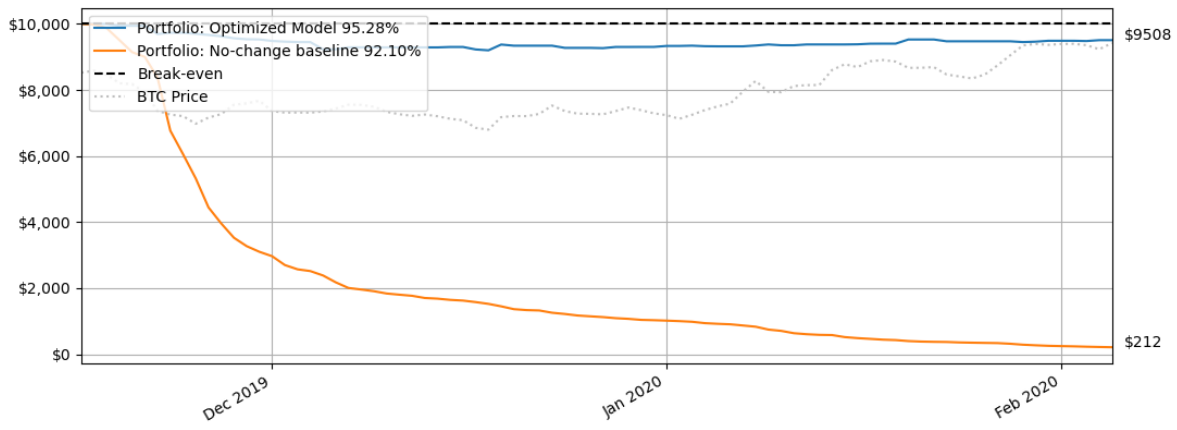

Figure 7: Backtesting 1 minute interval classification versus baseline, taker fee included

Trading on a 1 hour interval. We have also tried trading on a hourly interval from December 2018 to April 2020. There were 12.236 intervals to trade on during this time span, our best performing model bought 1.403 times but without any profit. During this time span our investment dropped from \$10.000 to \$5.477 losing roughly -46%.

Trading on a 1 day interval. We have found some interesting backtesting results by trading with our classification models on a 1 day interval and managed to make $+25.4\%$ in profit by trading from January 2019 to end of March 2020. We have used the classification model that only optimized Random Forest with an accuracy of 55.67% as seen in Table 10. Our best performing stratified baseline as seen in Table 10 of 52.85% accuracy also generated $+23.4\%$ in profit. By looking at the graph in Figure 8 you see that the stratified baseline (yellow line) heavily correlates with the Bitcoin price (dotted line), this means that when the price of Bitcoin goes up, our investment goes up and vice versa. It looks like our own model knows when to stop buying Bitcoin, since it trades our Bitcoin back to fiat money from July 2019 to March 2020 when Bitcoin's price decreased.

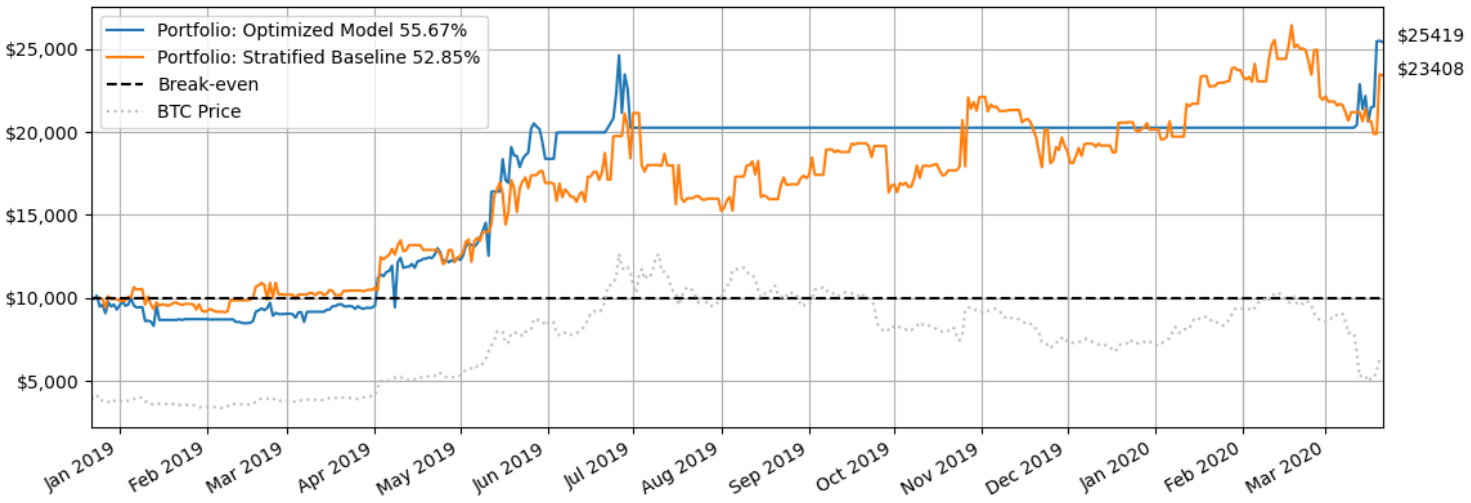

Figure 8: Backtesting 1 day interval classification versus baseline, taker fee included

Now the next obvious question would be: is this profit reproducible? We have decided to divide this time span of January 2019 to April 2020 into chunks of four months and run our backtesting strategy on each of these time periods. These time spans with their corresponding profit/loss can be found in Table 14. Both our model as the baseline model made a steady ROI during these periods. By taking the average over these four periods, we see that our own model wins with an average profit of +10.4% versus the stratified baseline with an average profit of +8.7%.

Table 14: Average ROI results for backtesting on day interval with classification.

Period	**ROI our model**	**ROI baseline**
January 2019 $\sim$ April 2019	+12.1%	+13.2%
May 2019 $\sim$ August 2019	+17.1%	+13.4%
August 2019 $\sim$ December 2019	+0.0%	+5.0%
December 2019 $\sim$ April 2020	+12.4%	+11.9%
Average ROI	+10.4%	+8.7%

According to these backtesting results, we can conclude that we are making a profit on a day trading interval and we are losing money by trading in hours and minutes for our classification model.

5.6 Backtesting regression

We will discuss the results of backtesting our regression models. We have performed backtesting on all available time series intervals and we have used different strategies to optimize our profit which

we will explain further on. We start with day interval then reviewing hour and at last minute.

Trading on a 1 day interval. We have again compared our best performing optimized model for regression to the best performing regression baseline. According to Table 12 our best performing optimized model is (4), our relation pre-processing component. This model has a MAE score of approximately 202 versus the mean baseline which has a MAE score of 194. Theoretically the mean baseline should outperform our own optimized model with backtesting since the MAE score is lower. Applying this strategy over 456 iterations, our model bought 19 times generating a steady profit of +20.4% as seen in Figure 9. Steady because we did not make any losses during this time span. The mean baseline ended with a profit of +15.4%. The mean baseline heavily correlates with the price of Bitcoin and thus more risky.

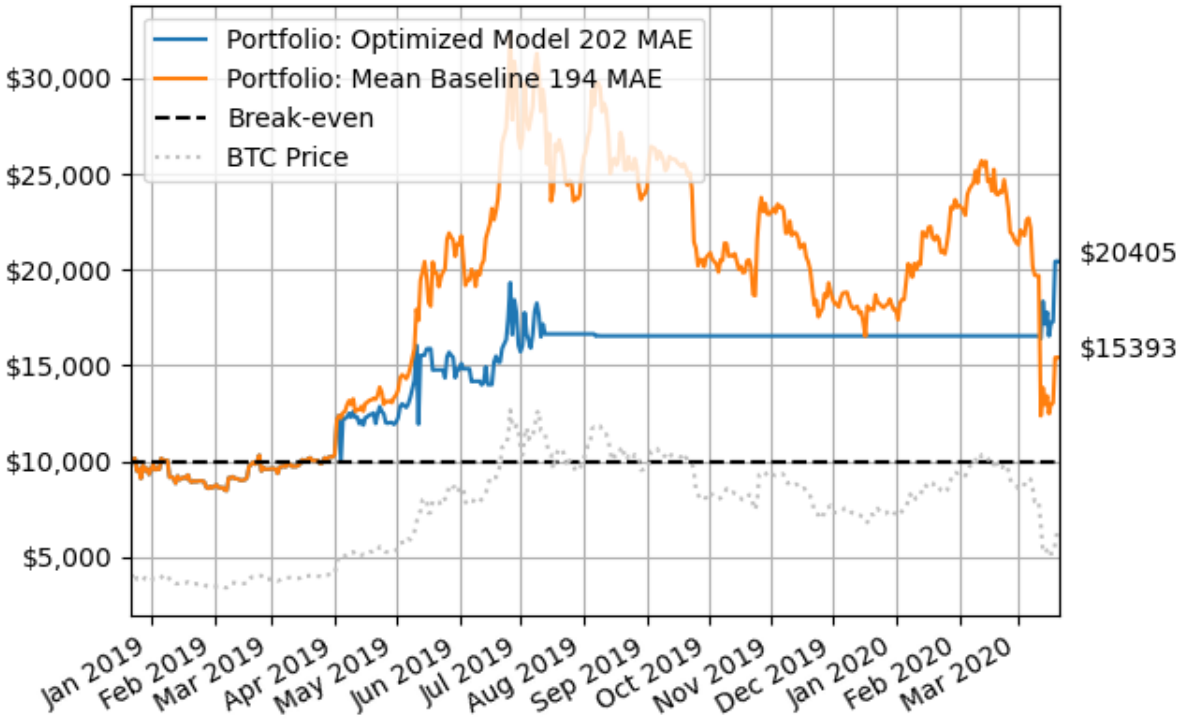

Figure 9: Backtesting 1 day interval regression versus baseline

We again split up this period into four equal time periods and compare the average ROI of our model and the baseline model. We can see these results in Table 15. The average profit of our own model is +9.6% and the average loss of the mean baseline is −5.9%.

Table 15: Average ROI results for backtesting on day interval with regression.

Period	**ROI our model**	**ROI baseline**
January 2019 $\sim$ April 2019	+12.1%	+12.9%
May 2019 $\sim$ August 2019	+14.0%	+22.6%
August 2019 $\sim$ December 2019	+0.0%	−38.0%
December 2019 $\sim$ April 2020	+12.4%	−21.0%
Average ROI	+9.6%	−5.9%

Trading on a 1 hour interval. We now do the same for 1 hour interval in Figure 10. The smaller the interval, the less the Bitcoin price moves within this interval. These backtesting results require more explanation. The blue and yellow lines in Figure 10 are using the same strategy as explained for regression 1 minute interval backtesting; buy and sell when model predicts so. This resulted in a loss for both the mean baseline and our optimized model. We introduced a new parameter, the *strictness* parameter. This parameter is a value ranged from $1 - 100$. The value that is predicted by our model (the amount the Bitcoin price goes up or down) will be divided by our *strictness* parameter. Let us explain this with a small example. If our model predicts that the price of Bitcoin will raise by +0.8% and our taker fee is 0.1%, then we would initially buy Bitcoin since $0.8 > 0.1$. Doing this on a hour interval backtesting resulted in overall loss of your investment. If we now introduce the *strictness* parameter with value 10, we will divide the prediction 0.8 by our *strictness* parameter: $0.8/10 = 0.08$. We now check again if our prediction is higher than our taker fee: $0.08 > 0.1$?, it is not higher so we will not buy Bitcoin. We tested all strictness values ranging from $1 - 100$ to see which value would be the most profitable for this time span and ended up with a *strictness* of value 8. We have tested this new strategy on our time span of January 2020 to April 2020 and managed to never lose our investment and ended up with +17.1% in profit as seen in Figure 10.
Note that this $strictness = 8$ parameter was tuned on the test dataset on this specific trading time span. It is possible that this value is not optimal when trading on different time spans.

The average ROI on four different time periods of our *strictness* strategy is +12.7%, the average profit of our standard strategy is −2.6%. By introducing the *strictness* parameter we could increase our overall ROI for hour interval with our regression model.

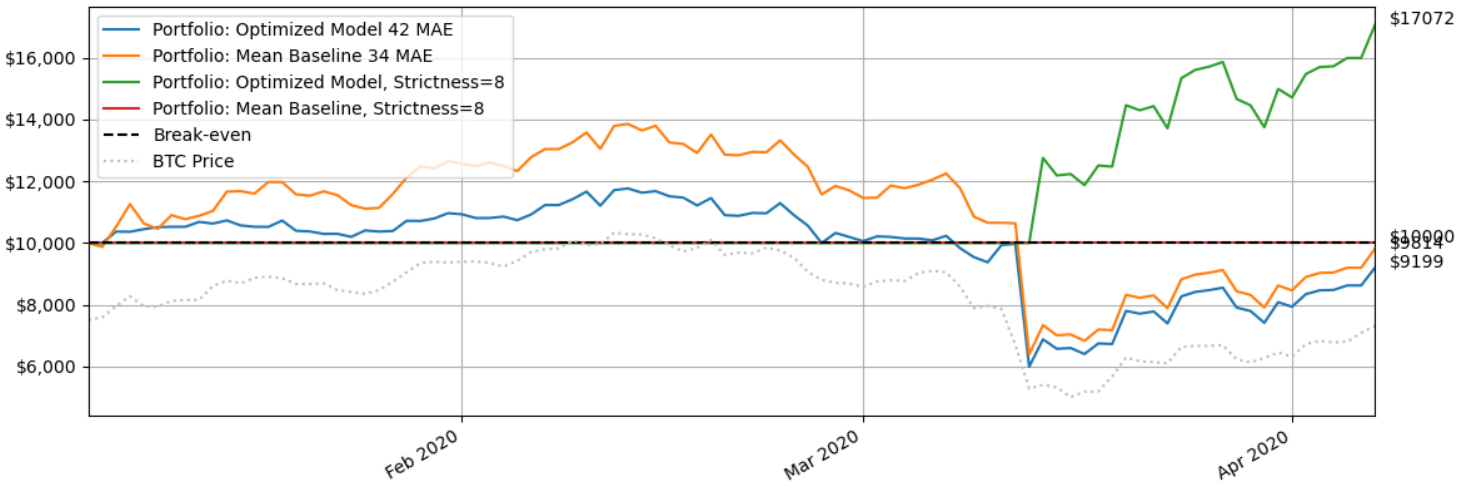

Figure 10: Backtesting 1 hour interval regression versus baseline

Trading on a 1 minute interval. As mentioned before, the smaller the time interval the smaller the change in price of Bitcoin between these intervals. We have backtested our regression model on a minute interval from November 2019 to February 2020 and found no way to generate profit.

6 Conclusions

Our goal was to find classification and regression models that would outperform our best performing baselines. We made use of technical indicators to pre-process our dataset to feed our machine learning algorithms with more data to train on based on the research of Schut (2019) [20]. We performed all tests on three different time series intervals; daily, hourly and minute interval. For classification we trained our model with and without including the taker fee for backtesting purposes.

We have managed to beat the best performing baselines for all time series intervals for our classification models without including the taker fee as described in Section 5.2. On all time series intervals (day, hour, minute) our no-relation pre-processing component would give us the highest accuracy of respectively 55.02%, 54.54%, 60.23% (Table 8). Linking technical indicator hyperparameters would not give us better performance.

The performance of our optimized models versus the best performing baseline dropped when taking the taker fee into account as explained in Section 5.3. Only our relation pre-processing component model (4) in Table 10 for daily interval performed better than the stratified baseline, whereas the hour and minute interval models was not better than this baseline. Again, linking technical indicator hyperparameters would not give us better performance.

Using these classification models for backtesting on a hour and minute interval would result in losing money over time as described in Section 5.5. Trading on a day interval with our best performing model would generate +25.4% in profit over a time span of 15 months with an average profit of

+10.4% by splitting up this time span into four equal parts.

Interestingly the optimized regression models were not better than the mean baseline on all time series intervals (Table 12). An explanation for this could be that our models sometimes predict outliers, while the mean baseline always predicts the mean of all target values, thus the mean absolute error is lower with the baseline.
Optimizing Random Forest hyperparameters alone was enough to get the best performance, which means optimizing technical indicator hyperparameters is not giving us better scores, Table 12. In terms of backtesting our regression models managed to beat the mean baseline in getting a higher average ROI for day and hour interval as seen in Section 5.6. On a day interval trading we made +20.4% in profit over a time span of 15 months with an average profit of +9.6% by splitting up this time span into four equal parts. For hour interval this was respectively +17.1% with an average of a +12.7% ROI.

Does optimizing technical indicator hyperparameters improve performance over just optimizing Random Forest? According to our results, in some cases it does and it depends on the time series interval. For classification hourly interval optimizing technical indicator hyperparameters increased our accuracy score from 52.92% to 54.54%. For minute interval the accuracy score increased slightly from 59.69% to 60.23%. We saw no improvement in performance for our regression models.

7 Further Research

Research back in 2015 [14] found that there was a correlation between positive tweets on Twitter and the increase of the Bitcoin price. They also found that there was a zero-lag correlation with the amount of Google searches. These tests were done on a time span of 60 days. As written in the Experiments section, we also spend some time looking at correlations between the Bitcoin price and social media data like Google searches and the growth change of subreddit Bitcoin and also found correlations. Combining features like technical indicators with social media data could increase the overall scores of the models because you will not only focus on mathematical data, but also on sentimental data. This would be something for future research.

According to Cryptowatch[4] and other researches [22, 8, 7] there are correlations between cryptocurrencies. Positively correlated cryptocurrencies tend to move together, negatively correlated cryptocurrencies move inversely to each other, and uncorrelated variables move independently of each other. One could extend this research by researching correlations between cryptocurrencies and including them in the machine learning algorithm potentially improving model performance.

It might be possible to improve the regression model scores mentioned in this paper. In this paper we made use of predicting the difference in price between the current interval en the next interval. One can try and predict the difference of the price in percentage or try and predict the exact price in the next interval. This way you would have different trained regression models which you can compare in terms of performance and in terms of profit and losses with backtesting.

As described in Section 4.4 we made use of RandomizedSearchCV to optimize our models. We had difficulties implementing Auto-sklearn with our pre-processing component, but a great way to try and optimize our current models would be by using Auto-sklearn to optimize. Another way

[4]Correlations - Cryptowatch. Correlations. `https://cryptowat.ch/correlations`

to optimize our models is to use more resources to find better hyperparameter values. Random search is computationally very costly for machine learning methods. One could use hyperband [3] to improve computation performance in finding configurations within acceptable time.

When backtesting our regression models, we developed a new strategy with the strictness parameter as explained in Section 5.6. We found that setting our *strictness* parameter on 8 would give us the best ROI by trading on that specific time span for our test dataset. We have not retrained our model on different years or months to check if this $strictness = 8$ value would give us the best ROI on these new trading time spans. One could investigate whether there is an optimal *strictness* parameter to increase the overall profit and which one is reproducible.

We have compared all model scores based on ten tests, we did not perform any significant tests. One could use Wilcoxon Rank Test to find significant differences between hyperparameter optimization models.

References

[1] Bergstra, J., Bengio, Y. (2012). Random search for hyper-parameter optimization. The Journal of Machine Learning Research, 13(1), 281-305.

[2] Breiman (2001), L. Random forests. Machine Learning 45, 1, 5–32.

[3] Falkner, S., Klein, A., Hutter, F. (2017). Combining hyperband and bayesian optimization. In NIPS 2017 Bayesian Optimization Workshop (Dec 2017).

[4] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., Hutter, F. (2019). Auto-sklearn: efficient and robust automated machine learning. In Automated Machine Learning (pp. 113-134). Springer, Cham.

[5] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. Efficient (2015) and robust automated machine learning. In Advances in neural information processing systems , pp. 2962–2970.

[6] Greaves, A., Au, B. (2015). Using the bitcoin transaction graph to predict the price of bitcoin.

[7] Ji, Q., Bouri, E., Lau, C. K. M., Roubaud, D. (2019). Dynamic connectedness and integration in cryptocurrency markets. International Review of Financial Analysis, 63, 257-272.

[8] Katsiampa, P., Corbet, S., Lucey, B. (2019). High frequency volatility co-movements in cryptocurrency markets. Journal of International Financial Markets, Institutions and Money, 62, 35-52.

[9] Katsiampa, P. (2017). Volatility estimation for Bitcoin: A comparison of GARCH models. Economics Letters, 158, 3-6.

[10] Kohavi, R., John, G. (1995): Automatic Parameter Selection by Minimizing Estimated Error. In: Prieditis, A., Russell, S. (eds.) Proceedings of the Twelfth International Conference on Machine Learning, pp. 304–312. Morgan Kaufmann Publishers

[11] Liaw, A., Wiener, M. (2002). Classification and regression by Random Forest. R news, 2(3), 18-22.

[12] Lo, A. W., Mamaysky, H., and Wang, J. (2000) Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation. The Journal of Finance 55, 4 , 1705–1765.

[13] Mantovani, R., Horvath, T., Cerri, R., Vanschoren, J., Carvalho, A. (2016): Hyper-Parameter Tuning of a Decision Tree Induction Algorithm. In: 2016 5th Brazilian Conference on Intelligent Systems (BRACIS). pp. 37–42. IEEE Computer Society Press

[14] Matta, M., Lunesu, I., Marchesi, M. (2015). Bitcoin Spread Prediction Using Social and Web Search Media. In UMAP workshops (pp. 1-10).

[15] McKinney, W. (2010), et al. Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference , vol. 445, Austin, TX, pp. 51–56.

[16] McNally, S., Roche, J., Caton, S. (2018). Predicting the price of bitcoin using machine learning. In 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP) (pp. 339-343). IEEE.

[17] Olson, R., La Cava, W., Mustahsan, Z., Varik, A., Moore, J. (2018): Data-driven advice for applying machine learning to bioinformatics problems. In: Proceedings of the Pacific Symposium in Biocomputing 2018. pp. 192–203

[18] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12 , 2825–2830.

[19] Sanders, S., Giraud-Carrier, C. (2017): Informing the Use of Hyperparameter Optimization Through Metalearning. In: Gottumukkala, R., Ning, X., Dong, G., Raghavan, V., Aluru, S., Karypis, G., Miele, L., Wu, X. (eds.) 2017 IEEE International Conference on Big Data (Big Data). IEEE Computer Society Press

[20] Schut. F, van Rijn, J. Hoos, H. (2019). Machine Learning and Technical Analysis for Foreign Exchange Data with Automated Trading (pp.1-33). University of Leiden.

[21] Springenberg, J. T., Klein, A., Falkner, S., Hutter, F. (2016). Bayesian optimization with robust Bayesian neural networks. In Advances in neural information processing systems (pp. 4134-4142).

[22] Stosic, D., Stosic, D., Ludermir, T. B., Stosic, T. (2018). Collective behavior of cryptocurrency price changes. Physica A: Statistical Mechanics and its Applications, 507, 499-509.

[23] Thornton, C., Hutter, F., Hoos, H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Dhillon, I., Koren, Y., Ghani, R., Senator, T., Bradley, P., Parekh, R., He, J., Grossman, R., Uthurusamy, R (2013). (eds.) The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13). pp. 847–855. ACM Press

[24] Valencia, F., Gómez-Espinosa, A., Valdés-Aguirre, B. (2019). Price movement prediction of cryptocurrencies using sentiment analysis and machine learning. Entropy, 21(6), 589.